A Real-Time Transfer Matrix Implementation of Layered Materials

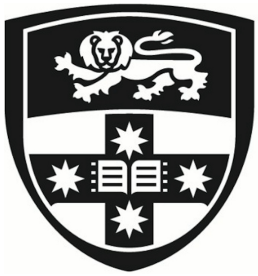
SOREN SAVILLE SCOTT
SID: 500464481

Supervisor: Dr. Masahiro Takatsuka
Associate Supervisor: Dr. John Stavrakakis

This thesis is submitted in partial fulfillment of
the requirements for the degree of
Bachelor of Science Honours (Computer Science)

School of Computer Science
The University of Sydney
Australia

29 January 2025



THE UNIVERSITY OF
SYDNEY

Student Plagiarism: Compliance Statement

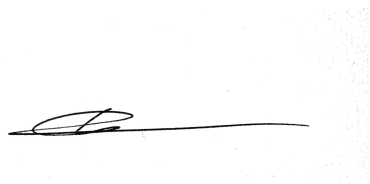
I Soren Saville Scott certify that:

I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure;

I understand that failure to comply with the Student Plagiarism: Coursework Policy and Procedure can lead to the University commencing proceedings against me for potential student misconduct under Chapter 8 of the University of Sydney By-Law 1999 (as amended);

This work is substantially my own, and to the extent that any part of this work is not my own I have indicated that it is not my own by acknowledging the source of that part or those parts of the work.

Name: Soren Saville Scott



Signature: **Date**: 01/11/2024

Abstract

The realistic and efficient simulation of light's interaction with surfaces, known as 'material models' is an open problem in computer graphics. In addition to accuracy and speed concerns, the primary users of these techniques in the film, TV and gaming industries require that these material models are both artistically flexible and intuitive, for use by non-technical users such as artists and designers. Material Layering is a technique where surfaces are defined by composing together individual material models into stacks, and light is then simulated through the stack. This adds a third dimension to the simulation of materials, allowing for complex interactions such as intra-layer scattering. Additionally, material layering adds more flexibility to the representation of surfaces, by allowing artists to design a surface by composing different materials together. Material Layering is the standard framework used in contemporary offline renderers, however real-time renderers have historically lagged behind, and there are open problems in simplifying material layering models and managing the complex interactions they produce in a scalable way, in order to make them appropriate for real-time contexts and performance constraints. This research aims to investigate the performance characteristics and suitability of a transfer-matrix based layered material model for real-time rendering on contemporary consumer hardware.

In this research, a real-time implementation of Randrianandrasana, Callet, and Lucas's 2 and 6 flux transfer matrix models presented in *'Transfer matrix based layered materials rendering'* (2021) is produced, fitting it to preintegrated lighting. This is validated against the path-traced reference provided in Randrianandrasana et al.'s paper by reproducing and comparing layer stacks. The performance of the real-time implementation is profiled on contemporary hardware and an in-depth analysis of the hardware utilisation characteristics is provided, as well as an exploration of the impact of various optimisations present in the literature. This research finds that the transfer-matrix model is architecturally unsuitable for real-time renderering on contemporary hardware, due to its large size and difficulty in scaling performance characteristics to desired features. Although the high cost as-is was expected, being bottlenecked by size was unexpected and made the effect of applying optimisations far less significant than anticipated. This unsuitability makes other layering approaches discussed in this paper more attractive for real-time rendering.

Acknowledgements

Thank you to my family for their support over the course of completing this thesis.

Thank you to our cat, Smudge, for lying around.

CONTENTS

List of Figures

List of Tables

CHAPTER 1

Introduction

FIGURE 1.1. The 'Stanford Dragon'(Stanford, 1996) rendered with a layered material using the model developed in this research.

Recent years have seen a proliferation in the use of real time rendering technologies across many industries, such as film & television or visualisation for design, education and architecture. The 'visual gap' in fidelity between real time renderers and their offline counterparts has shortened dramatically, making the technology appropriate for use in final productions, rather than as an R&D or pre-production tool. This has allowed studios to leverage the instant feedback and interactive nature of these renderers to shorten their design iteration and increase productivity. This means that users are increasingly demanding more accuracy and flexibility from these renderers.

Unfortunately, the technologies employed in representing surface materials in these renderers lag behind their offline counterparts. Artists are often restricted to a fixed set of models that have deep dependencies on the internal architecture of the renderer, requiring skilled and labour intensive engineering work to extend. Borrowing from offline renderers, real time renderers are starting to incorporate graph based layering and mixing of material models in order to extend the set of representable materials and give artists more freedom. This introduces a new problem: the combination of layers becomes computationally complex to store and evaluate for the final lighting output, and so methods of simplifying these topologies while preserving their correctness and visual fidelity are needed.

This paper presents a review of the existing state of the art of material layering, investigating the history of layering as a technique and the approaches contemporary production renderers are taking. Some experimental approaches are also discussed. I produce real-time implementations of two variants of a transfer matrix based layered material model, based on research by Randrianandrasana et al. in '*Transfer Matrix Based Layered Materials Rendering*' (2021). Modifications are made to fit the model to a preintegrated, rasterised rendering context, and the real-time implementations are validated against the original paper's path-traced implementation for correctness. An analytical approximation for Total Internal Reflection is developed in order to improve the performance of the model. I investigate the real-time model's usability, performance and usefulness as a general material layering framework for production use.

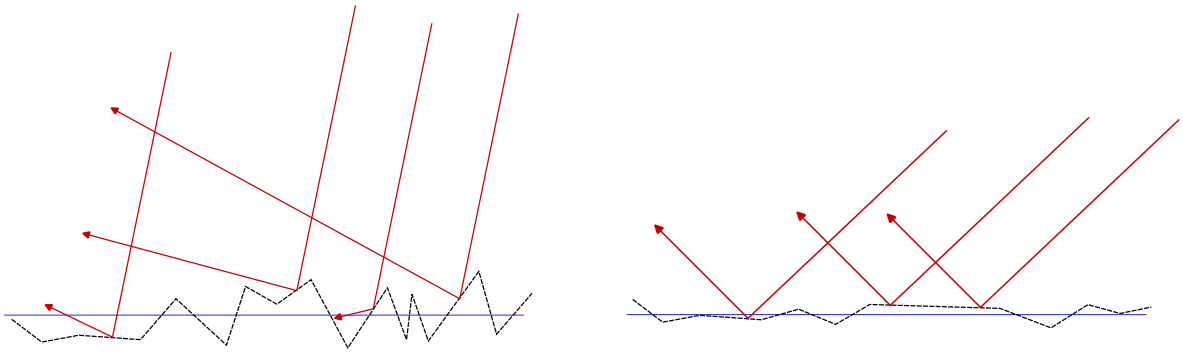
CHAPTER 2

Background

There are some foundational concepts in rendering that are important for - but not directly related to - this research. This includes the general theory of light-surface interaction (the 'material', 'shading model', or some combination of those terms), including the microfacet model that underpins much of contemporary research and production use. Alongside this, there is the theory of light transport with respect to rendering, which describes how light is propagated throughout a scene.

In 3D rendering, surfaces are defined by a 'material', a model and parameters that defines how that surface interacts with incoming light rays - how they should be reflected, scattered, transmitted, and so on. The general form for a material model is the BxDF - The Bidirectional (something) Distribution Function, with the 'x' denoting the particular class of the model. For example, the Bidirectional Reflectance Distribution Function (BRDF) accounts for all reflective interactions with the surface, while the Bidirectional Transmission Distribution Function accounts for all transmissive interactions, where rays transmit *through* the surface, and out the other side. This can be generalised to a Bidirectional Scattering Distribution Function, which composes both the BRDF and BTDF together into a model handling both reflection and transmission interactions with a surface. As it is a distribution function, the BxDF returns the ratio of energy for a given input ray and output ray, and material parameters (Pharr & Humphreys, 2023).

Historically, material models have been phenomenological (Guarnera, Guarnera, Ghosh, Denk, & Glencross, 2016), aiming to replicate the visual phenomena of light's interaction with a surface, with little consideration for the underlying physics. This was largely due to computational constraints, coupled with a low demand for physical accuracy. Canonical examples of such models are the Phong and Blinn-Phong BRDF, which is simply the dot product of the halfway vector between the incoming and outgoing directions and the surface normal, raised to some exponent modelling shininess (Blinn, 1977).



(a) Rough Microfacet Distribution (b) Smooth Microfacet Distribution

FIGURE 2.1. Conceptual illustration of the microfacet model of reflection. The noisier distribution of microfacets in (a) vs (b) produces a rougher surface appearance

Contemporary renderers however typically model surfaces using some variant of the microfacet model. This is a physically based family of BRDFs that models surfaces as a field of micro-scale perfectly reflective facets, facing in random directions according to some Distribution of Normals Function (NDF) (Heitz, 2014). Conceptually, incoming light rays interact with individual microfacets, reflecting at 90 degrees (according to the law of specular reflection) to the facet normal. The surface appearance is thus defined by the aggregate behaviour of the relative variation of the facing direction from one facet to the next - a smoother surface will have a more uniform distribution of normals, and so rays will be scattered less, all reflecting in the same direction. Conversely, rough surfaces will have a very wide distribution of normals, with very few microfacets facing the same direction, and so the reflected image will be blurry and diffuse. The overall microfacet BRDF as originally derived by Cook and Torrance is defined as:

$$f(\omega_i, \omega_o, \alpha) = \frac{F(\omega_o, \omega_h) G_2(\omega_o, \omega_i, \omega_h, \alpha) D(\omega_h, \alpha)}{4|\omega_g \cdot \omega_o||\omega_g \cdot \omega_i|} \tag{2.1}$$

Where ω_i is the incident direction, ω_o is the outgoing direction, α is the surface roughness, ω_g is the geometric normal, and ω_h is the half vector between ω_i and ω_o. D is the Distribution of Visible Normals Function described earlier, where the most common distribution used is the GGX formulation, however others such as Beckmann and even Blinn-Phong may be used. The GGX - or 'Ground Glass Unknown' (Walter, Marschner, Li, & Torrance, 2007) - distribution is generally preferred due to its softer falloff in its impulse curve, corresponding to a more natural tail in specular highlights. The microfacet model further accounts for the fresnel effect via the fresnel equations F, where specular energy increases as the incident angle gets shallower (approaches 0). The masking-shadowing G_2 term attenuates the

distribution by the facets that are visible and not in shadow to the viewer, preventing leakage from rays travelling through other facets. This is typically a single-scattering formulation, where only a single reflection off a microfacet is accounted for. Any rays that would bounce off other microfacets after the first reflection are considered to be masked by the function (Heitz, 2014). This is physically based as the model is tied directly to physical laws such as specular reflection and the fresnel equations. Furthermore, it is reciprocal and energy conserving: the result of evaluating the function is the same if ω_i and ω_o are swapped, and the total outgoing energy ratio is never more than 1 (i.e, it never creates energy) (Pharr & Humphreys, 2023).

Computing the final rendered colour of a sampled surface point means solving the 'Light Transport Equation' (Pharr & Humphreys, 2023):

$$L_o(\omega_o) = \int_\Omega f(\omega_o, \omega_i) L(\omega_i) |\cos \theta_i| d\omega_i \tag{2.2}$$

This is the integral over the hemisphere of incident directions Ω of all incoming light, attenuated by the surface BxDF f for the given outgoing direction ω_o. The equation given above is a special case of the full 'rendering equation', accounting for a single light-surface interaction. The full rendering equation includes a term for the overall scene description and the associated occlusion and multiple bounce interactions that come with it. For this research an understanding of a single light-surface interaction is sufficient, while being simpler to work with and understand. The single-interaction case can be readily generalised to a full scene description by repeated application of the function, typically along the bounces of a ray. In the specialised case given in Equation 2.2, the $L(\omega_i)$ function describes the light energy coming from some arbitrary emitter. This can be a surface radiating energy into a scene, such as a sun or lamp, or it can be the result of a reflection or transmission event from some other surface. The Light Transport Equation, in general, has no closed solution, and so many renderers solve it via Monte Carlo methods, where the function is sampled stochastically (often using some prior knowledge about its distribution) to converge on the final result. There also exist specialisations of the L function that make evaluating light transport simpler in certain scenarios, such as point sources which model a light emitter as an infinitesimally small point, collapsing the integral down to a Dirac delta which can then be evaluated just once at a single direction (Mack, 2007). Other approaches include pre-integration, where the integral is precomputed into a look-up table with respect to some light source and material model. The latter approach is typically the only option available for real-time lighting with arbitrary emitter geometry, as computing the integral at run-time is slow, costly and often comes with an additional degree of repeated and wasted work. Arbitrary emitter geometry is required for Image-Based Lighting, where

high dynamic range images are used as emitter surfaces, contributing to highly realistic environment and ambient lighting (Lagarde & De Rousiers, 2014).

CHAPTER 3

Literature Review

This is a review of the existing literature of material rendering & layering, across offline path traced implementations and real time rasterizers. The correctness and flexibility of the approaches given in each reviewed paper is analysed, and judged with respect to their actual or potential performance in a real time context. An overview of the existing, fixed model material approach is given to provide a benchmark for the layering techniques discussed - any candidate must be at least 'as good' as this benchmark, in terms of visual correctness and flexibility. A brief history of material layering research is explored, covering Weidlich and Wilkie's foundational research with optically thin coatings and the current state of material layering in production contexts with Hillaire and De Rousiers's *Substrate* framework as used in *Unreal Engine 5*. A brief exploration of novel approaches, such as Guo et al.'s neural layering framework is conducted. Jakob, d'Eon, Jakob, and Marschner's Fourier tabulation of arbitrarily layered surfaces is covered as it later informs the approaches used in Belcour and Randrianandrasana et al.'s layering frameworks. The latter approach forms the basis for the real-time layering model developed in this research.

3.0.1 Current Material Models

The majority of real time renderers currently use some extension of Disney's 'Principled' shading model, developed in 2012. This model encapsulates both specular and diffuse surfaces, able to accurately represent the vast majority of dielectric and metallic surfaces (Burley, 2012). This also saw the introduction of 'Physically Based' shading as a guiding principle for shading models in real-time renderers, meaning that they should have properties that are grounded in physics such as reciprocity and energy conservation. This model largely superseded the collection of so-called 'Phenomenological' models used up until that point, such as Blinn-Phong, which attempt to simply emulate visual phenomena with little to no consideration for the underlying physics (Guarnera et al., 2016).

Disney's principled model is based on the microfacet model, which models surfaces as a discrete set of microscopic undulating faces (a microfacet) that reflect incoming light rays according to the microfacet normal. Under Disney's model, the specular energy is assumed to be constant over the entire surface. Surface appearance is based on the uniformity of its microfacets, with smoother, glossier surfaces having a uniform distribution of face normals, while rough surfaces have a highly decorrelated distribution - resulting in highly scattered reflection rays - essentially 'blurring' the reflected image and making it appear rougher (Pharr & Humphreys, 2023).

The Disney model's popularity owes much to its flexibility and intuitive design. The model, particularly in its Metal/Rough parameterisation, presents to artists a fixed (and small) set of combinable, normalised parameter ranges. Any permutation of values within these ranges is guaranteed to be 'physically accurate', at least according to the assumptions made in the model. Further, the Metal/Rough parameterisation is incredibly intuitive, allowing artists to 'sight-read' real surfaces by asking whether it is a conductor or a dielectric, and whether it is smooth or rough. This combination of accuracy and intuitive design allowed users to quickly iterate on materials during art passes, and eliminates much of the 'do-overs' during lighting suffered by previous productions (Burley, 2012). This highlights that a viable successor needs to not only allow for more accuracy across a wider set of materials, but should also preserve the same user experience philosophy as Disney's model. This observation is not obvious, and many models in the literature suffer from poor flexibility and user experience.

The papers *Real Shading in Unreal Engine 4* (Karis, 2013) and *Moving Frostbite to Physically Based Rendering* (Lagarde & De Rousiers, 2014) provide a detailed overview of the work conducted to fit the Disney Principled model - hitherto an offline model - to industry standard real-time renderers - *Unreal Engine 4* and *Frostbite* respectively. There is a comprehensive breakdown of the work conducted to fit the Disney model to a preintegrated lighting context, alongside analysis and discussion of several approximations for components of the microfacet model, such as the F, G and D terms. The work done in these papers form the foundations for much of the wave of games and engines making use of 'Physically Based Rendering' (PBR) that occurred over the next 5 years. The significant contributions are in the development of new approximations for various light types, avoiding expensive Monte Carlo integration. These include point and spot lights, area lights and uniform directional 'sun' lights. Karis introduces a technique for preintegrating environment lights (IBL probes) with respect to the Disney microfacet model. This is known as the 'split-sum' approximation, as the numeric integration of the Light Transport Equation (Equation 2.2) is split into two sums that closely approximate the original

sum:

$$\text{LTE} \approx \frac{1}{N}\sum_{k=1}^{N}\frac{L(\omega_k)f(\omega_k,\omega_i)\cos\theta_k}{p(\omega_k,\omega_i)} \tag{3.1}$$

$$\approx (\frac{1}{N}\sum_{k=1}^{N}L(\omega_k))(\frac{1}{N}\sum_{k=1}^{N}\frac{f(\omega_k,\omega_i)\cos\theta_k}{p(\omega_k,\omega_i)}) \tag{3.2}$$

Where p is the probabilities distribution function of the microfacet BRDF f. This 'split-sum' essentially divides the Light Transport Equation into a light term and a BRDF term, the first measuring just the contribution of the light over the hemisphere of view angles, and the second the BRDF's microfacet response in a pure white furnace - essentially its directional albedo. The first term may be precomputed in its entirety with respect to the IBL, while the second term can be precomputed in-part by factoring out the F Fresnel term from the Microfacet BRDF. This second precomputation is generally referred to as '*the* Split-Sum LUT', even though it only refers to the second term of the split-sum. In this way, both the BRDF and IBL have been pre-integrated. One of the main assumptions made in the split-sum approximation is a fixed view angle. This means that the anisotropic lengthening of reflections seen at grazing angles is lost. Lagarde and De Rousiers note this issue in their IBL preintegration as well. They introduce a function that approximates the lobe-shift of the GGX vNDF over roughness, which helps bring back some of that anisotropy to reflections - although their IBL pre-integration is still done with the fixed angle assumption. Lagarde and De Rousiers also introduce a technique for pre-integrating lighting with respect to their diffuse model. As Karis use a simple Lambertian model for their diffuse implementation, preintegration is not necessary and so is absent from their paper. These papers are extremely useful references for implementing and validating microfacet BRDFs in real-time renderers, and the techniques developed continue to be used and built upon extensively.

3.0.2 A Short History of Material Layering

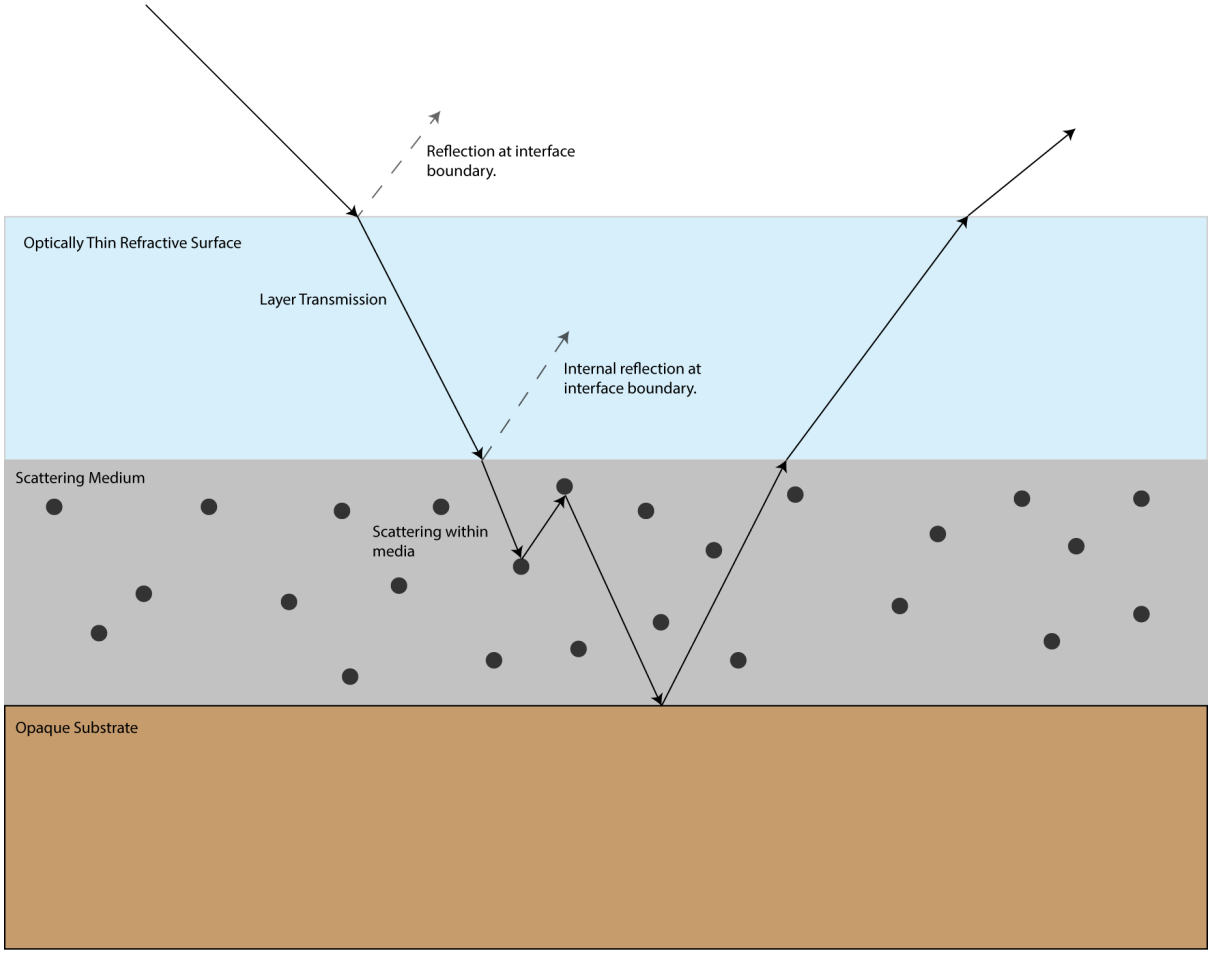


FIGURE 3.1. Conceptual illustration of a ray being traced through a stack of material layers.

Material Layering is not a particularly new concept in computer graphics. Offline renderers have been using it in a production context for many years now, such as Renderman's 'PxrSurface' model that forms the primary shading paradigm in use at Disney/Pixar animation studios (Hery, Villemin, & Ling, 2017). Material Layering has seen little use in real time renderers due to its higher computational complexity, memory cost, and being seen as poorly suited architecturally to a rasterization pipeline. As opposed to the 'ubershader' formulation used in the Disney Principled model that seeks to capture a wide range of surfaces under a single parameterisation, material layering approaches consist of a library of specific models for certain surface effects, that are then layered on top of each other with blending operators in a graph. This emulates the way surfaces are created in real life, as layers of different physical media.

The final ‘material’ then is derived from the interaction between the different layers as light rays are absorbed, reflected or transmitted through them to varying degrees, based on their properties.

Early experiments with material layering, such as Weidlich and Wilkie’s *Arbitrarily Layered Microfacet Surfaces* (2007), conceived of layering as a way to employ the existing ‘basket’ of phenomenological BRDFs in the literature at that time for more accurate material representations. The primary use-case explored is that of glossy dielectric, transmissive layers on top of rough and/or diffuse substrate, such as varnish on clay. This use case is readily covered by Disney’s Principled model, however the paper is a good foundational basis for layering in practice, due to its simple and straightforward approach. The approach limits itself to a BRDF - it does not account for scattering between layers, as would happen in reality. The model assumes that all incoming rays exit at the original point of incidence, and that refraction rays all meet at the same point on the interface of the next layer. This requires that layers be optically thin, and the lack of scattering simulation does not allow for the characteristic blurring or refraction of lower layers exhibited by some materials, like frosting. Evaluating the layers for the final BRDF can be done recursively, in what is essentially a linear sum (with energy attenuation as appropriate). This makes the evaluation of a BRDF largely dependent on the number of layers and the complexity of the individual BRDFs for each layer, which is about as good as one can get - the overhead from the layering itself is minimal. The approach given is targeted towards a Monte Carlo renderer, however the assumptions made also make evaluating the model convenient for GPU rasterizers, as the independence between sample points is well suited to their massively parallel nature. It is conceivable that this approach could incorporate the Disney Principled model as one of its layer options, which would yield a natural extension of the principled model by allowing for several lobes for e.g. several clearcoat layers, rather than the fixed, single simulated layer offered by the Principled model. The method given for computing the probability distribution function of the composition of layers is concerning, as it appears to be dependent on assigning a weighted sum to each BRDF based on which layer in the stack ‘dominates’. Computing the dominating layer is a non-trivial task and may limit the freedom users would have in developing their own layer stacks.

A less often explored option for storing and evaluating layered materials is to work inside a Fourier basis. In this formulation, each layer can be understood as a signal, with the final lighting evaluation based on the composition of each signal. This would allow for any model to be incorporated into any layer stack, so long as it can be expressed in the Fourier basis, thus eliminating the reliance on the microfacet model used in Weidlich and Wilkie’s approach, and the principled ‘slab’ used in the Substrate framework,

or any other assumptions fixed into a given model. Jakob et al. present a method for storing layered materials in a Fourier basis in *A Comprehensive Framework for Rendering Layered Materials* (2014). The fundamental approach taken is to discretise continuous radiance functions along a Fourier basis, encoding scattering coefficients along azimuth and elevation angles. The paper presents an algorithm for constructing the reflection and transmission matrices of each layer via an adding-doubling method, and then composing them together into a final scattering matrix, from layer phase functions, depth and boundary BSDFs. It is assumed that the algorithm is run offline, in something of a 'compilation' step, storing the discretised result in a sparse matrix. At render time, the Fourier expansion for the given light ray directions (as in a standard BSDF) is summed over the azimuth angles (with interpolation) between the directions provided, producing the final light-transport values. The stored matrices are incredibly large, particularly for highly glossy surfaces that require very small and tight peaks in their Fourier series, requiring the storage of many coefficients. With the sparse data structures employed, models are typically in the 10s of megabytes, which is unacceptably large. This is in spite of their incredibly high compression ratios with respect to their dense equivalents (usually less than 1%). These 'compilation' runtime and storage characteristics are difficult to justify, even in an offline context. A potential avenue of exploration is whether this time and size can be reduced by removing the need to discretise and precompute the scattering matrices in order to lookup the entire BSDF at render time, and instead working in the Fourier basis throughout the entire pipeline. Overall this approach has seen zero use in production contexts, however the techniques developed, such as the adding-doubling algorithm and the Fourier tabulation of scattering components inform a considerable amount of later research, and this paper is considered to be foundational.

3.0.3 Real-time Layering Frameworks

Realtime implementations of material layering have begun to emerge over the last five years. Notable examples include Belcour's layered materials used in the Unity HDRP renderer (2018), and Unreal Engine's 'Substrate' framework, presented at SIGGRAPH in 2023 (Hillaire & De Rousiers, 2023). It is important to distinguish these implementations from earlier work also referred to as 'material layering', such as that described in Epic Game's *Real Shading in Unreal Engine 4* (Karis, 2013) and Burley's paper on the Disney Principled model. This should be more accurately thought of as attribute or parameter layering, as it only provides a convenient user interface for blending different parameter sets of the same underlying shading model. That is to say, any 'layer' stack in this framework can be - and is after compilation - equivalently expressed by just a single layer, and so the layering does not extend

the representable set of surfaces, nor does it provide any more surfacing flexibility over the single-layer model.

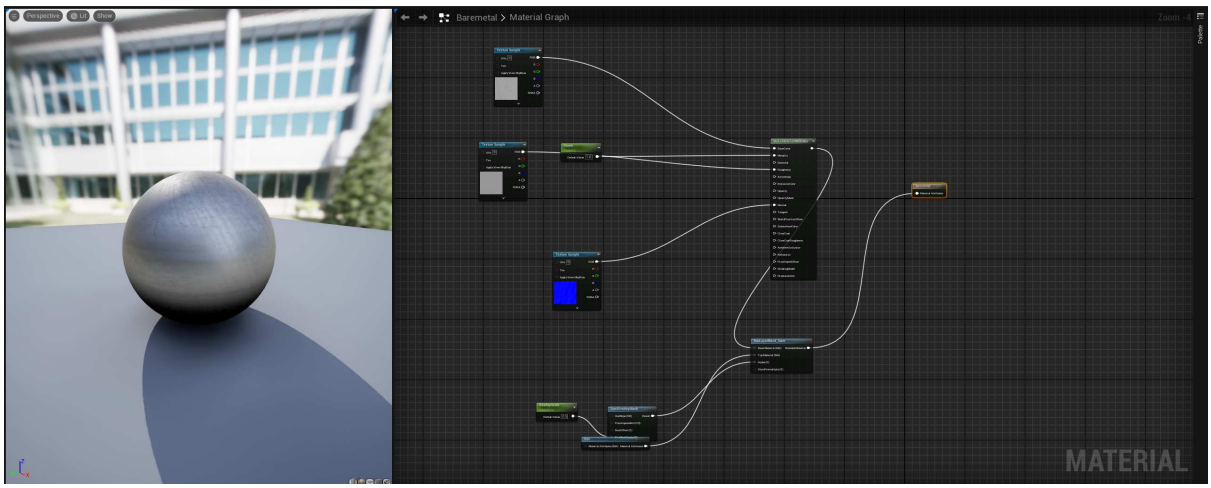


FIGURE 3.2. An attribute blend consisting of a dielectric Dirt layer over the top of a metallic base layer.

```
PixelMaterialInputs.EmissiveColor = Local25;
    PixelMaterialInputs.Opacity = 1.00000000;
    PixelMaterialInputs.OpacityMask = 1.00000000;
    PixelMaterialInputs.BaseColor = Local32;
    PixelMaterialInputs.Metallic = Local39;
    PixelMaterialInputs.Specular = 0.50000000;
    PixelMaterialInputs.Roughness = Local43;
    PixelMaterialInputs.Anisotropy = Local45;
    PixelMaterialInputs.Normal = Local2.rgb;
    PixelMaterialInputs.Tangent = MaterialFloat3
        (1.00000000,0.00000000,0.00000000);
    PixelMaterialInputs.Subsurface = 0;
    PixelMaterialInputs.AmbientOcclusion = Local49;
    PixelMaterialInputs.Refraction = 0;
    PixelMaterialInputs.PixelDepthOffset = 0.00000000;
    PixelMaterialInputs.ShadingModel = 1;
    PixelMaterialInputs.FrontMaterial = GetInitialisedStrataData();
    PixelMaterialInputs.SurfaceThickness = 0.01000000;
```

```
    PixelMaterialInputs.Displacement = 0.00000000;
```

LISTING 3.1. Snippet of intermediate shading language (HLSL) compiled from the above layer graph (Figure 3.2). Only a single layer is being output, each parameter is a linear blend of the parameters in the base and overlay layer.

The Substrate approach can be considered 'true' material layering, as it attempts to simulate physical layer interactions in much the same way as the aforementioned Weidlich and Wilkie layering, or offline layer frameworks such as Renderman's PxrSurface.

The fundamental realisation made in Substrate is that the limiting factor in the efficiency of material layering is the number of layers evaluated, and the size in memory of layer stacks. There are practical concerns around minimising thread divergence and memory contention for efficiency on contemporary graphics hardware (Laine et al, 2013) that indirectly influence the design and implementation of the framework too. Therefore, Substrate aims to minimise, for any layer stack constructed in the framework, the number of layers that the renderer actually has to store and evaluate. This is done by fixing the set of layerable models, and their operators, in such a way that allows for convenient and efficient static and dynamic simplification of topologies. The fundamental model is the 'Principled Slab', which can be thought of as an evolution of the model based on the Disney Principled BRDF used in earlier versions of Unreal Engine (Karis, 2013), extended to a true BSDF accounting for transmission and scattering. The parameterisation is similar to the interface provided by the Disney Principled model, divided into feature categories that may be disabled or enabled as desired. For example, subsurface scattering and retroreflection are considered 'complex' features. These categories are important for the simplification passes.

These 'slabs' may be layered together with one of 3 operators:

- A vertical layering operator, that places a slab on top of another with a defined thickness. Light will transmit from the topmost medium down to the bottom, reflecting at media boundaries according to the transmission and reflection properties of each slab.
- A horizontal blending operator, that blends 2 slabs together horizontally by weight, used for transitioning between different materials over the area of a surface.
- Finally, a coverage operator is defined to allow for translucent blending of layers.

This forms a tree topology of operators, with individual slabs at the leaves. Evaluation of the material is done by walking up the tree, propagating closure properties such as transmittance, reflectance and coverage.

For complex topologies this results in a large number of closures that need to be evaluated for a given sample, which is intractable (Hillaire & De Rousiers, 2023). Instead, multiple slabs may be virtualised into a single layer via a statistical representation using weighted sums, returning a single closure composed of the weighted sum of its constituent closures. Coverage and horizontal blending can be weighted by their blend weights directly, while vertical layering evaluates a probability tree to compute the resulting weights. Similarly, horizontal blends between slabs sharing the same feature categories can be simplified into a single slab by reducing the layer blend to the aforementioned attribute blend technique, producing a virtual slab that is simply the linear interpolation of the source slabs' parameters. It is important to note that these simplifications are only possible due to the fixed slab model that allows assumptions to be made about how different layers interact with each other. Static analysis of the material topology can also be done to prune out non-contributing nodes at compile time via constant propagation and folding (e.g. a slab with a constant coverage of 0 can never contribute to the final material). This produces a final material that can be scaled to performance and memory requirements, by applying simplification passes until within budget, and by forcing layers down to specific feature categories where necessary. The former option preserves layer interactions as much as possible, while the latter does not (that is, information is lost in the simplification), but is necessary for scaling to different hardware capabilities. The main limitation of this framework is that it requires its library of slab types to conform to the assumptions made by the layering operators and the simplification passes, which greatly limits the ability of artists to incorporate new, custom shading models.

Belcour in *Efficient Rendering of Layered Materials using an Atomic Decomposition with Statistical Operators* (2018) provides an approach for layered materials based on the composition of several GGX lobes, using a set of statistical operators developed in the paper. An offline and real time implementation is provided, and this model is employed for layering in the Unity game engine's "High Definition Render Pipeline" (HDRP), targeted at film, TV and virtual production users. Each layer in the model is paramaterised by a tuple of complex index of refraction (IOR) and interface microfacet roughness. The complex IOR term is described as $\eta + i\kappa$, where the η term describes the bending of light as rays pass through the layer's interface boundary, and the κ term describes any absorption of light through the same medium. A non-zero κ term essentially describes a 'conductor', or metallic material, whereas

a zero κ term describes a dielectric material, where light is able to transmit through to the next layer. The paper decomposes light transfer into statistical operators for reflection, refraction, scattering and absorption, applied on a collection of lobes representing the layers in the stack. The paper builds on the work of Jakob, d'Eon Jakob & Marschner (2014) in the use of the adding-doubling method to evaluate layer stacks. The chosen representation for the lobes is lightweight, which allows the layer evaluation to remain small, important for fighting against the memory consumption from the large number of interactions produced by several layers, and in particular for maximising throughput on GPU architectures, as lower register use allows the hardware scheduler to dispatch more threads simultaneously. The restriction to GGX lobes means that diffuse (e.g. Lambertian) surfaces are not properly represented, and the model has significant error at high roughnesses. This largely limits the model to simulating forms of coated metal, such as seen in many car paints, which have dielectric clear-coating over metallic substrates.

3.0.4 Novel Layering Approaches

Randrianandrasana et al. in *Transfer Matrix Based Layered Materials Rendering* (2021) build on Belcour's framework by replacing the adding-doubling method of layer evaluation with a transfer-matrix approach, where each layer is represented by a matrix, and the final BSDF is the matrix product of the layer matrices in sequence. The approach extends Belcour's model by introducing multiple scattering for arbitrarily thick layers, important for natural and realistic participating mediums that don't suffer from energy loss. The transfer matrix approach decouples allows for evaluating arbitrarily thick layers in constant time, an improvement over the adding-doubling method, which has to decompose arbitrarily thick layers into arbitrarily many virtual stacks of optically thin layers. The transfer matrix approach can be considered an approximate version of Jakob et al.'s scattering matrices, in which the scattering properties are summarised by Henyey-Greenstein lobes rather than the full angular representation. Randrianandrasana et al. provide two variants of the transfer matrix approach: a 2-flux model, built on a 2x2 matrix accounting for reflected and transmitted light at interface boundaries, and a 6-flux model, that accounts for the same properties as the 2-flux model, alongside backscattering, intra-layer scattering and re-emission. This effectively allows for 36 degrees of freedom for a given incident light ray - 6 forms of 're-emission' at the top interface (either by reflection or scattering), and 6 forms of transmission at the lower interface. The 2-flux model can be considered approximately equivalent to the model proposed by Weidlich and Wilkie, however with the addition of accounting for rough interfaces and rough refraction. The 6-flux model correctly accounts for multiple scattering within the layer media. In order

for the transfer matrix approach to be viable, the authors work with the Henyey-Greenstein phase function for scattering. This allows them to make use of the functions stability under convolution, which microfacet GGX lobes do not possess. The authors provide an approximate transformation from GGX roughness to Henyey-Greenstein asymmetry for the purposes of the model parameterisation, which reuses the parameterisation used by Belcour. The model's conceptual simplicity is attractive for practical implementations, as all of the statistical operators described in Belcour's paper are expressed singularly in matrix-multiplication. The fixed-time cost is also attractive from a performance standpoint, although this assumes that in real terms the matrix-multiply is at least as fast or faster than the adding doubling method used by Belcour, on contemporary hardware.

Novel approaches to generalised material layering seek to eliminate precomputation by training neural networks to learn the BSDF function for a layer stack. Guo et al. in *Metalayer: a Meta-learned BSDF Model for Layered Materials* (2023) presents an approach for learning a neural representation of layered materials, producing an efficient and compact model that can be generated and evaluated quickly, with a learned mapping from the artist-facing parameterisation to the underlying neural weights. The paper attempts to resolve the tension between large, generalised, accurate but low performance neural models, and small, efficient but low accuracy models suited for runtime evaluation through a 'meta-learning framework'. Essentially, an arbitrary layer stack may be encoded into the weights of a network modelling a generalised BSDF (dubbed 'BSDF-net'). A second network, 'MetaNet', is trained to produce weight-assignments for BSDF-net, corresponding to the desired layer stack. The 'MetaNet' acts as a translation layer, allowing it to be trained for a designed parameterisation of the model, that does not necessarily have to map directly to the underlying BSDF. This gives renderer authors freedom in designing the user experience of their layering framework. Difficulties arise with this approach in producing a comprehensive enough training dataset for 'BSDF-net', as with all machine learning approaches the model is only general to the point that its source dataset allows it to be. In the case where users need a surface that cannot be represented by BSDF-net, this may increase the cost of extending the model, as new training material would need to be collected or synthesised, and the network re-trained. Inherent to machine learning is also some degree of error, which means that it can not be guaranteed that model results are energy conserving, or otherwise correct. The authors note that this does not seem to be a noticeable problem in practice, and the degree of error is similar to the other approaches discussed.

CHAPTER 4

Methodology

In this research, I develop a real-time implementation of the 2 and 6 flux transfer matrix models (TM2 and TM6 respectively) described in Randrianandrasana et al.'s *Transfer Matrix Based Layered Materials Rendering*.

This research has 2 core components:

- Fitting the Transfer Matrix model of layered materials to a real-time rendering context.
- Detailed performance analysis of the implementation and potential optimisations.

Ultimately, the goal is to present a proof-of-concept for the transfer matrix model in a realtime context, which means that it must be demonstrated to work in a real-time renderer, and work within real-time performance constraints.

A 'real-time renderer' means a rasterisation engine running on contemporary consumer graphics hardware. This will use one of the available hardware APIs for interfacing with consumer graphics hardware such as DirectX, Vulkan or OpenGL. Lighting will be pre-integrated, and the model is modified in order to work in this context. These choices match the rendering architecture of contemporary industry-standard real-time renderers, such as Epic Games' *Unreal Engine*, Unity's *Unity Game Engine*, and EA's *Frostbite* engines. For simplicity, I implement the model in a forward-rendered context, where geometry is rasterised and shaded in a single forward pass. This is contrasted against a 'deferred' renderer, where intermediate buffers containing pre-lighting attributes are written out in screen-space, and lighting is then evaluated in a separate pass against these screen buffers (Naty, 2009). Conceptually, this is an optimisation applied to eliminate issues with wasted work redrawing geometry ('overdraw'), allow for re-use of intermediate data in post processing effects, and trade extra render passes for smaller (and thus, ideally, faster) shaders. In theory there is little conceptual difference with respect to how material models are evaluated and lighting is calculated, and so, along with the fact that many real-time deferred

renderers also have support for forward paths, the flexibility and ease of implementation offered by forward rendering is an appropriate decision for this research.

There are 2 main success criteria for this research: correctness and speed. The former is more important than the latter, however given that realtime rendering is the target, it should be demonstrated that the approach at least appears to be feasible for a realtime renderer. This essentially requires a proof-of-concept to be developed and profiled on some reference hardware, and that its rendered frame time is shown to be within, or reasonably close to real time constraints. In a perfect case, this means a few milliseconds for the layer evaluation, but tens of milliseconds would also be acceptable if it can be demonstrated that further work would be capable of reducing this time.

Correctness can be evaluated in a theoretical sense against 'physically based rendering' principles, e.g. by proving properties such as energy conservation and reciprocity (Hoffman, Gotanda, Martinez, & Snow, 2010). The traditional technique for validating energy conservation is known as 'the furnace' test, where a sphere with the material model is placed in a scene, surrounded by a uniform constant emitter, usually white or grey. If the model is energy conserving, it should reflect and/or transmit exactly the amount of light incident, and so should appear to be 'invisible' against the uniform backdrop. The use of a sphere is significant as it captures the entire range of possible view angles. The presence of pixels brighter than the background implies excess energy being generated, and conversely darkening implies energy being lost. This test is used extensively in the literature to benchmark the implementation of a particular shading model[1]. The implementation will also be evaluated by comparing rendered frames in the real-time implementation against the author's provided implementation for Mitsuba, a ground-truth renderer with path tracing. If the image frames compare equal to a given significance threshold (margin to account for noise and slight implementation differences) the real-time render may be considered 'correct'. This is the method used to evaluate correctness in much of the existing literature, including Randrianandrasana et al.'s original paper.

In terms of speed, this is defined as the time to execute the transfer matrix pixel shader across a full-screen 1920x1080 render, on some contemporary desktop hardware. This definition eliminates unrelated variables that may contribute to the *overall* performance of a renderer, such as rasterisation time, post processing effects, and so on. Profiling statistics for the hardware will also be analysed, such as the implementation's register pressure, occupancy, memory usage and throughput, again isolated to just

[1]The origins of the 'furnace test' are murky and without citation in graphics literature, although the intuition is sound. It may have a foundation in radiometric testing.

the transfer matrix shader invocation. Using this profiling data, optimisation approaches are then explored and their frame improvements contrasted against tradeoffs in terms of their approximation error. Understanding the performance of shaders on contemporary hardware is somewhat byzantine, as it is influenced by a wide variety of factors that are interdependent and subtle. Without a full discussion of GPU and parallel programming architecture, as other authors have done far more competently[2], shader performance is a matter of improving throughput on the system. This can be done through some combination of dispatching more invocations of the shader in parallel and making individual shader dispatches smaller and faster. These factors are in turn influenced by the number of registers required for a given shader invocation, the size, layout and number of dependent memory resources such as textures (as well as how effectively their latency can be hidden), keeping thread divergence within a warp low, as well as more traditional serial optimisation principles such as cache-friendly programming and exploiting instruction-level parallelism and pipelining.

[2]See *Megakernels Considered Harmful* (Laine, Karras, & Aila, 2013)

CHAPTER 5

Results

5.1 A Real-time Implementation

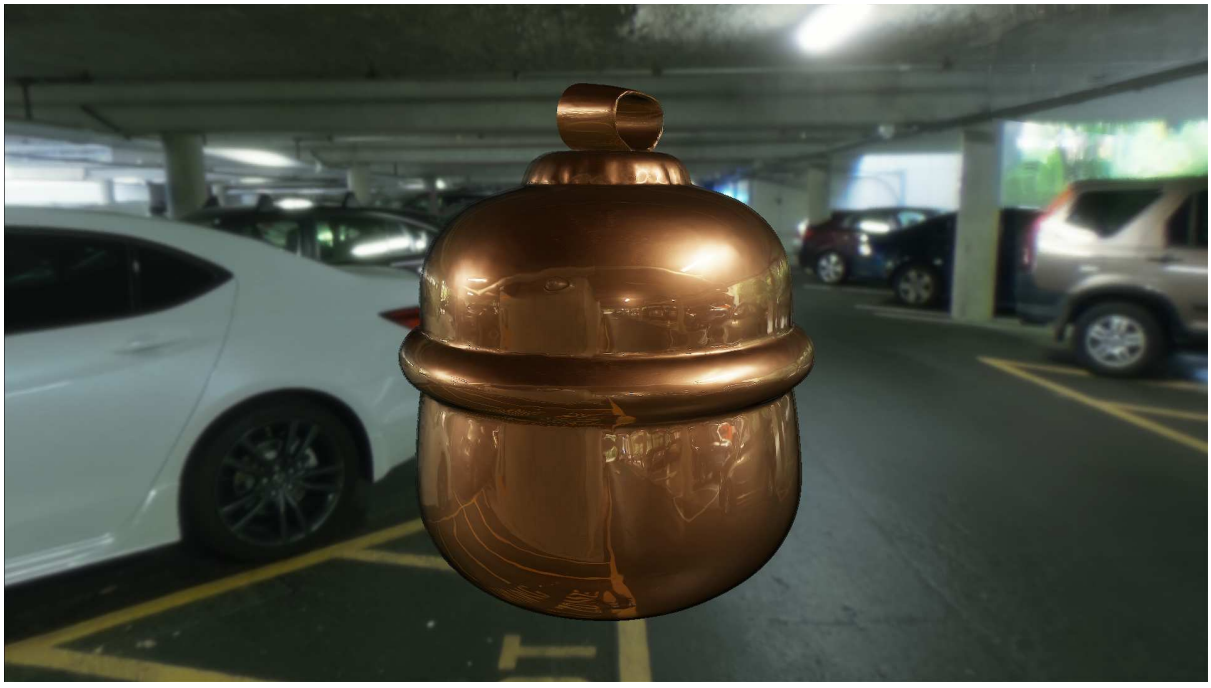


FIGURE 5.1. A Copper varnished pot rendered with the 2-flux transfer matrix.

An implementation of Randrianandrasana et al.'s 2 and 6 flux Transfer Matrix models (TM2 and TM6 respectively) is developed in a real time renderer. The models' sampling schemes are modified to work with pre-integrated lighting, an essential component of real-time rendering due to the intractability of many of the integrals present in the rendering equation. In addition, the directional albedo (FGD) LUT used in the original paper is replaced with a smaller and more accurate split-sum form developed by Belcour, Bati, and Barla. This implementation is validated for accuracy and physical correctness against

the path-traced ground-truth implementation provided in Randrianandrasana et al.'s paper. The implementation is then profiled across a range of optimisations and approximations, and a comprehensive analysis of its performance characteristics on contemporary hardware is provided.

My Transfer Matrix implementations were built using Microsoft's DirectX 12 Framework, based on the 'MiniEngine' sample provided by Microsoft under the MIT License. This provided a comprehensive foundation to build on, skipping much of the boilerplate and infrastructure development that would otherwise be required for experimenting with real time BSDF implementations. The 'MiniEngine' was chosen over other frameworks/engines as it is compact enough to be flexible, allowing one to re-design and iterate on the architecture as needed. It provides tools for loading models from a GLTF scene description, and a very simple forward-shaded render pipeline with temporal antialiasing, tonemapping and a library of pre-integrated (with respect to GGX) IBL probes. This engine was used to develop an implementation of both the 2-flux and 6-flux transfer matrix approaches described in Randrianandrasana et al.'s paper. The ImGUI library was used to provide a real-time parameter interface, allowing for rapid experimentation during the development iteration.

Other investigated lightweight options, such as BGFX, were unsuitable due to sacrifices made in serving cross-platform use (a feature not needed for this project), such as outdated feature support and a poorly maintained, opaque shader preprocessor system that could not be opted-out from. Similarly, full fledged engines such as Unity or Unreal Engine would introduce too much development overhead for integrating experimental shading models with the rest of the rendering pipeline, for features that were unnecessary for this research (for example, shadow-mapping, real-time global illumination, material graph compilation, etc).

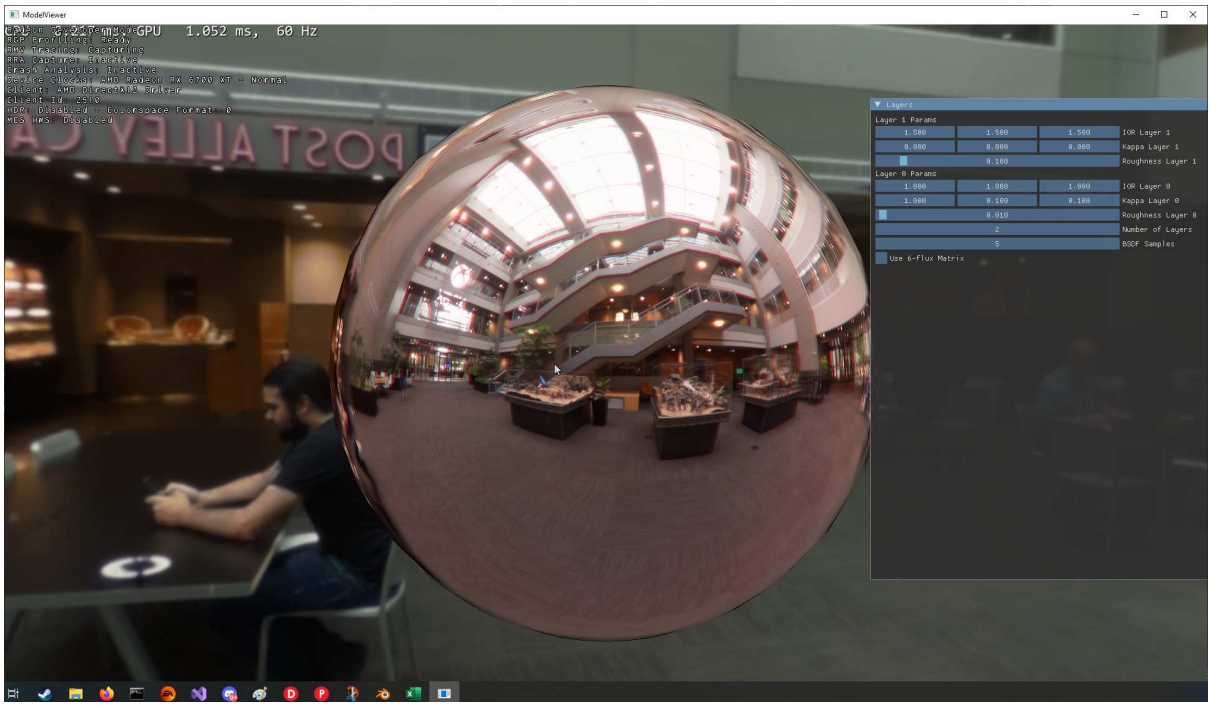


FIGURE 5.2. The user interface for the real-time impelementation of the transfer-matrix model, built on Microsofts 'MiniEngine' template.

I extend the MiniEngine to contain a basic UI for selecting the number of layers, and specifying the parameters per-layer. The currently rendered model can be swapped between the 2 and 6 flux variants in flight, with parameter state preserved across switches. All parameters update in realtime, allowing for fast experimentation and validation of results. An additional 'furnace' probe was added to the library of IBL probes for validation tests (discussed in the next section [5.2]). The MiniEngine's real time profiler is modified to display average, minimum and maximum timings for command-list invocations, using built-in DirectX hardware timer query functions. This reports timings over the last 64 frames. Rendering can be paused in order to take consistent snapshots of particular timings or scenes.

5.1.1 Fitting To Pre-integrated Lighting

In order to fit the Transfer Matrix models to real time rendering, it had to be made suitable for pre-integrated lighting. As discussed in Chapter 2, pre-integration is typically the only real-time appropriate solution for Image Based Lighting, which is in turn important for realistic and expressive environmental and ambient lighting. Common analytical approximations such as point sources do not depend on prior knowledge about the shading model or its parameterisation in the way that pre-integration does, and so

can be trivially adapted to any material model. Therefore for this research only pre-integrated lighting is considered and investigated, due to the potential additional complexity introduced by the dependence on the shading model. Preintegration often necessitates some further approximation in order to fit a general LUT, e.g. by assuming separability of component terms in the preintegrated function, such as in the Split-Sum approximation used by Karis, (2013). Computing the interaction between an IBL L and a BRDF f requires evaluating the following integral, for view direction ω_i, light direction ω_o and BRDF parameters Θ (e.g. roughness, IOR, etc):

$$L(\omega_i) = \int_\Omega f(\omega_o, \omega_i, \Theta) L(\omega_o) d\omega_o \tag{5.1}$$

(Lagarde & De Rousiers, 2014)

Note that this is very similar to the form given in Equation 2.2, albeit with the cosine weighting term missing as the light is assumed to be coming from all directions, and the incident and outgoing directions swapped - the latter being a matter of notation in Lagarde and De Rousiers's paper. For microfacet models (such as the GGX form of the lobes used in the Transfer Matrix Model), the result of preintegrating the GGX lobe with respect to the IBL (i.e, solving the light transport equation) is precomputed in the mip-chain of the IBL, parameterised by the lobe roughness α. This accounts for the widening of the specular lobe (and thus the reflection cone) under high roughness, corresponding to the characteristic blurring of reflections.

FIGURE 5.3. Preintegrated IBL probe. Note the progressive blurring at lower mip levels.

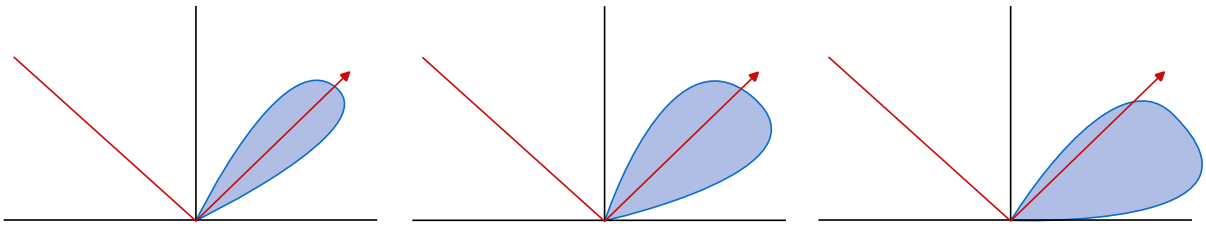
In order to retrieve the preintegrated lighting value at runtime, the surface's roughness is used as an index into the IBL mip-chain, retrieving the correctly pre-integrated value for the sampled location.

The Transfer Matrix model returns several GGX lobes, the contributions of which must be balanced appropriately. This complicates the preintegrated sampling scheme, as several samples must be taken and balanced accordingly. Under the path-traced implementation, Randrianandrasana et al. use Belcour's multiple importance sampling scheme for mixing and balancing each lobe's contribution. This produces

a probability distribution function (PDF) p that weights the BSDF's throughput, given below:

$$p = \frac{e_{\text{all}}}{\sum_{i=0}^{N} e_i p_i} \sum_{i=0}^{N} e_i p_i(\omega_i, \omega_o, \alpha_i) \tag{5.2}$$

where e_i is the energy of the ith lobe, e_all is the sum of the energy of all lobes, and p_i is the standard GGX PDF for the ith lobe. Randrianandrasana et al. first select a lobe at random, sample its visible normals for the outgoing ray direction, and then compute the BRDF throughput based on this particular lobe. Over several samples, this converges on the throughput for the entire BSDF. For pre-integrated lighting however, stochastic sampling can be avoided, as the BSDF has been precomputed with respect to the lighting source. As such, rather than sampling the vNDF, a perfect mirror reflection is taken off the incident geometry. This mirror ray is then shifted slightly using an approximation developed by Lagarde and De Rousiers in *Moving Frostbite to Physically Based Rendering* (2014)[1]. This accounts for the slight off-specular peak of GGX BRDFs at high roughness (Figure 5.4). This is slightly more accurate when calculating the distribution of visible normals (D) and masking-shadowing (G) terms of the lobe, as the half-vector is shifted proportional to the underlying microfacet distribution, rather than being fixed to the geometric normal.



(a) Low Roughness (b) Medium Roughness (c) High Roughness

FIGURE 5.4. Illustration of the GGX lobe 'shift' off from ideal specular reflection at higher roughnesses. The blue shaded region indicates the approximate distribution of reflected rays.

[1] `SpecularDominant`, section 4.9.3

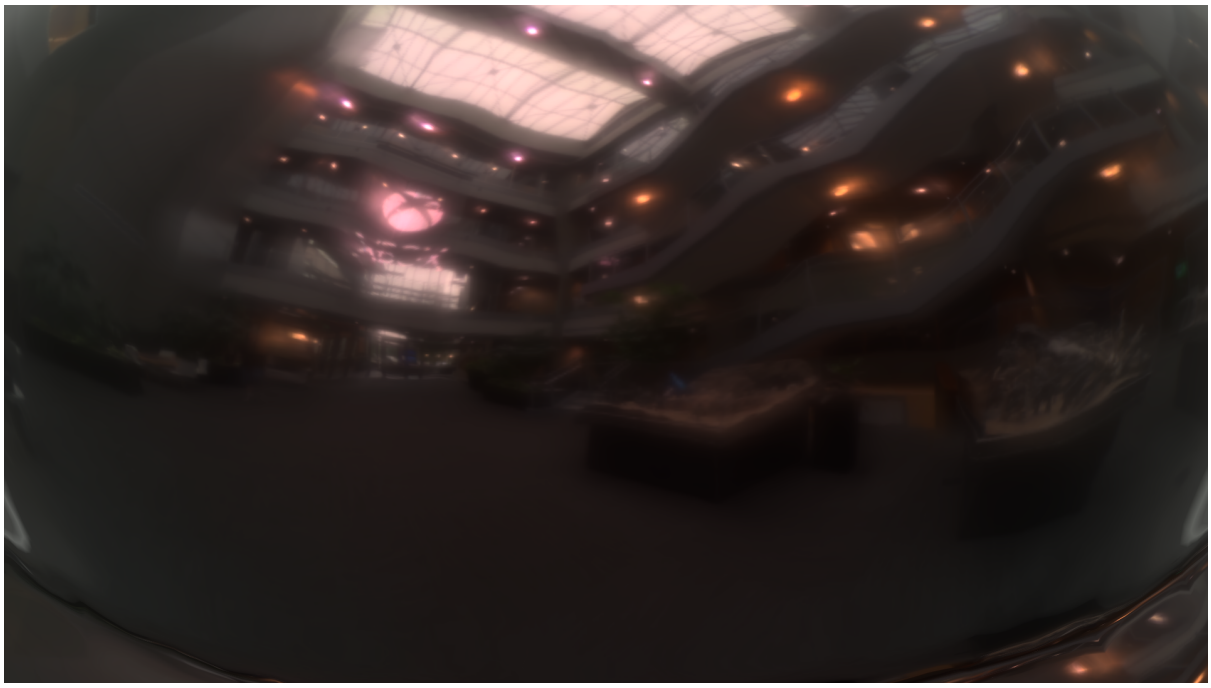
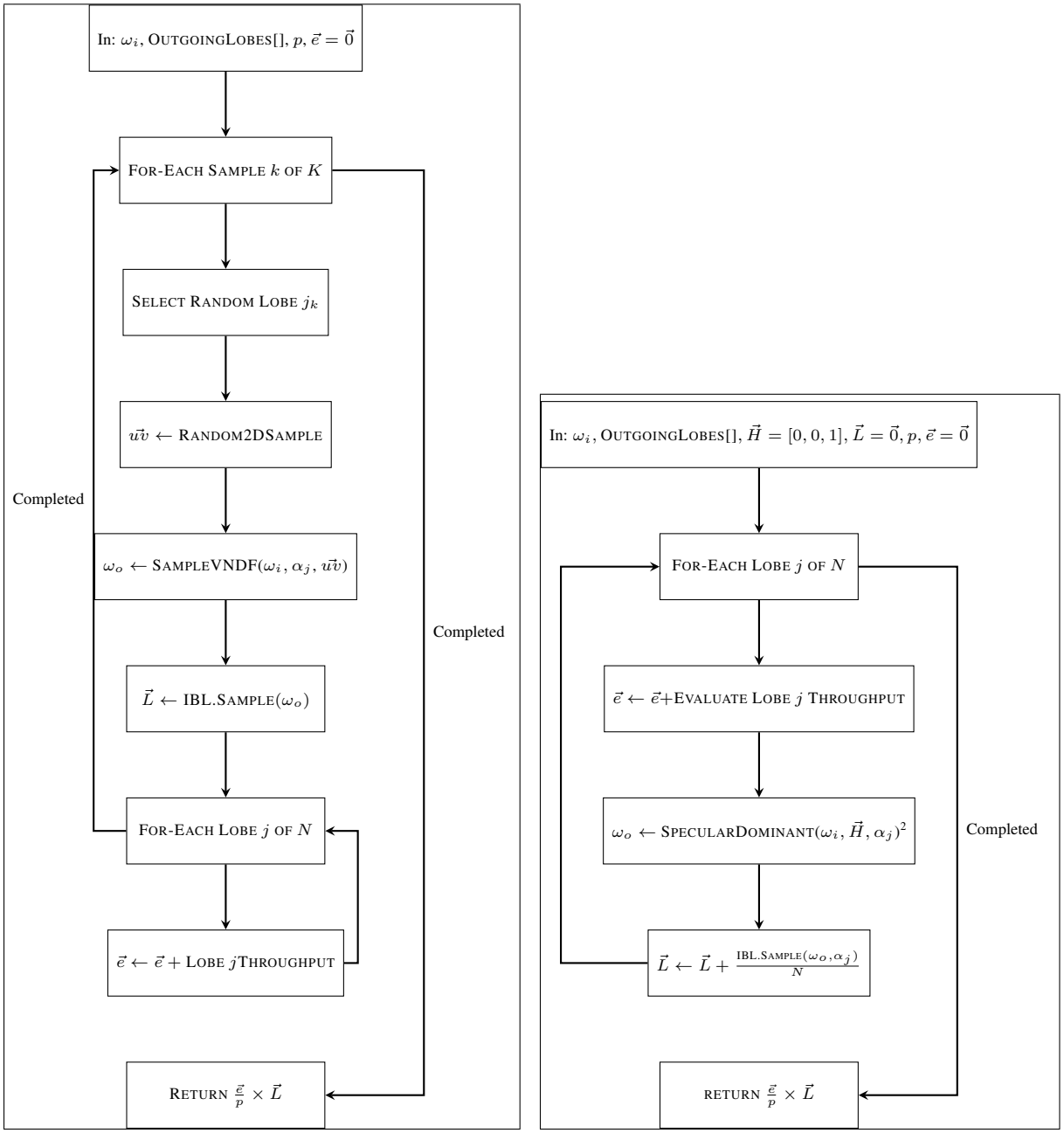


FIGURE 5.5. BRDF off-specular lobe shift. Note how the reflected highlights from the lights on the top-layer is shifted slightly from the bottom layer, owing to the top layer's higher roughness and IOR.

For each computed BRDF lobe, the light source (an environment map) is sampled at this outgoing ray. The outgoing lobe's roughness is used as an index into the precomputed lighting data, in order to correctly account for the 'blurring' of the specular reflection due to scattering of rays at high roughness. The sampled lighting is weighted by the number of lobes (to conserve energy when repeatedly sampling the same point) and then accumulated per lobe, and multiplied by the final BRDF throughput using the MIS scheme in Equation 5.2. This implementation also reuses the top-reflection correction term employed by Randrianandrasana et al. in their implementation, where the topmost lobe is evaluated with a full evaluation of Fresnel, in addition to the microfacet response.



(A) path-traced sampling scheme. Note that typically for convergence, K must be very large.

(B) preintegrated sampling scheme.

FIGURE 5.6. High level comparison of the sampling schemes used in the path traced reference and the preintegrated implementation.

5.1.2 A More Accurate Albedo LUT

Renderers typically also pre-integrate the directional albedo FGD term of the microfacet BRDF (Equation 2). This describes the ratio of reflected energy for an input view direction, roughness and index of refraction. Randrianandrasana et al. provide a 4D 64x64x64x64 LUT in their path-traced implementation of the Transfer Matrix models. For my implementation, this LUT is modified due to technical constraints with the DirectX API. Alongside this, I also provide an alternative formulation based on Belcour et al.'s Split-Sum form, which tests (discussed in the next section [5.2]) show to be both smaller and more accurate than the 4D LUT.

As DirectX 12 only supports resources with up to 3 dimensions, with a maximum width of 2048 pixels per dimension, Randrianandrasana et al.'s 4D LUT had to be resampled down to a 3D 64x64x2048 LUT, where the 4th dimension is binned into groups of two, averaged along the axis, and then stacked in strides along the 3rd dimension to create a 64x32 array of values. This is a naive and generally poor quality form of downsampling, and indeed the absolute error in this downsampling scheme is as high as 46%. However this downsampling scheme was found to preserve the shape of the LUT better than other, more advanced resampling schemes such as Chebyshev or FIR, which saw maximum error increase to 80%. This downsampled LUT is then indexed in my implementation at runtime by computing an offset from the 3rd dimension. This has an overall Absolute Error of 0.7%, with a maximum of 46.25%. The absolute error is plotted below in Figure 5.7.

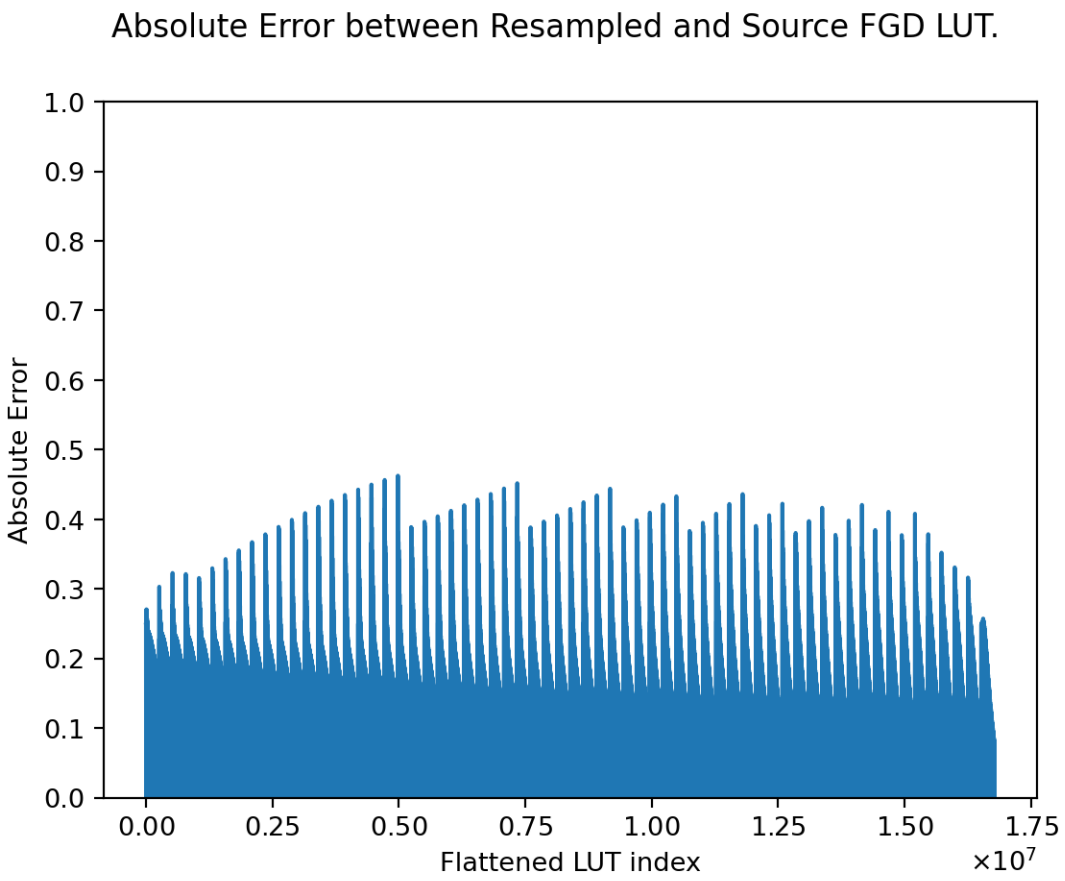


FIGURE 5.7

The peaks in the original signal tended to be along the 1st and 2nd axis bounds, as it captures the fresnel effect at grazing angles and low roughnesses. The box filter used to downsample captures these high frequencies poorly, hence the periodic pattern in the degree of error. Problems also appear when indexing into the LUT at runtime, where the desired coordinate lands on or near the 0th or 32th W slice in the original LUT, as the GPU texture sampling hardware will trilinearly interpolate between incorrect values: the last slice of the Z coordinate, rather than the neighbouring W coordinate.

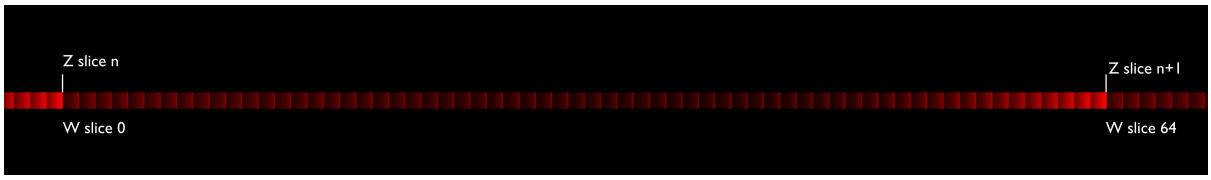


FIGURE 5.8. Diagram showing the issue with interpolation across slice boundaries when resampling the 4D LUT to 3D. Note the sharp transition between W slice 32 and the next slice (W slice 0, Z slice $n + 1$).

In practice, this resampled LUT introduced significant errors under certain parameter sets (as shown in the validation section 5.2), and so an alternative formulation was needed to resolve this problem. This comes in the form of Belcour et al.'s Fresnel formulation developed in *Bringing an Accurate Fresnel To Real-Time Rendering: A Preintegrable Decomposition*. Typically, real-time engines use a formulation of the FGD term known as the 'Split-Sum' LUT, developed by Karis (2013) for *Unreal Engine 4*. This uses the Schlick Fresnel approximation, in which the F_0 reflectance at normal incidence is factored out, and the Schlick basis functions and microfacet response (i.e the visibility and distribution of the surface microfacets) are precomputed into an $n \times n \times 2$ LUT, parameterised by ω_i and α. The resulting FGD term can be reconstructed by performing a dot product of the LUT sample with a vector composed of F_0 and its inverse:

$$FGD_{\omega_i,\alpha} \approx F_0 S_0 + (1 - F_0) S_1 \tag{5.3}$$

Where S_0 and S_1 are the precomputed LUT samples. This, however, is unsuitable for this model, as it fails to capture the absorption term in complex IOR, and Schlick is generally a low quality (albeit usually acceptable) approximation of Fresnel. Belcour et al. (2020) developed a reparameterisation of the Fresnel term that accounts for the complex IOR, and is amenable to this split-sum form. This uses the exact form of the Fresnel equations to compute a set of coefficients for 4 basis functions, so again this can be precomputed into an $n \times n \times 4$ LUT. The sampling scheme is very similar to the Karis 'Split-Sum' LUT:

$$FGD_{\omega_i,\alpha} \approx c_1(\eta + i\kappa)b_1 + c_2(\eta + i\kappa)b_2 + c_3(\eta + i\kappa) + c_4(\eta + i\kappa)b_4 \tag{5.4}$$

where c_i is the projection operator mapping Fresnel to coefficients for the ith basis function, and b_i is the ith basis function multiplied by the GD microfacet response (i.e the visibility and distribution of surface microfacets). The coefficients are computed at run-time, while b_i is precomputed and stored in the LUT. The channels of the LUT are given below:

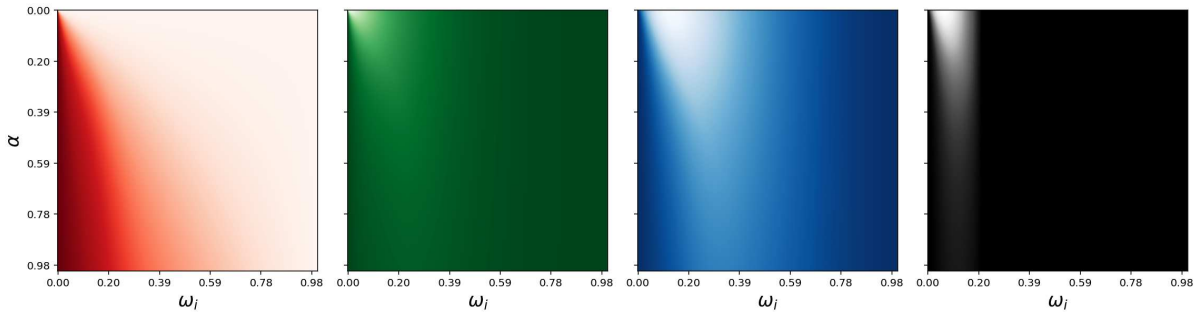


FIGURE 5.9. The RGBA channels of the Belcour FGD LUT

This new split-sum LUT (Figure 5.9) was generated with a method similar to the scheme given by Karis in the *'Real Shading in Unreal Engine 4'* paper (2013) [3], modified to use the height-correlated Smith G masking-shadowing function as described by Heitz (2014), rather than the Schlick approximation used in Karis's paper.

Unfortunately it is not easy to validate the correctness of this LUT, as Belcour et al.'s discussion of its formulation is very brief and abstract, and the supplementary materials provide no reference LUT or implementation for generating it. The course page does provide a screenshot of the LUT, however it is implied that this has been parameterised for Gulbrandsen or possibly Schlick Fresnel, and so can not be compared to my implementation (nor would the precision from a screenshot really be acceptable in any event). I instead verify the overall algorithm used to generate the Split-Sum LUT by using it to reproduce Karis's original Split-Sum LUT (as this differs only in the coefficients and basis functions used). I then benchmark the generated Belcour et al. Split-Sum LUT by making use of it in the model validation tests in the next section (5.2). This Split-Sum LUT is included in my implementation as an alternative to the resampled full-form FGD LUT.

Other modifications were made for performance reasons, which will be discussed in the following chapters.

5.2 Validation

As previously discussed, one of the main correctness criteria is the principle of energy conservation - the model should not emit or absorb more light energy than was incident. This constraint is particularly important for path tracers, as this property guarantees convergence of the result. However, for realtime renderers and rasterisers, which usually don't have a technical reason for preserving energy, this property is instead important for maintaining a consistent visual look & feel under any lighting scenario, an important artistic property of 'Physically Based Rendering'. As such, the strict conservation requirement can be relaxed somewhat if it allows for a faster, more approximate model to be used. Validation is done using the 'furnace test', discussed in Chapter 4. The relaxation of the strict conservation requirement means that the furnace test is also used to measure the 'degree' of error, by examining how much energy on average is over-absorbed or over-emitted.

[3] See "Image Based Lighting" and "Environment BRDF" sections

For these experiments, a furnace of a uniform energy of 0.42 is used. This is an arbitrary selection, save for being a value approximately mid-way between 'no energy' and 'total energy', so that both energy loss and generation can be seen without clipping in a low dynamic range image. Randrianandrasana et al. use a similar value for the furnace tests done in their paper. The rasterised implementation is compared against the path-traced reference implementation for Mitsuba, provided by Randrianandrasana et al.. The path-traced reference is rendered with 512 samples, to a linear output. The rasterised render is generated by saving the pre-tonemapping linear render target from the MiniEngine implementation. This isolates the effect of post processing and tonemapping from both implementations, so that like-for-like energy can be compared. A difference plot is generated by taking the absolute difference of the average of each pixels RGB channel. This shows overall similarity between the images, however will miss errors across individual Red, Green and Blue channels. These differences can be seen easily by inspection, however.

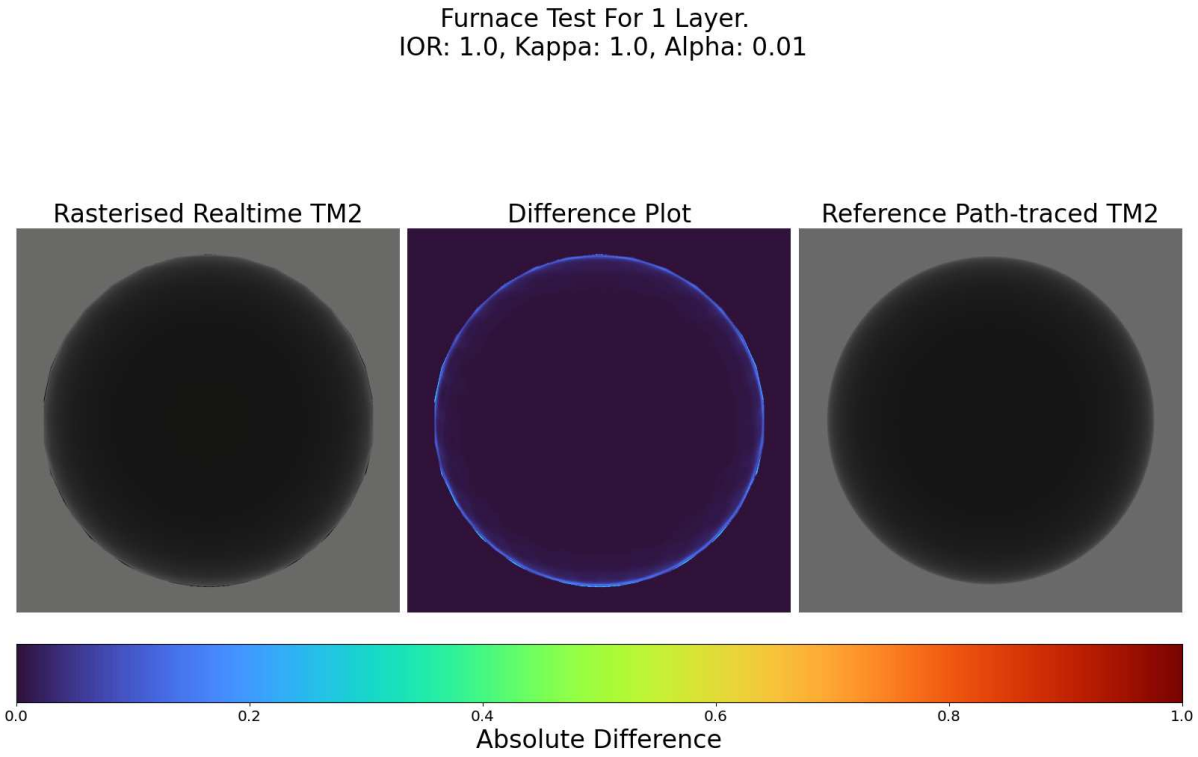


FIGURE 5.10

Figure 5.10 shows a comparison of my rasterised implementation of the 2-flux matrix with Randrianasandra et al's path traced implementation for the Mitsuba renderer. The error is concentrated around the edges of the sphere, where due to the limited geometry resolution in the rasterised implementation

the incident angle is 0 or negative, and the output is clamped to 0. The Mitsuba implementation is rendered onto a parametric sphere, and so does not suffer from this problem. This could be corrected fairly simply, but is absent from the rasterised implementation due to time constraints. What this does show, however, is that due to the complex IOR for conducting materials, this model will steal energy due to absorption that would usually be re-emitted as infrared (or some other, non-visible electromagnetic radiation), which is evidently not modelled in most renderers. This is to say that unlike a traditional furnace test, it is expected that this model show some darkening in the furnace test, depending on the absorption term of the substrate. However, the *amount* of absorption should not change with each additional dielectric layer, as the vast majority of incoming light should be either transmitted or reflected.

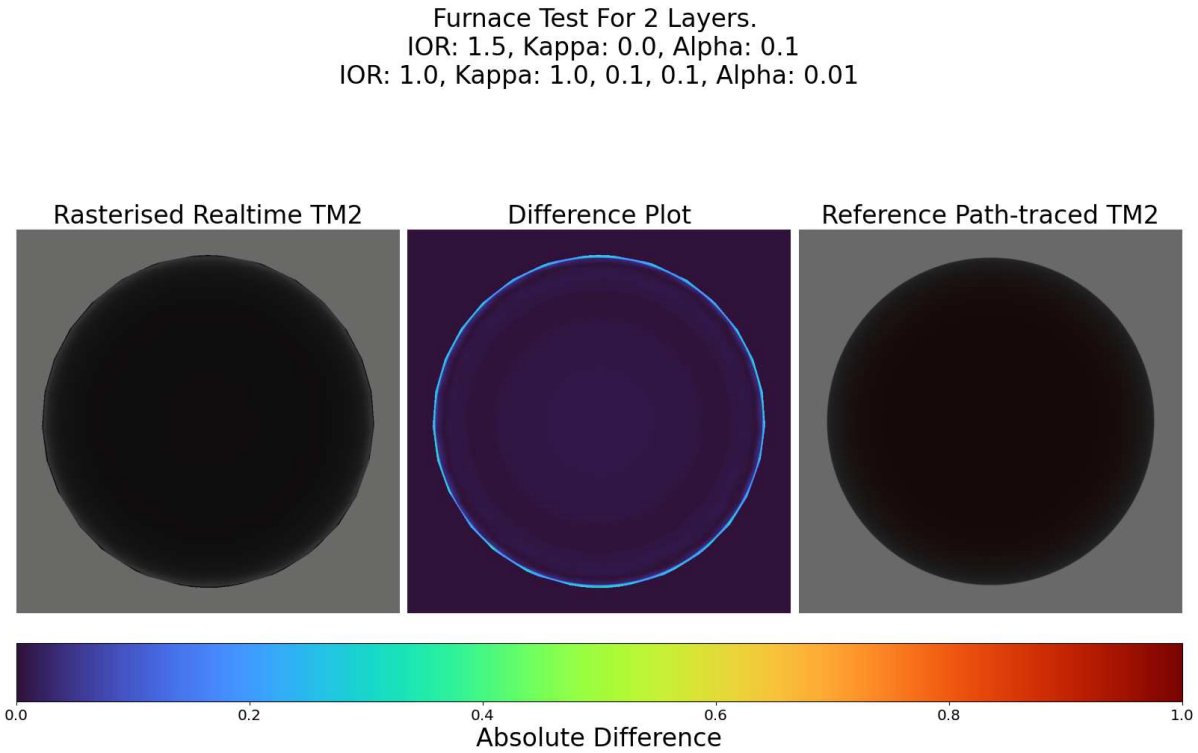


FIGURE 5.11

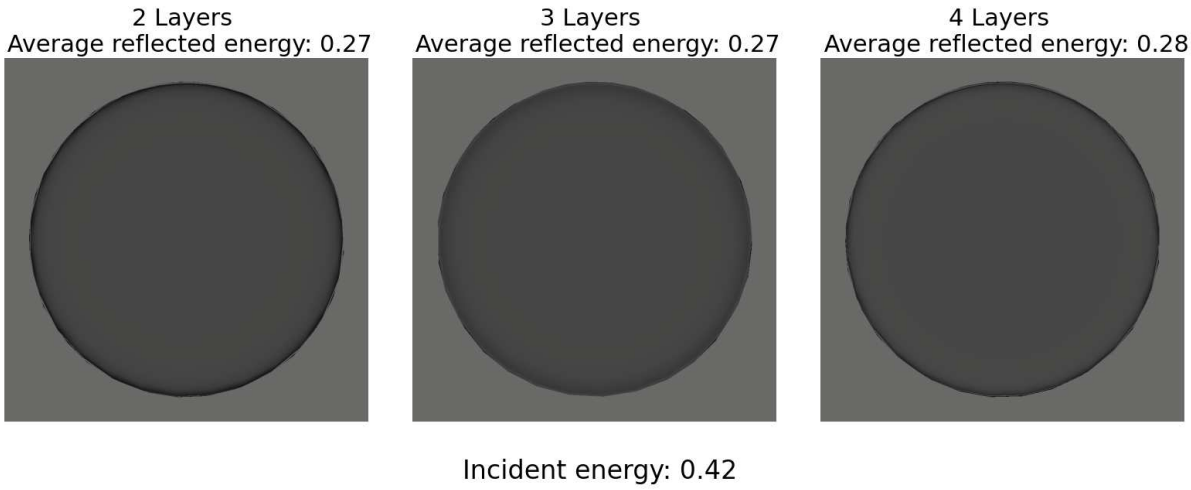


FIGURE 5.12. Energy is conserved as additional dielectric layers are added.

My model is currently *close* to Randrianandrasana et al.'s path-traced reference, as shown in Figure 5.11. The degree of error is on average quite low, and never more than 10%, except at the extreme grazing angles, as previously mentioned. However the model misses some edge tinting present in the path traced implementation, compressing the down to a grey tint much faster than the reference model does. In addition, the edges suffer from over-darkening. The model conserves its energy over multiple layers (Figure 5.12).

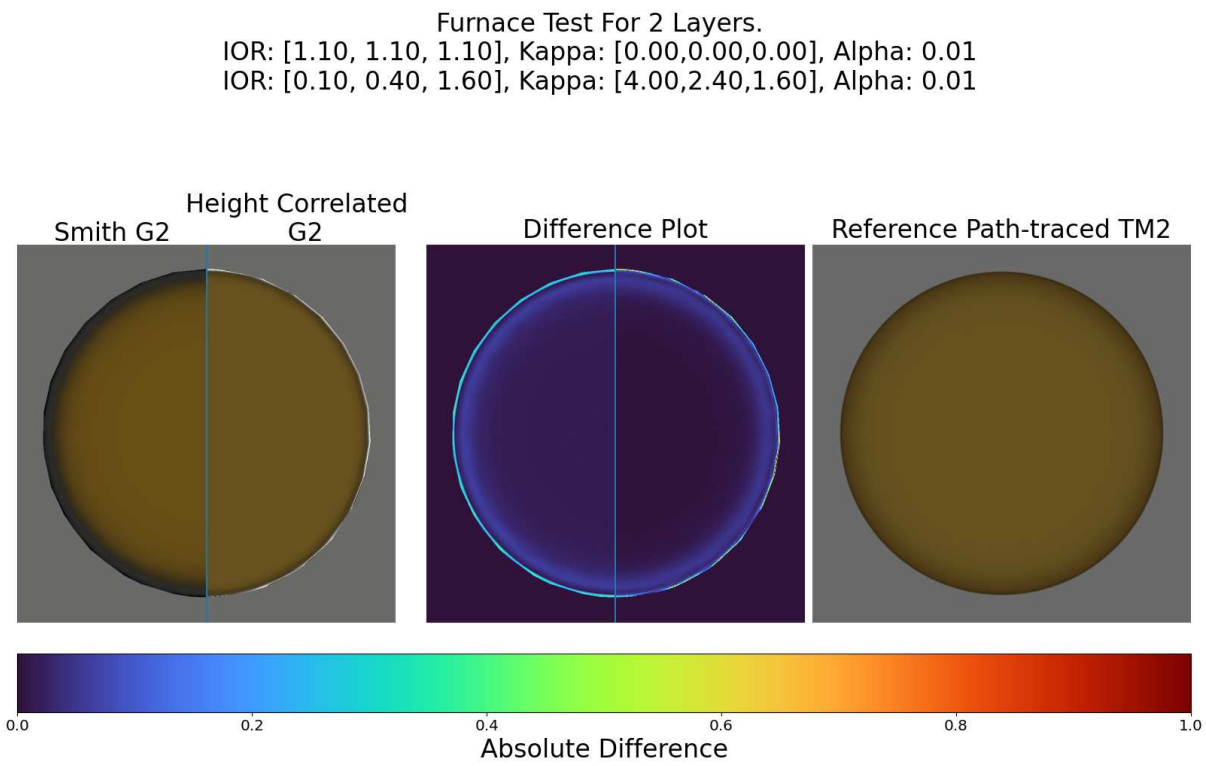


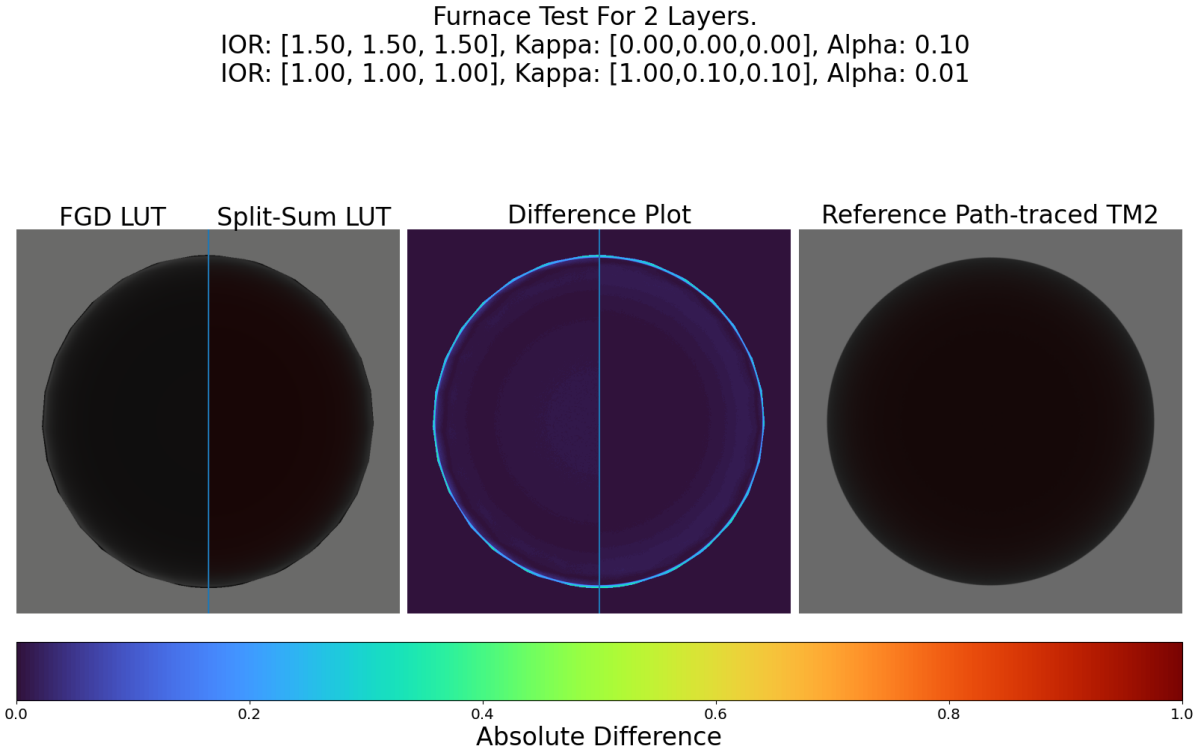
FIGURE 5.13. Correcting for the over-darkening at grazing angles with a height correlated masking-shadowing function.

The overdarkening in the implementation can be compensated for by using a height correlated masking-shadowing function, such as the one described by Pharr and Humphreys in the 'PBRT' book (2023). This is a more conservative variant of the masking shadowing function over the smith implementation, which simply multiplies the masking $G1$ term from ingoing and outgoing perspectives together. Instead, the height-correlated variant accounts for the assumption that more elevated microfacets are more likely to be visible from both the incoming and outgoing directions, and so will neither be masked nor shadowed. While this does greatly improve the overdarkening at grazing angles, it introduces some overbrightening at roughnesses between 0.4 and 0.6 along the extreme edges of the object, where the incident angle is very close to 0 (Figure 5.13). This is again due to a mismatch between geometric normal and the actual position of the sample point caused by low geometry resolution. It largely be resolved by tweaking the epsilon values used for determining the termination of the geometry. In general, the GGX microfacet model used in this research (and most production renderers) only accounts for a single reflection event off of a microfacet[4], and so some energy is typically lost - particularly at grazing angles and high

[4]typically referred to as 'single-scattering', however I reserve the use of 'scattering' for referring to bounces *within* participating media, rather than off the surface of an interface.

roughnesses - due to rays that would realistically be reflected as back-scatter being instead masked or shadowed by the $G2$ term. This is a tradeoff made by many renderers as a multi-scattering formulation is expensive to evaluate and difficult to approximate.

Some limited validation of the Belcour Split-Sum LUT is done by spot-checking various layer combinations against the path-traced reference.



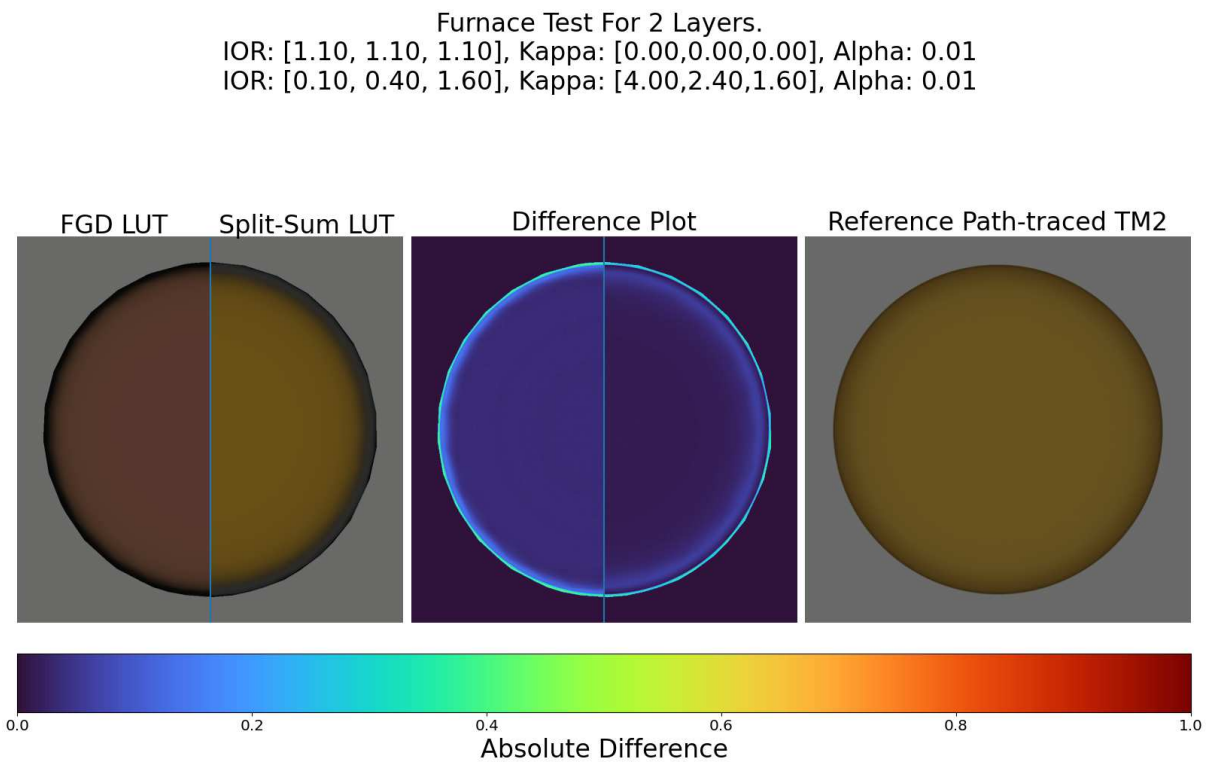


FIGURE 5.14. Comparison of accuracy with FGD and split-sum LUTs.

The split-sum LUT is much more accurate overall, and more importantly, does not have the same banding issues visible with the FGD LUT, allowing for a continuous transition over different IORs, roughness and incident angle. It produces a much richer reproduction of specular tint in conducting materials when compared to the 4D LUT. This is due in part to the significantly higher resolution the split-sum LUT can have (for a smaller size), as well as the analytic form used for Fresnel. There are general accuracy problems with several dielectric layers with the 4D FGD LUT, as the reflected colour becomes overly dark and incorrectly tinted for certain layer stacks. It is not clear why this is, but is likely due to the inaccuracies introduced in the course of downsampling and reshaping the LUT due to the aforementioned technical constraints.

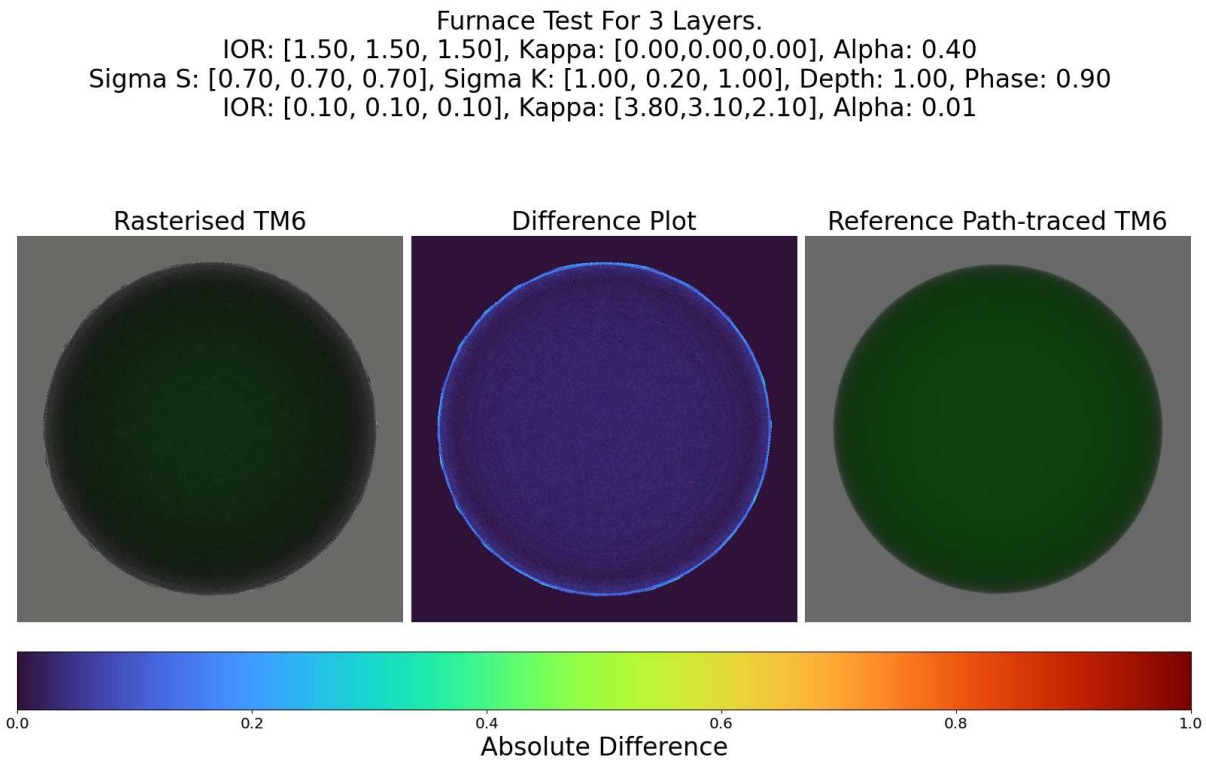


FIGURE 5.15. Energy Loss with a Scattering Medium in the 6-flux Model.

Being an extension of the 2-flux model, the real-time implementation of the 6-flux model is similarly accurate with respect the reference implementation. Problems with the real-time implementation arise when introducing a layer stack with a scattering medium, as there are significant problems with energy loss. This is compensated in part by reweighting the IBL samples for the outgoing lobes - the lobe contribution of participating media is ignored when weighting the samples. Further, the scattering medium introduces significant amounts of noise, likely due to numerical instabilities in the model.

5.3 Performance Profiling

Performance and correctness was evaluated with Microsoft's PIX Debugger and AMD's GPUOpen profiling tools. The former provided the ability to interactively view bound render resources and step through the shader in a debugger, which was valuable for tracking down correctness problems (such as with numerical robustness) and other bugs. AMD's tools were chosen as they are the vendor for the GPU used in the machine on which this project was developed. These tools provided deeper instrumentation

of the shader ISA, such as per-instruction latency timings and register pressure, alongside overall occupancy, throughput and utilisation analyses. This helped with identifying the performance bottlenecks during the optimisation passes on the implementation.

The profiled machine is an AMD 6700XT (RDNA2 architecture), with a Ryzen 5 CPU.

We begin with a review of baseline performance for the 2 and 6-flux models, and then proceed through a number of optimisations and review their performance and accuracy tradeoffs.

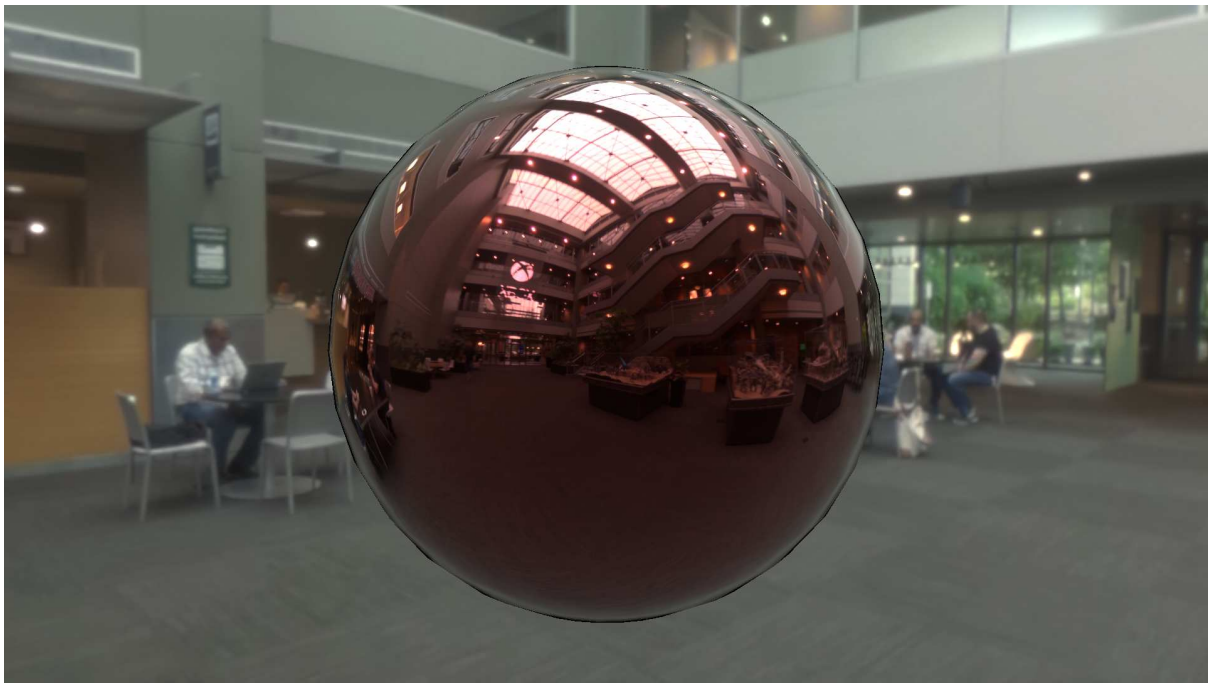


FIGURE 5.16. Reference scene used to profile. Layer 0 params: IOR: 1. Kappa: 1.0, 0.1, 0.1. Roughness: 0.01. Top Layer params: IOR 1.5, Kappa: 0, Roughness 0.1.

Performance was profiled on a full-screen 1080p render of the BSDF with 2 layers, such that geometry (and thus the pixel shader running the model) occupies the entire screen. The 2 layer parameter set is indicative of a 'typical' use case for layered materials, e.g. clearcoating, and matches one of the samples used in Randrianandrasana et al.'s paper. Profiling is done by taking a rolling average, minimum and maximum across 64 frames, timing only the draw call for the transfer matrix pixel shader. This avoids including overhead from features such as temporal anti-aliasing, auto-exposure, and other post process effects, isolating *only* the time taken to execute the shading model itself. The below profile is running

a release build of my 'MiniEngine' fork, with all optimisations enabled, and using full-precision types. This represents the 'baseline' timings, with no special optimisations applied.

	Average(ms)	Minimum (ms)	Maximum (ms)
2-flux Model	1.433	1.401	1.486
6-flux model	23.275	23.167	23.536

For completeness, I also show that shader time is independent of the layer parameters. Maximum variance is about 20 microseconds. Intuitively, this is because there is little to no branching dependent on the parameters, and so the same code-path is always executed across the parameterisation.

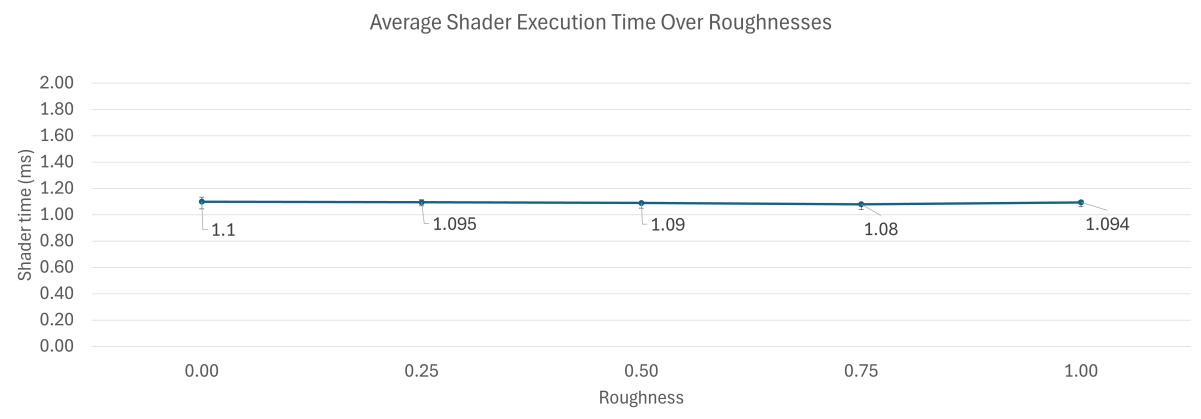


FIGURE 5.17. Average shader invocation time across varying roughness (note: not spatially varying) on a single layer variant of the 2-flux reference scene in Figure 5.

When profiling roughness, I use only a single layer and vary the lobe roughness over it. This represents the cost of varying a single lobe's roughness, without the influence of multiple layers and the scattering interactions between them. The maximum variance between minimum and maximum invocation times was 50 microseconds.

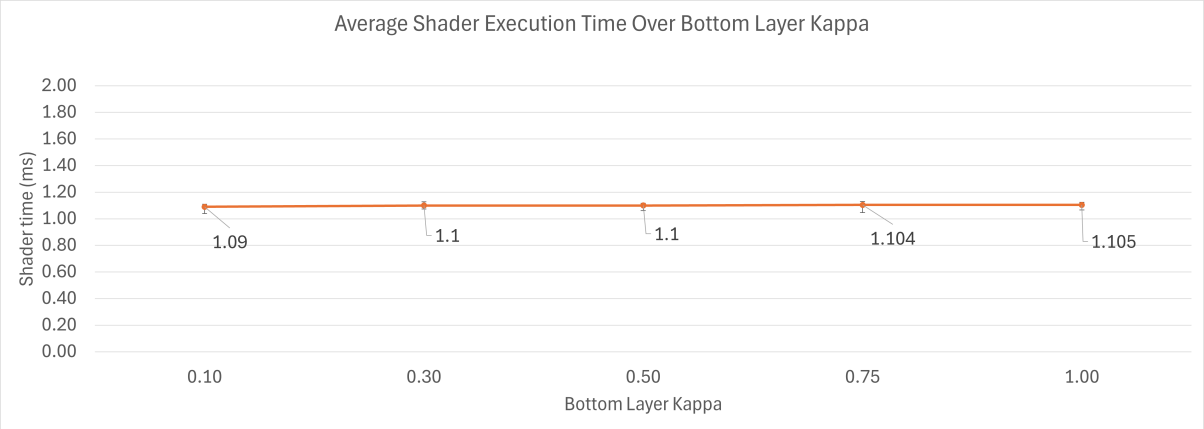
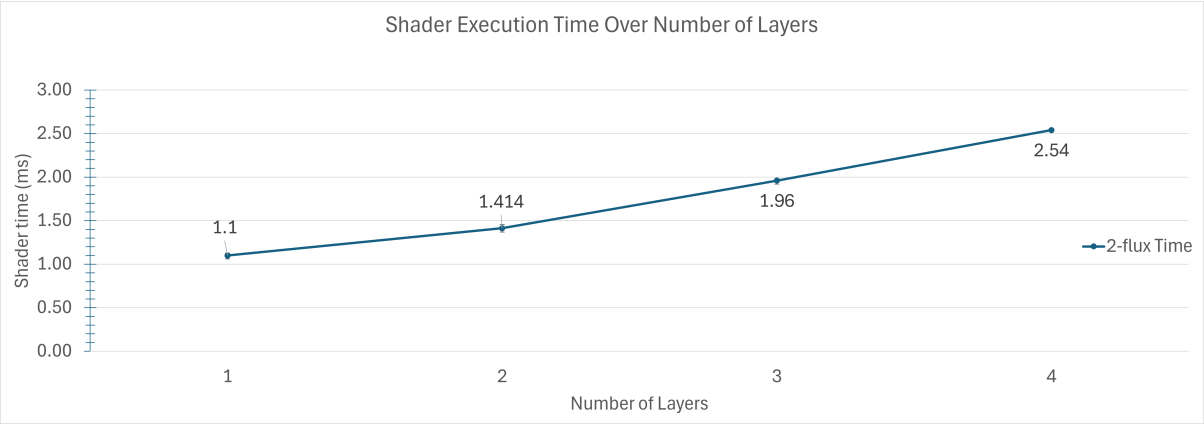


FIGURE 5.18. Shader invocation time across varying κ (note: not spatially varying) on the 2-flux reference scene in Figure 5. Only the bottom layer κ is varied as no layer evaluation is done after the first conductor in the stack.



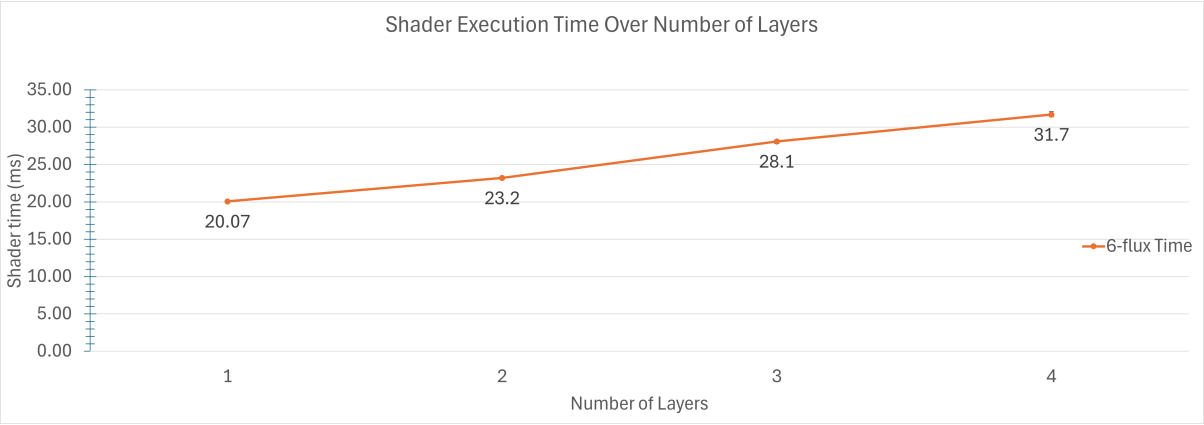


FIGURE 5.19. Shader invocation time across the number of layers in the stack, for both 6 and 2-flux models.

The runtime is instead dependent on the number of layers in the layer stack. This has a roughly linear growth, growing $\sim 1.3\times$ for each layer with the 2-flux model, and $\sim 1.2\times$ for the 6-flux model. This is consistent with the algorithm, which is again dominated by the matrix multiplication cost per-layer, and so extra layers contribute to a linear growth in matrix multiplies. Architecturally, the maximum number of layers is fixed at compile-time (5, in this case). An implementation using a dynamic layer buffer may see more pessimistic growth rates as the compiler & hardware will have a poorer understanding of the memory layout and may not be able to cache the layer data as effectively, nor optimise evaluation.

The baseline performance for the 2-flux model is within a close margin for an ideal production ready model, at $\sim$ 1.5ms without evaluating any pre-lighting shader code (e.g. for dynamic parameters), which makes it a good candidate for applying further simplifications and optimisations. Given the similarities between the 2 and 6 flux models, the optimisations discussed in this chapter can be applied to the 6 flux model as well. However, given the sheer size of the 6-flux model, it is likely that it is limited largely by register occupancy, and so optimisations are unlikely to improve its efficiency significantly. Overall, these baseline results demonstrate a significant improvement over a path-traced, offline implementation, which took approximately 10 seconds to render at 512 samples on the reference scene.

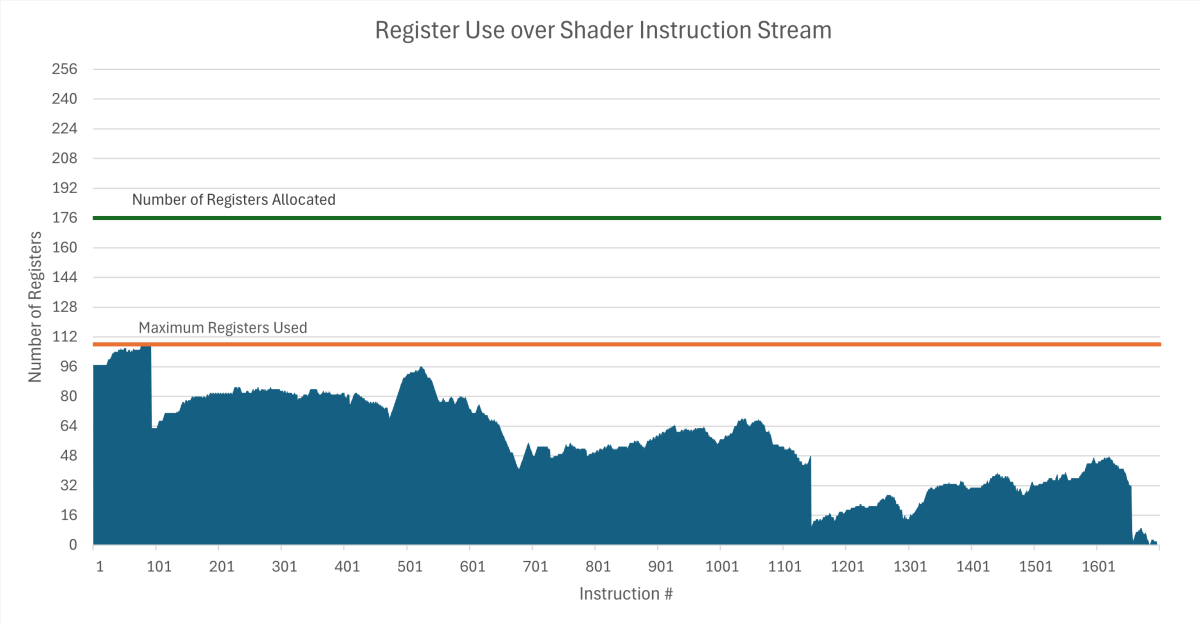


FIGURE 5.20. Register use for the 2-flux model.

The register use of the shader is very high, which severely limits the maximum occupancy it can have, and thus its parallelism. The densest parts of the model is the translation from parameters to component

factors, in turn used for setting up and evaluating the transfer matrices. 4 `henyey-greenstein` lobes per layer are required for the TM2 model, and up to 6 per layer are required for the TM6 model. At 36 bytes per lobe, this is the largest part of the model evaluation, consuming at least 36 registers per layer. On the profiled hardware, there is a maximum of 256 registers available to be allocated to a wavefront. Depending on the proportion of registers used by a wavefront, this changes the number of wavefronts that can be executed in parallel - more registers consumed means fewer wavefronts are able to be dispatched at once. For this particular shader there is a high degree of waste in the register allocations, shown by the large difference between the number of allocated registers versus the maximum number actually utilised (Figure 5.20). This is because space for all layers - up to the maximum of 5 - is allocated within the shader statically, and so if less than 5 layers are used this is wasted space as the compiler is unable to eliminate the extra allocations. This is partly an artificial limitation imposed to make testing and experimentation easier, as it would not be unreasonable in a production context to fix the number of layers for a given material at compile time, as the Substrate (Hillaire & De Rousiers, 2023) framework does.

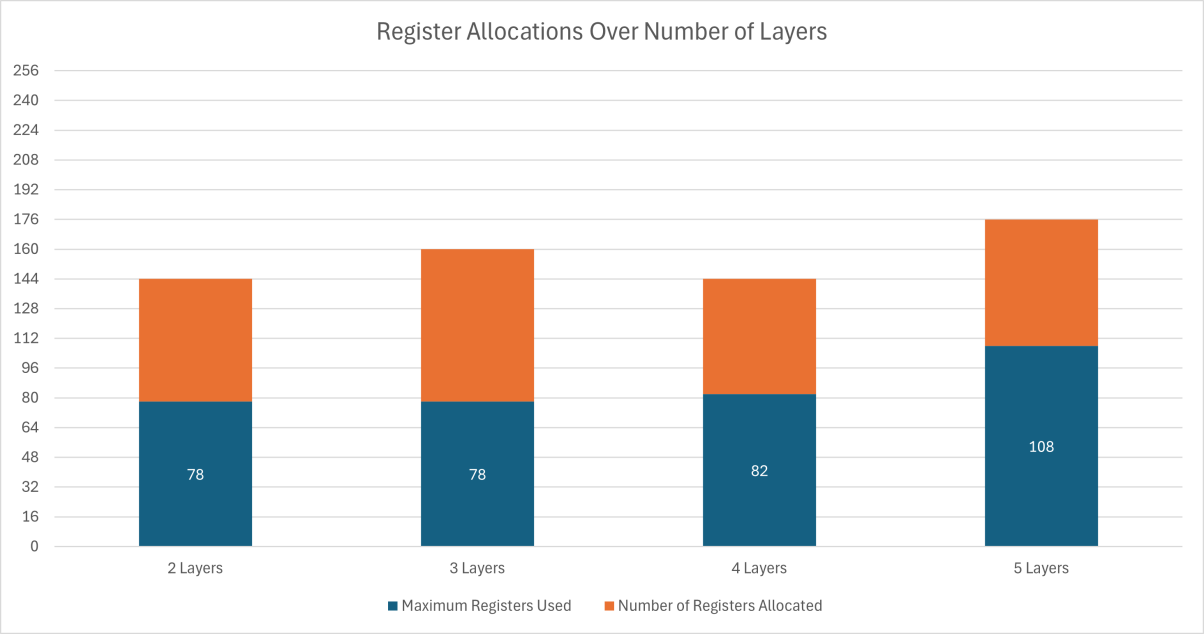


FIGURE 5.21. Register allocations when varying the maximum layer constant.

If the maximum layer constant is reduced (requiring a shader recompile each time, so can not be done at run-time), the register use does drop significantly. This is enough to schedule a single additional wavefront per SIMD in parallel: the profiled GPU has 1024 VGPR registers available, with the peak of

register allocation at 176 for 5 layers, and the minimum at 144 for 2 layers. The shader is scheduled in wave64 mode, and so the number of available registers is effectively halved. Therefore the theoretical occupancy for the 5 layer maximum at 176 registers allocated was 2 wavefronts per SIMD, while the theoretical occupancy for the 2 layer maximum at 144 registers is 3 wavefronts per SIMD (Guthmann, 2023), out of a total of 16 wavefront slots per SIMD. There is some variation in register allocation, rather than a linear downward trend with respect ot the number of layers, as the compiler is using the newly available space to trade off program size for other performance gains realised through loop unrolling and better cache coherency.

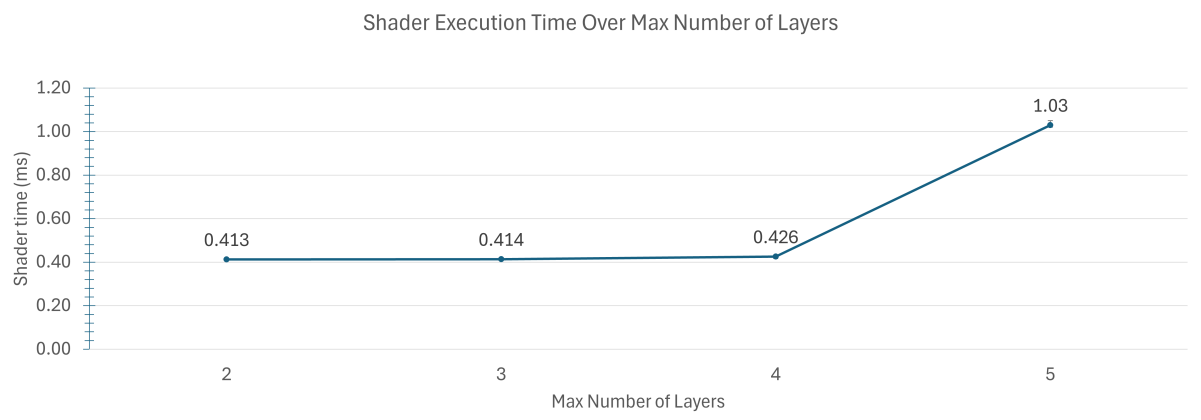


FIGURE 5.22. Execution time for a single layer in the TM2 model when varying the maximum layer constant.

Notably, all use of scratch space is eliminated when transitioning from 5 to 4 maximum layers. Scratch space is a small, thread-private cache used for storing temporary data, or data spilled from registers due to an inability to fulfill register demand. Spills to scratch are particularly bad for performance as, despite being thread-private, the address space itself is situated in global memory, which has a significant distance to the chip when compared to the caches and registers, and thus a very high latency (Curtis & Fanfarillo, 2022). This elimination of scratch use is far more significant for the performance gains seen in limiting the maximum layers than the minor occupancy improvement, as it allows VALU throughput to increase significantly as the GPU is no longer waiting on slow dependent loads and stores for spilled data. This is reflected in the steep decrease in execution time from 5 to 4 layers, which immediately flattens out, seeing essentially no gains as maximum layers are further reduced (Figure 5.22). Note that a minimum of 2 layers is required, as the 1st layer is always the 'air' medium. Therefore, if the 1st layer is set-in shader at compile time, rather than passed in via the layer parameters, approximately 100 microseconds can be saved by the additional constant folding/propagation the compiler is able to do.

This drops runtime in the 2-flux model from 1.4ms to 1.3ms. This does not however change the register allocations, as space for the air medium henyey-greenstein lobe representation still needs to be made, and this cannot be precomputed as it is dependent on the parameters of the layer below it.

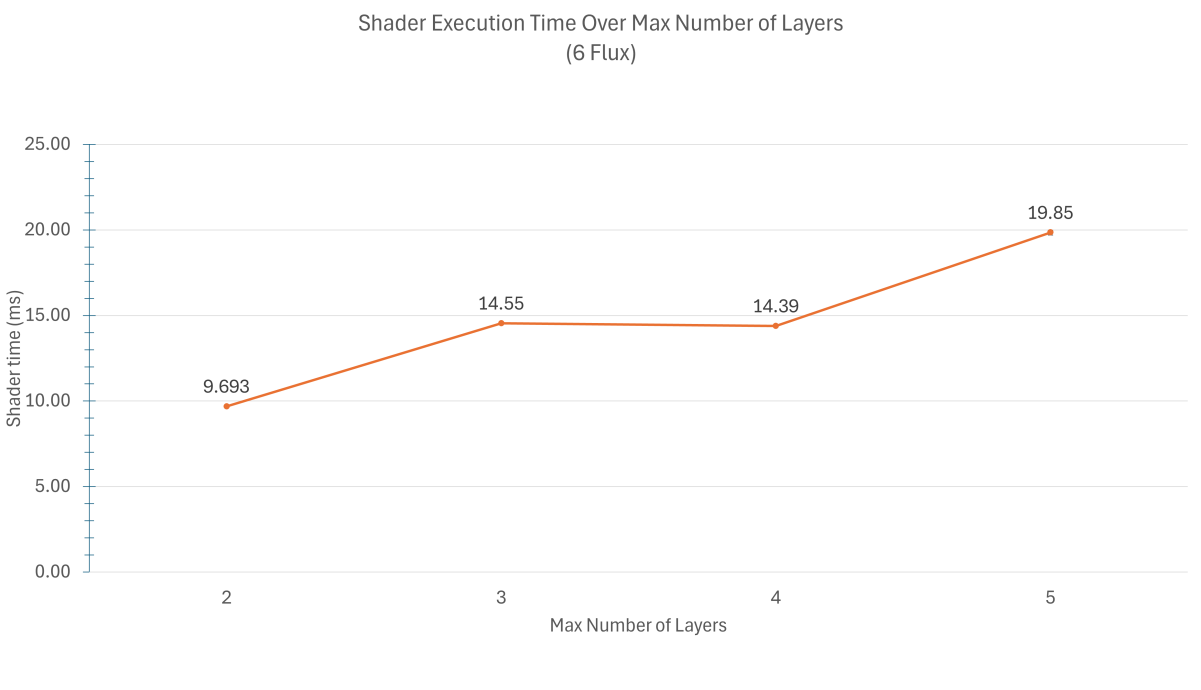


FIGURE 5.23. Execution time for a single layer in the TM6 model when varying the maximum layer constant.

A similar relationship between runtime and the maximum layer constant can be seen with the 6-flux model. The 6-flux model is so large that all 256 registers are saturated across any maximum layer bound, and not only does data always spill out to scratch space, but the instruction stream itself is too large to fit in the instruction cache[5]. This adds additional latency due to loading the next instruction in the stream from global memory, and hampers the ability of pipelining techniques such as branch prediction and instruction reordering to work effectively. The improvement from maximum 5 layers to maximum 4 layers is due to the smaller program generated (approximately 1000 instructions smaller), and so the average instruction cache hit rate improves from 90% to 98%. The other improvement from 3 to 2 maximum layers is where the entire program is now able to fit into the instruction cache, and its length is again approximately 1000 instructions smaller. There is a minor improvement in hit-rate

[5]note that although the 6-flux model uses the same lobe format as the 2-flux model, it generates three times as many lobes per layer to simulate multi-scattering and other effects.

against the L2 data cache from 39% to 41%. This overall means that both variants of the model, 2-flux and 6-flux, are bottlenecked by their size, and thus associated memory latency.

5.3.1 Optimisations

One of the simplest optimisations tested is the use of half precision types for much of the arithmetic. The structures used in the core of the computation are quite large, even when exploiting sparsity. For example, at full precision the 6-flux matrix occupies 80 bytes. Two matrices are needed per layer for tracking energy and asymmetry. This can quickly saturate the vector registers available on consumer hardware, as well as the data and instruction cache. While the implementation instructs the compiler not to unroll the matrix multiplication loop, which limits the number of matrices in-flight to 4, this is still more than enough to saturate the available registers. Half precision types should in theory immediately halve the size of these structures, while having the additional benefit of associated arithmetic being processed at double-rate on the profiled hardware (AMD, 2023).

This appeared to pessimise performance on both the 2 and 6 flux models, with the 2-flux model taking 2.686ms on the reference scene, and the 6-flux model taking 24.000ms. Additionally, it introduces some significant banding artefacts which are generally unacceptable for surface representation:

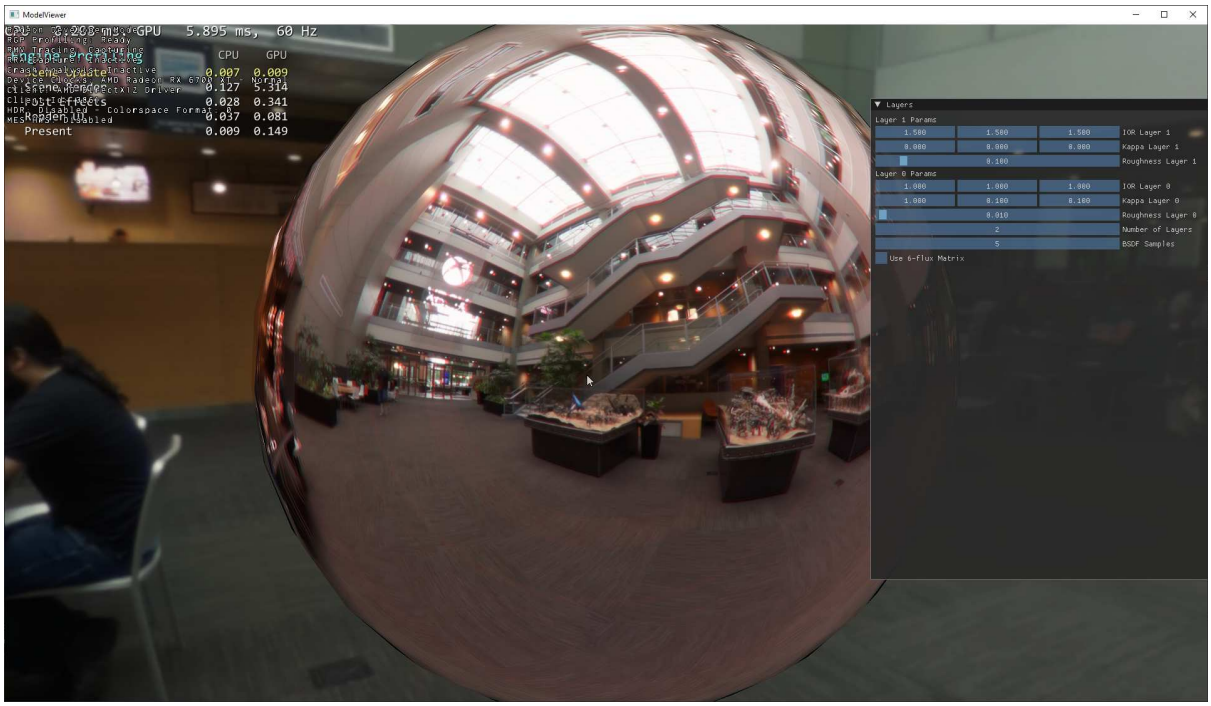


FIGURE 5.24. Banding artefacts visible when using half precision types for layer evaluation.

Vendor tools show that the register use drops by about 8 registers, which in turn allows for the peak register allocation to drop by 16, as registers are allocated in blocks of 16. Scalar register use decreases by 8, which implies a reduction in scalarisation that may be contributing to the slowdown. There were 400 more instructions in the FP16 implementation assembly, an increase of 20%. The disassembly does show some conversion instructions within hot parts of the code, which are the usual culprits of slowdown when using half precision types, as round trip F32 to F16 to F32 conversions within a block of computation is obviously spurious, however only 88 of the instruction stream are conversion operations, compared to 4 in the FP32 implementation. This means the extra instruction overhead is not primarily due to these conversions.

Other low-hanging fruit optimisations were applied by dropping in approximations for common functions that exist in the literature, such as the GGX D NDF term approximation developed by Karis for Unreal Engine 4, and a similar approximation for the Smith G geometry term developed by Earl (2017) for Titanfall 2.

	Average (ms)	Min (ms)	Max (ms)
Baseline (Smith D/G GGX)	1.433	1.401	1.486
Karis D GGX	1.458	1.397	1.516
Earl G GGX	1.479	1.403	1.531
Both	1.454	1.414	1.504

This had little to no impact on performance, possibly even pessimising it slightly (20 microsecond difference between baseline average and both approximations applied), again implying a memory bottleneck.

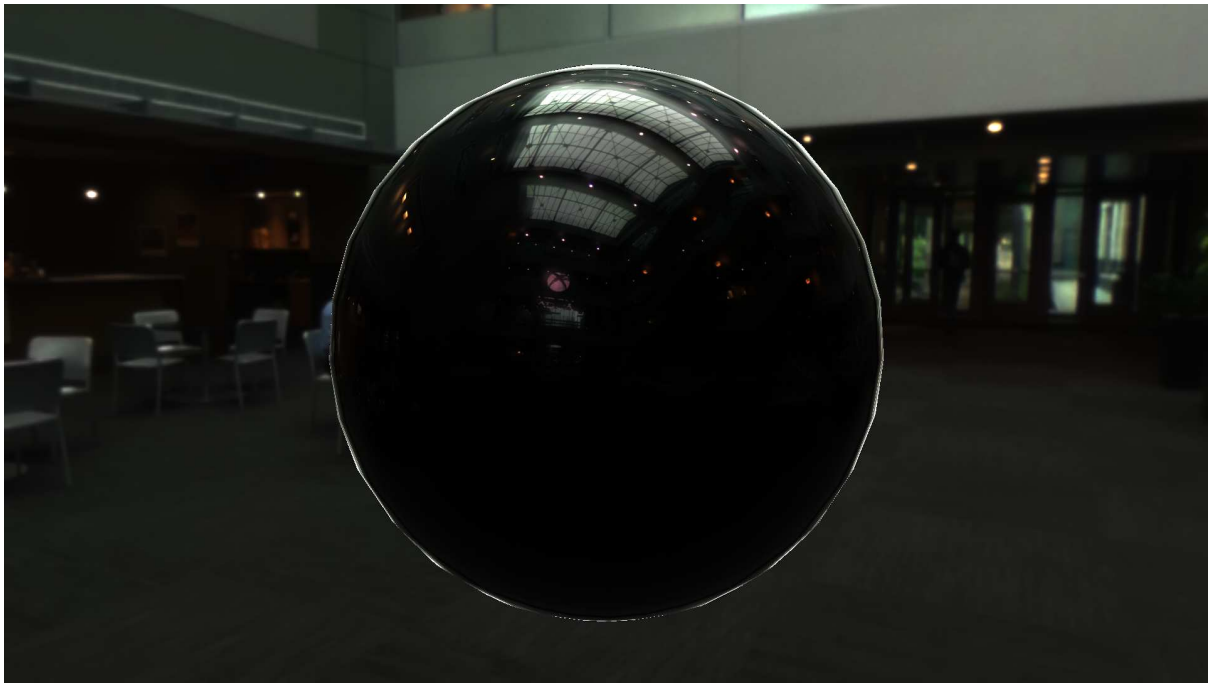


FIGURE 5.25. Overbrightening at grazing angles with the Earl G Function.

The Earl approximation also produces some overbrightening in general, particularly at grazing angles. This seems to get worse with higher lobe roughnesses.

Baseline 4D FGD LUT	Belcour Split-Sum FGD	Karis FGD
1.438ms	1.491ms	1.475ms

TABLE 5.1. Average shader invocation times for the 2-flux model with various FGD forms.

The Belcour split-sum LUT performs slightly worse than the full 4D FGD LUT, likely owing to the extra evaluation of the analytical fresnel term per-layer to compute coefficients, rather than being encoded in

the LUT. The split-sum LUT performs roughly 3.6% worse on average, on the profiled test scene. This delta will stay constant over the number of layers, as the number of FGD samples required is linear with the number of layers. The Belcour LUT also reduces the number of texture samples required from 3 to just 1, which is significant as fetches to global memory are slow operations which can bottleneck shaders if there is not a large amount of available independent ALU work to do alongside it. For reference, The Belcour Split-Sum LUT is compared against the Karis Split-Sum LUT, which is only 1.1% faster. This means they have similar costs, despite the overall improvement in accuracy and expressiveness offered by the Belcour formulation.

5.4 An Analytical Approximation for Total Internal Reflection

Within the matrix multiplication loop, a correction factor is applied for Total Internal Reflection (TIR), stored in a 3D LUT. Dropping the TIR correction term from the 2-flux model entirely improved performance by 0.5ms, or $\sim$ 30%. This corresponds to an increase in arithmetic throughput: 48% ALU utilisation when the TIR term is dropped vs 34% with it, which in turn can likely be attributed to the better cache hit rate (90% L2 hits vs 64%). Scratch usage drops by about 100 bytes, which is significant in terms of keeping slow memory accesses out of the hot loop.

```
if (ior_ij < 1.0)
{
   tir_norm = TIR_lookup(float3(abs(ops[i].reflection_down.mean.z),
      hg_to_ggx(asymmetry_T_0i), ior_ij)) * ops[i].transmission_down.norm;

   ops[i].reflection_down.norm += tir_norm;
   ops[i].transmission_down.norm -= tir_norm;
}
else
{
   tir_norm = TIR_lookup(float3(abs(ops[i].transmission_down.mean.z),
      hg_to_ggx(asymmetry_T_0j_R), ior_ji)) * ops[i].transmission_up.norm;

   ops[i].reflection_up.norm += tir_norm;
   ops[i].transmission_up.norm -= tir_norm; //could go negative if
      transfer_factors produces negative norm?
```

```
}
```

LISTING 5.1. The code used in the layer evaluation to apply the TIR correction term.

Keeping the TIR term but at a lower precision (16 bit rather than 32) has no discernable impact on the performance. The branching in the snippet above should not cause thread divergence, as the layer parameters are uniform over the entire dispatch, and if instead of dropping the term entirely the TIR lookup is replaced with a fixed constant (A 'dummy' term), we see an improvement of 0.4ms, with ALU utilisation up to 43%, and similar L2 cache hits. This is about 15 microseconds difference to dropping the TIR entirely, which suggests that the overhead is likely in the post-processing done on the sampled TIR term (that is, multiplying and assigning it to the lobe factors). These results suggest that latency from the texture lookup is contributing significantly to the runtime. This suggests that fitting an analytical approximation to replace the table could improve runtime significantly. This approach is suggested by Belcour as further work in the original paper. Alternatively, improving the register usage in order to allow more thread occupancy could assist with hiding the latency from these texture accesses.

When a light ray is transmitted through a medium, some proportion of it is reflected at the interface boundary. There is some configuration of angle for which all the light is reflected at the boundary, and so the amount of internal reflection is total. Belcour's transmission operator does not account for internal reflection, and so a weight is applied to the transmitted energy to correct for this phenomena. This TIR term is defined as the integral of the Fresnel transmission term and the GGX (D) geometry term over the hemisphere. Formally:

$$\text{TIR} = \int_{\Omega} (1 - F(\omega_t)) D(\omega_t, \alpha_{23}) d\omega_i \tag{5.5}$$

where ω_t and ω_i are the transmitted and incident rays, and α_{23} is the roughness of the interface between layers two and three - that is, the media on both sides is neither the substrate (layer 0) or air (layer n).

As there is no closed form of this integral (Belcour, 2018), it has to be estimated numerically, for example via Monte Carlo estimation. It is a 3D function, parameterised by the incoming light direction, the IOR of the 2 media the light transitions between, and the roughness (α) of the interface between the 2 media. In the original paper, the integral is precomputed into a 3D LUT, which is then sampled at runtime during layer evaluation.

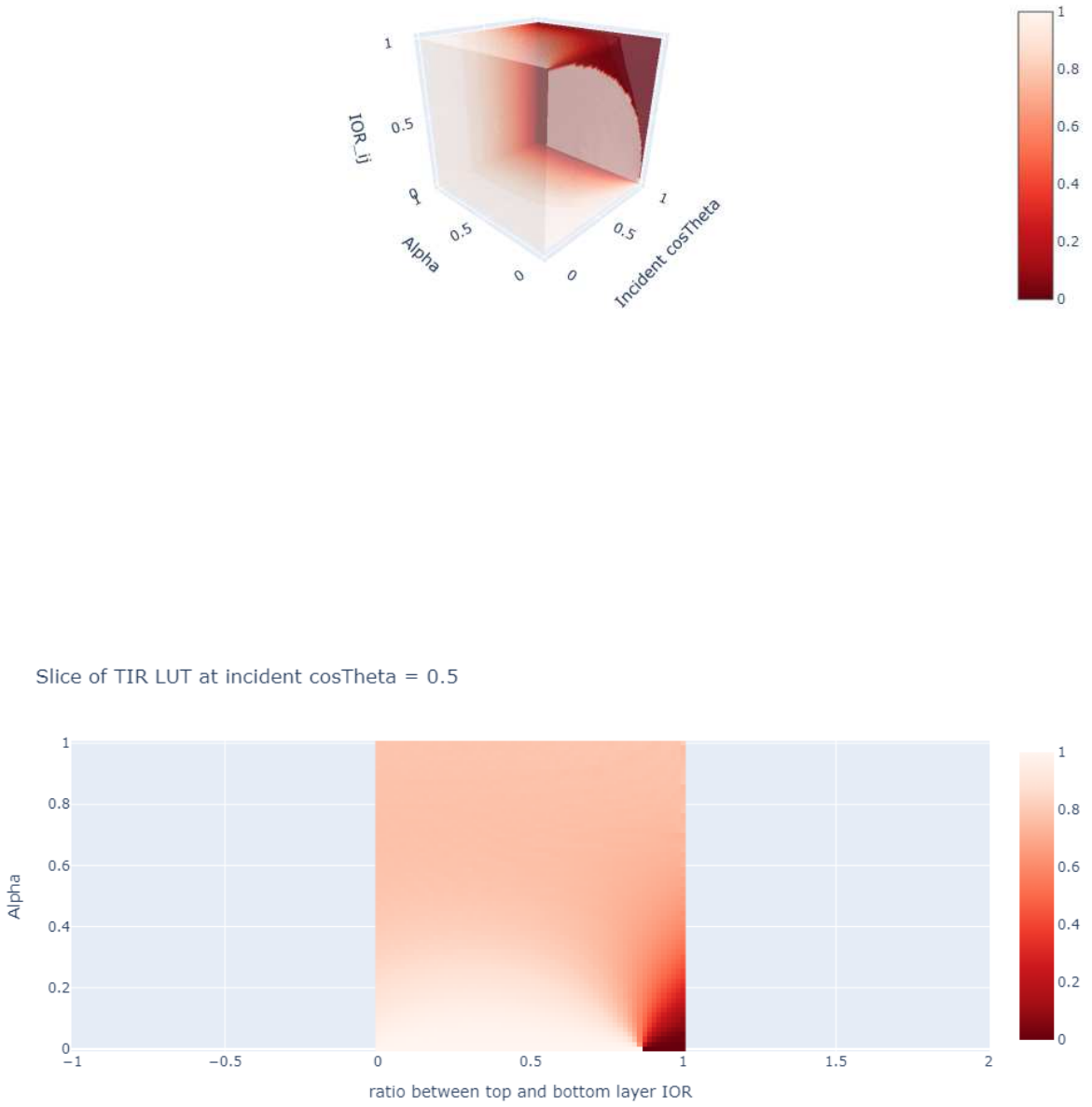
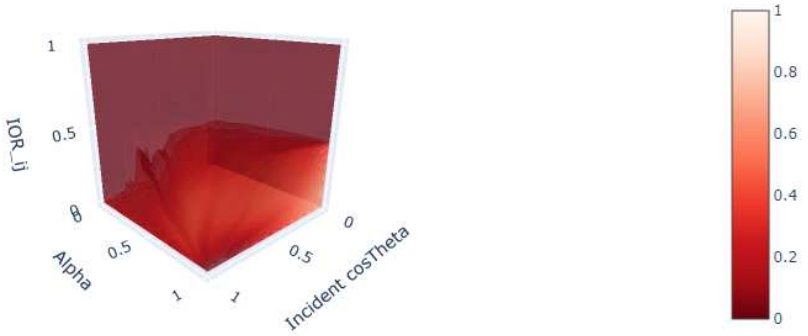


FIGURE 5.26. Visualisations of the TIR function over its parameter space. Brighter values mean less of the energy is 'lost' to TIR when transmitting through the medium.

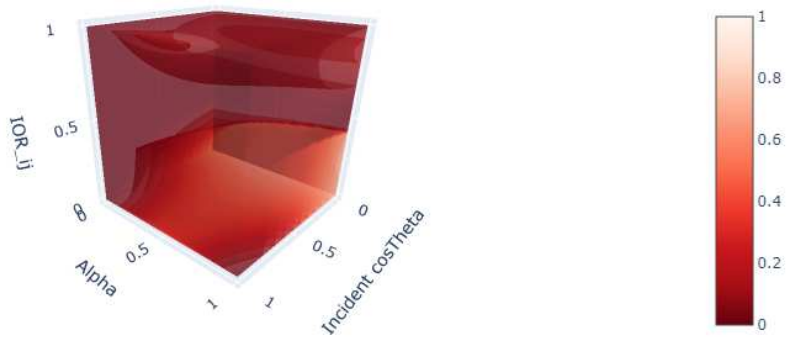
A naive estimator is also given by (Belcour, 2018), which evaluates the transmittance by the mean outgoing direction, assuming a clear-coat top layer. This is plotted below:



This is obviously far less accurate than the decorrelated integral used to generate the LUT in the original paper, with a mean-squared error of 0.464, but it is much simpler to approximate. Given that the only term estimated stochastically is the outgoing lobe direction as this is dependent on the vNDF for the GGX BRDF, finding some suitable approximation involves modelling analytically the distribution of microfacet normals produced by the vNDF. I re-use the off-specular approximation by Lagarde and De Rousiers to model the shift in microfacet normal, which roughly compensates for the differences in fresnel transmittance over different roughnesses. Formally, the approximation is defined as:

$$\begin{aligned} \text{TIR} &= (1 - F(\omega_o, \eta_{10}))F(\omega_o \cdot \omega_\alpha, \eta_{12})G_1(\omega_s, \alpha_{23}) \\ \omega_\alpha &= \textsc{OffSpecular}(H, \omega_o, H \cdot \omega_i, \alpha_{23}) \\ \omega_s &= -\textsc{Reflect}(\omega_o, \omega_\alpha) \\ H &= \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{5.6}$$

Where η_{10} and η_{12} are the indices of refraction between the media below and above the interface respectively, and G_1 is the Smith geometry term. This approximation is plotted below:



This approximation has a mean-squared error of 0.0724, with the maximum error being 50%, and converging to 0 at higher roughness. This approximate form tends to overestimate the correction term at the mid-range roughness and high incident angles, which will lead to an underestimation of the light lost to TIR when calculating transmittance.

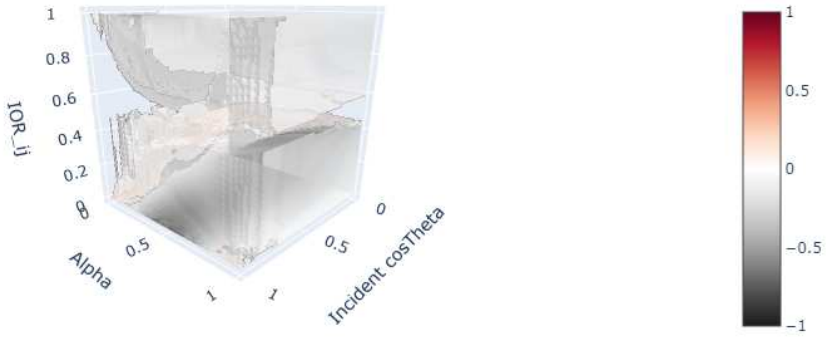


FIGURE 5.27. Degree of error between approximation and reference (naive reference minus approximation). Values below 0.01 are culled as being not significant.

Dropping this approximation back into the implementation shows an average runtime of 1.635ms. This pessimises performance by about 200 microseconds compared to the baseline. This is correlated with an increase in VGPR allocations that reduces total warp occupancy from 3 waves to 2, due to the increased pressure. There is however an increase in VALU throughput that implies if register pressure could be reduced, a small performance increase would be realised. This suggests that a lot of the potential speed improvements would be consumed by the relatively expensive Fresnel equations required to evaluate the TIR analytically. Using a cheaper form for Fresnel, such as Schlick, would improve this cost.

Baseline with full precision TIR	Baseline with half precision TIR	Analytic TIR	Dummy TIR constant	No TIR
1.433ms	1.435ms	1.635ms	0.985ms	0.944ms

TABLE 5.2. Average shader invocation times for the 2-flux reference scene with various TIR corrections.

CHAPTER 6

Discussion

The overall conclusion to be made here is that, despite being theoretically feasible to fit the transfer matrix framework to a realtime context, it is ultimately too heavy and impractical to be used in production scenarios, in a way fundamental to the architecture of the model that makes further optimisation difficult. While the 2-flux model comes within the range that could be feasible for production use, at just over a millisecond, it is both heavier and less useful than other existing layering frameworks, such as Belcour's or the Substrate (2023) framework, being only able to represent optically thin coatings over a conducting substrate. The 6-flux model, which introduces more advanced features such as intra-layer multiple scattering is closer to parity with other layering frameworks, however is far too expensive to be considered for realtime applications, demonstrated by the register and cache saturation at even just 2 layers. The performance problems relate directly to the architectural design of this model, making it ill-suited for pixel shading hardware on GPUs. Where other approaches such as the *Substrate* framework has granular control over the features that can be enabled, such as the ability to opt of coloured transmittance, allowing for users to scale their surfaces to an appropriate performance and fidelity tradeoff, the transfer matrix formulation enforces a tight dependency between all the interaction paths that exist for a layer, and so such fine, seperable control is not possible - one pays for all simulated features (such as multiscattering, thick media, complex IOR and coloured transmittance) even if the desired surface can be approximated sufficiently with only a subset.

The fundamental problem with the transfer matrix model is its size. All material layering approaches suffer from a blowup in interactions to compute, however the transfer matrix model requires the maintenance of very large data structures for the majority of its runtime - the transfer matrices themselves. As mentioned, the energy and asymmetry matrices for the 2-flux model occupy 48 and 16 bytes respectively. The evaluation loop requires at least 2 of these to be alive at any time - the previous matrix, and the new matrix to multiply the previous by - and so this becomes 128 bytes. On the profiled hardware, this translates into roughly 32 registers allocated for just the matrices. If the compiler decides to unroll

the evaluation loop, this increases in size. This increase depends on how much the compiler was able to recognise and reuse stale registers, but it will be roughly linear in the limit. In addition to this, the component setup step, which maps the input parameters to the Henyey-Greenstein representation of the layer, requires 4 instances of a 28 byte `henyey-greenstein` structure, for an additional 28 registers per layer. This high register use limits throughput on GPU hardware, as the scheduler is unable to dispatch as many wavefronts due to limited register availability. For comparison, a single 'simple' layer in the 'Substrate' framework - which maps most closely to the features supported by the 2-flux transfer matrix model - consumes just 56 registers (Hillaire & De Rousiers, 2023), compared to the 78 required by the minimum size layer stack (2 layers) with the 2-flux matrix. This is despite fewer supported features in the latter model. This register pressure is significantly worse for the 6 flux model, even when exploiting the sparsity present in the 6-flux matrix. The dependence on the matrix multiplication for layer evaluation means that this register use is not easily separable, and so can not be amortised over the shader execution in order to reduce the peak register allocation (e.g. by calculating certain components in sequence, rather than simultaneously), nor is it particularly feasible to reduce pressure by disabling certain features, harming the scalability of the model. Further, the presence of register spills significantly bottlenecks ALU throughput. This is a core limitation that makes the impact of applying any individual optimisations (for example, common approximations for D and G present in the literature) small or even nil. The single most important optimisation that can be made is in capping the maximum number of layers to a relatively small value, up to 4 for the 2-flux model, and potentially only 2 for the 6-flux model. This is a significant limitation with respect to the generality of the model. A 2 layer cap in particular makes the model no more versatile than any other clearcoating model, which can be approximated both more accurately and far cheaper.

In addition, the model is less general than it appears conceptually. There is a not insignificant degree of branching required in the layer evaluation, as dielectrics, media and substrate must be special-cased in order to update the required statistics correctly. For this implementation, where there is no spatial variation in BSDF parameters, this does not harm occupancy significantly because code paths are uniform across a wavefront. However, in a production context where texturing (i.e. dynamic spatial variation) is a given, this will likely introduce significant divergence across threads, which will cause unnecessary work to be done and stalls in execution to occur due to wave-level parallelism breaking down. The use of a complex IOR based parameterisation, rather than the more common F0/F90 parameterisation, means that there is a reliance on both a significantly more complex Fresnel formulation, and a significantly heavier FGD LUT than the Schlick Fresnel and Karis Split-Sum FGD typically employed in realtime

renderers. The Belcour et al. split-sum LUT eliminates much of the size and sampling cost of the 4D FGD LUT, however this is traded for 3 additional runtime evaluations of the fresnel term, which is not insignificant. This overall cost is still dominated by the layer evaluation however.

Performance aside, fitting the model to a preintegrated context was relatively painless. Extending the implementation to further light types would be simple, due to the reciprocity of the model, owing to Belcour's lobe parameterisation and the symmetry of the Henyey-Greenstein phase function (Randrianandrasana et al., 2021). The real-time results are close to the path traced reference, however misses much of edge tint at grazing angles. It is unclear why this is the case, other than errors due to resampling of the FGD LUT. That being said, the limitation to GGX lobes excludes a wide range of diffuse materials common in contemporary scenes. In practical terms, this model is only appropriate for modelling forms of painted or patinaed metals. This makes it less useful than existing material layering or single-layer models, although it's not a given that fitting statistics for a diffuse lobe type would be impossible, albeit likely to introduce further branching in the implementation. The primary improvement in surface representation is in the multiscattering present in the 6-flux model. The support for retroreflection and subsurface scattering is significant in allowing for rich and accurate layering combinations.

CHAPTER 7

Threats to Validity

There are a few issues with the execution of this research that pose a threat to its overall validity. This is mostly to do with small and homogeneous sample sizes in experiments, but there are some larger issues with experiment scenarios and design that are problematic for generalising this implementation to a production environment. The single biggest issue with this research is that the model, as currently implemented is *not* spatially varying. This means that the layer parameters are constant across the entire surface - every point's parameters[1] are identical to every other point. There is, in principle, nothing that prevents this model on a technical level from being spatially varying, indeed Randrianandrasana et al. show it to be in the original paper. The issue is one of performance. Pixel shaders are executed per-pixel, in large groups called 'waves' or 'wavefronts'. Every thread in the wave executes the same instructions in the same program in lockstep, it is only the data being processed that varies across threads. When encountering a branch, if the outcome of the branch diverges across threads in the wave, the GPU will typically have to serialise the execution over the branch - executing the threads that take path A first, then executing the threads that take path B, then performing a sync to wait for both groups of threads to return to lockstep execution[2]. This harms the overall paralellism of the wave, ultimately reducing throughput as threads are being used without doing useful work, extending the execution time. In a spatially varying case, avoiding divergence takes careful programming to scalarise all branches, or otherwise reformulate branches as functions of data, rather than control-flow, e.g. via a `lerp` or a `step`. However, in the non-spatially varying case of this model, all branches are scalar by default, since they ultimately depend on the layer parameters, which do not vary from pixel to pixel or thread to thread. As production environments demand spatial variation, this invalidates the performance profiling done in this research if branch divergence turns out to be a significant problem for a spatially varying implementation. There are two main sections of the code where divergence could appear: in the

[1] Note: this is distinct from the BRDF output itself, which does vary.

[2] Or some variation of this, e.g. some hardware will execute both sides of a conditional and discard the result on the failing side.

lobe component setup (`components_transfer_factors`) and the outgoing lobe transfer matrix evaluation (`outgoing_lobes`), as well as anywhere there is a fresnel evaluation. These all branch based on the classification of the current layer - dielectric, conducting, or a transmitting medium. The 6-flux model also has branching in the transfer matrix evaluation based on secondary effects - forward and backwards reflectance and transmittance, which increases the number of permutations that could potentially diverge.

For ease of implementation, lobe stacks are allocated statically within the shader itself. This means that, naively, registers and memory are allocated for all layers up to the maximum (5 by default), even if fewer layers are actually present in the stack. I rely on the compiler static analysis and GPU driver JIT compilation to factor out these unnecessary allocations, but testing has shown that setting the maximum layer constant to a smaller value has a significant effect on timings for all layer permuations, implying that there is waste occurring. The performance implications of a true dynamic parameter buffer is however not clear, as this would essentially require using some global memory buffer as scratch space, rather than thread-local stack/static allocations within the shader itself. This would in theory make operations dependent on lobe parameters much slower due to the increased distance to memory, as well as having to now consider potential thread contention to the global resource, so gains would only be realised if the reduced register waste resulted in an equally significant occupancy increase, in order to hide the memory access latency.

Similarly, given that this model was only benchmarked on a single machine, it is difficult to generalise the profiling results to the wide range of hardware available on the consumer market. For example, the double-rate VALU throughput afforded to half-precision floats (even if this did not appear to materialise in practice) is only available on AMD RDNA2 architectures. This may make the FP16 optimisation useless or even pessimistic on other hardware, such as those offered by NVIDIA or Intel. The benchmark hardware too is relatively high-end, and so hardware even within the same architectural class but on the lower end of the market may have different performance characteristics, or simply not work. This was in fact the case for some testing done on a laptop using Intel integrated graphics, which simply refused to launch the renderer due to a TDR timeout - possibly caused by exhausting its memory, or otherwise just being too heavy for the chip to keep up with the OS keep-alive pings.

Ultimately performance profiling is highly sensitive to implementation decisions and profiling hardware, and so a comprehensive picture would require significantly more time and resources than was available in this project in order to build a much larger and wider sample set.

CHAPTER 8

Limitations & Future Work

The key limitation of this research is, as already discussed, the lack of spatial variation in the implementation of the model. This is a significant problem for its usefulness in a production context, which requires textured assets to support the high levels of detail expected by contemporary productions. Randrianandrasana et al.'s original implementation does support spatial variation, and implementing it in this research's implementation would be relatively simple, by swapping the constant buffer of layer parameters for an array of textures to sample. This requires significantly more bookkeeping in the renderer back-end, and so the time and energy cost for implementing this feature was deemed not worth the relatively small contribution it would have to the research. As mentioned in the previous chapter, this poses a threat to the validity of the existing performance profiling results, and so further work may be necessary to scalarise the branching in this implementation of the model.

Given the large blow-up in size suffered by many layering frameworks, including this one, static and dynamic simplification passes similar to those described by Hillaire and De Rousiers for the Substrate framework are necessary for deploying to real-time renderers. The matrix decomposition of layer stacks is convenient for this kind of analysis, as layers can be composed and decomposed through matrix products. There is room for investigating *how* to apply such a preprocessing pass to this model, which may help with mitigating the size constraints it has currently. Fixing the layer topology at compile time is not unreasonable for production use, as bound resources such as textures and constant buffers must already be fixed at compile time, and this is heavily dependent on the layer topology, implying that this must also be known and fixed at compile time. Situations where layers truly do need to be appended and removed at run-time are generally rare, and it would still be possible for artists to implement such an effect by simply binding all resources for the maximum number of layers - that is, their layer stack is n layers, even if $m < n$ are used for the majority of the time. Artists would have to accept the wasted resources and lost performance from unused layers, but this is already the case for attribute

blend systems currently employed such as in *Unreal Engine 4*, where attributes with a runtime weight of 0 - indeterminate at compile time - are still evaluated.

The current parameterisation of the model, using IOR as the primary definition of the media is somewhat unintuitive, and it is possible to produce layer stacks that are not physically accurate. Parameters are not independent, as changes in lower layer parameters can change the appearance of upper layers, as many calculations are dependent on the ratio between two layers' indices of refraction. In a technical sense, an IOR based parameterisation *is* the most accurate form, however most renderers (including the one used in this research) are tristimulus renderer, only accounting for 3 perceptual output bands - Red, Green and Blue, while the Fresnel equations are spectral equations and must be evaluated on physical spectral samples. As such, the process used in this research (and many other renderers) where IOR is mapped to RGB on an ad-hoc basis is not physically accurate. A parameterisation based on the perceptual reflectance of surfaces - rather than their spectral characteristics - is more suitable for a tristimulus renderer. Hoffman, in *Fresnel Equations Considered Harmful* (2019) proposes a simple extension to the common Schlick Fresnel approximation used in contemporary real-time renderers that compensates for its approximation error. This can be combined with an F_0/F_{90} parameterisation, such as currently used in the 'Substrate' framework, which maps the perceptual reflectance of a surface at 0 degrees and 90 degrees to the Schlick Fresnel function, or any of its extensions. This parameterisation is more intuitive for artists, due to being based on perceived rather than physical quantities. This brings back the 'sight-reading' property discussed earlier with respect to the 'Disney' model, which contributed significantly to its success in production environments (Burley, 2012). Fitting the transfer matrix model to a perceptual parameterisation like the ones discussed would improve the usability of the model and make it more suitable for production use by artists.

There is room for further work in the expressiveness of the model. For example, fitting de Dinechin and Belcour's Diffuse layered materials to the transfer matrix model would allow it to reach parity with existing physically based models, such as the Disney Principled (Burley, 2012) model. This would allow for the simulation of Lambertian surfaces within the layering framework, considered the general model for non-metallic 'diffuse' surfaces such as plastic, ceramics, wood, etc. de Dinechin and Belcour's model builds off the statistical framework developed in *Efficient Rendering of Layered Materials Using an Atomic Decomposition with Statistical Operators* (Belcour, 2018), which is in turn adapted for use in Randrianandrasana et al.'s transfer matrix model. This implies that this extension is suitable for

integration into an extended transfer matrix model. The key difficulty will be in finding a lobe representation that can be convolved with the existing Henyey-Greenstein representation used in the transfer matrix model for GGX lobes.

CHAPTER 9

Conclusion

This research has demonstrated that the 2-flux transfer model would be feasible for use in a realtime context. It has also shown that the more expressive 6-flux model is too heavy for realtime performance constraints. Despite the theoretical feasibility, there are fundamental problems with the architecture of both variants of the model that make it ultimately unsuitable for production use in real-time renderers. In the course of fitting the model to a realtime renderer, the sampling scheme was modified to use preintegrated lighting, and Belcour et al.'s split-sum FGD approximation was used in place of the less accurate and heavier 4D FGD LUT from Randrianandrasana et al. paper. A comprehensive analysis of the model's performance was completed, and a number of optimisations were applied and tested. During the course of optimising and profiling, an analytical approximation of Belcour's naive total internal reflection estimator was developed. There is room for further work to improve the accuracy and performance of this approximation.

There are further open problems in improving the parameterisation of the model and fitting feature extensions. The model's suitability for production use is contingent on the future development of some method to greatly reduce its size. However, the flexibility and general nature of the transfer matrix model is worth continued investigation with, toward the goal of creating a more general and flexible material design framework for artists.

Appendix

Notation

- ω_i - Incident light direction.
- ω_o - Outgoing light direction.
- H - The half-vector, defined as $\omega_i + \omega_o$, normalised.
- N - The normal vector, a vector perpendicular to a surface.
- α - Microfacet roughness parameter for an interface.
- η - Index of Refraction.
- κ - The complex absorption component of IOR.
- G_1 - The Smith microfacet masking function.
- G - The Smith microfacet masking-shadowing function.
- D - The GGX vNDF function.
- F - The Fresnel function.
- Ω - The set containing all ω_i in the hemisphere.

Glossary of Terms

- BxDF - Bidirectional 'x' Distribution Function. A function that determines the direction, location and energy a given ray of light will have once it has interacted with the material. The 'x' is substituted for the phenomena that the model simulates.
- BRDF - Bidirectional Reflectance Distribution Function. BxDF that only models ray reflectance off the surface, assuming no transmission occurs. Examples include the Blinn-Phong BRDF and Cook-Torrance, or Lambert.
- BTDF - Bidirectional Transmission Distribution Function. A BxDF that models ray transmission through the surface.

- BSDF - Bidirectional Scattering Distribution Function. Can be understood as the composition of a BRDF and a BTDF. Models ray reflection, absorption, diffusion, transmission and scattering within media.
- Microfacet - a microscopic (practically speaking, sub-pixel) facet on a surface. Usually represented as a distribution function that models the correlation of reflectance rays in response to smooth and rough surfaces.
- Dielectric - 'non-metallic', not a conductor. Many materials are categorised as either dielectric or metallic, as there is usually a very large distinction in specular reflectance between them - 5-10% for dielectrics, 80-100% for metals.
- Texel - The smallest unit in a texture or image. Distinguished from 'pixel' as when textures are rendered on screen, their units may be composed of several screen pixels.
- Ubershader - a material model that aims to encapsulate a wide range of possible materials in a single model. Note that this is distinct from the notion of layering - the user has no control over arbitrary layering.
- GPU - Graphics processing unit.
- UI - User Interface.
- TIR - Total Internal Reflection. Phenomena where all energy in a light ray is reflected at an interface boundary, with none of it transmitted to the next medium.
- FGD - Directional albedo term for a surface, computed by integrating over the hemisphere of Fresnel F, microfacet distribution D and masking-shadowing G terms.
- GD - The microfacet response term, comprising just the microfacet distribution D and the masking-shadowing function G.
- LUT - 'Look-Up Table', a precomputed list of data that can be indexed or 'looked up' by a given set of parameters.
- GGX - 'Ground Glass Unknown', a particular formulation of the normal distribution D term (developed independently by several authors), commonly used in contemporary renderers for its accuracy and efficiency. Also known as Trowbridge-Reitz (Heitz, 2014).
- IBL - 'Image Based Lighting'. A lighting method where photometric images, rather than analytic models of light sources, are used to inject lighting into a scene. Often used for modelling accurate ambient lighting.

- vNDF - 'Distribution of Visible Normals Function'. The D term in the microfacet equation, modelling the field of visible microfacet normals from a particular viewpoint. Common distributions include GGX, Phong and Beckmann (Heitz, 2014).
- PDF - 'Probabilities Distribution Function'. Function describing all possible values and likelihoods of a random variable.
- IOR - Index of Refraction. Determines the degree to which light is bent when passing from medium to medium.
- SIMD - Single Instruction Multiple Data. A hardware architecture where a number of parallel threads run identical programs and are executed together in lockstep, while the data operated on by each thread may vary.
- VGPR - Vector General Purpose Register. A single vector register in a GPU's SIMD core.
- VALU - Vector Arithmetic & Logic Unit. The main computational unit in a GPU SIMD core, responsible for floating point arithmetic and logic operations across the whole wavefront.

Code

There is a repository containing the fork of the 'MiniEngine' used in this research available at:

https://github.com/ssav7912/TransferMatrix/tree/transfer-matrix

Follow the instructions to build and run in the README.md provided. Make sure you have the 'transfer-matrix' branch checked out.

Supplementary materials, such as python notebooks used to generate the comparisons and some of the LUTs are provided in a separate repository:

https://github.com/ssav7912/TransferMatrixSupplementary

References

AMD. (2023). Rdna3 instruction set architecture: Reference guide [Computer software manual].

Belcour, L. (2018, jul). Efficient rendering of layered materials using an atomic decomposition with statistical operators. *ACM Trans. Graph.*, *37*(4). Retrieved from https://doi.org/10.1145/3197517.3201289 doi: 10.1145/3197517.3201289

Belcour, L., Bati, M., & Barla, P. (2020, August). *Bringing an Accurate Fresnel to Real-Time Rendering: a Preintegrable Decomposition.* SIGGRAPH'20 Courses. Retrieved from https://inria.hal.science/hal-02883680 doi: 10.1145/3388767.3407325

Blinn, J. F. (1977). Models of light reflection for computer synthesized pictures. In *Proceedings of the 4th annual conference on computer graphics and interactive techniques* (p. 192–198). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/563858.563893 doi: 10.1145/563858.563893

Burley, B. (2012). Physically based shading at disney. *Walt Disney Animation Studios*. Retrieved from https://media.disneyanimation.com/uploads/production/publication_asset/48/asset/s2012_pbs_disney_brdf_notes_v3.pdf

Curtis, N., & Fanfarillo, A. (2022). *Intro to register pressure in amd compilers.* Retrieved from https://www.olcf.ornl.gov/wp-content/uploads/Intro_Register_pressure_ORNL_20220812_2083.pdf

de Dinechin, H., & Belcour, L. (2022, May). Rendering layered materials with diffuse interfaces. *Proc. ACM Comput. Graph. Interact. Tech.*, *5*(1). Retrieved from https://doi.org/10.1145/3522620 doi: 10.1145/3522620

Guarnera, D., Guarnera, G., Ghosh, A., Denk, C., & Glencross, M. (2016). Brdf representation and acquisition. In *Computer graphics forum* (p. 625-650). Retrieved from https://doi.org/10.1111/cgf.12867

Guo, J., Li, Z., He, X., Wang, B., Li, W., Guo, Y., & Yan, L.-Q. (2023, 12). Metalayer: A meta-learned bsdf model for layered materials. *ACM Transactions on Graphics*, *42*, 1-15. doi: 10.1145/3618365

Guthmann, F. (2023). Retrieved from https://gpuopen.com/learn/occupancy-explained/

Heitz, E. (2014, June 30). Understanding the masking-shadowing function in microfacet-based brdfs. *Journal of Computer Graphics Techniques (JCGT)*, *3*(2), 48–107. Retrieved from http://jcgt.org/published/0003/02/03/

Hery, C., Villemin, R., & Ling, J. (2017). Pixar's foundation for materials. *Pixar*

Animation Studios. Retrieved from https://graphics.pixar.com/library/PxrMaterialsCourse2017/paper.pdf

Hillaire, S., & De Rousiers, C. (2023). Unreal engine substrate: Authoring materials that matter. In *Advances in real-time rendering in games, siggraph 2023*. Retrieved from https://advances.realtimerendering.com/s2023/2023%20Siggraph%20-%20Substrate.pdf

Hoffman, N. (2019). Fresnel Equations Considered Harmful . In R. Klein & H. Rushmeier (Eds.), *Workshop on material appearance modeling*. The Eurographics Association. doi: 10.2312/mam.20191305

Hoffman, N., Gotanda, Y., Martinez, A., & Snow, B. (2010). Physically based shading models in film and games production. In *Siggraph 2010 course notes*. Retrieved from https://renderwonk.com/publications/s2010-shading-course/hoffman/s2010_physically_based_shading_hoffman_a_notes.pdf

Jakob, W., d'Eon, E., Jakob, O., & Marschner, S. (2014, jul). A comprehensive framework for rendering layered materials. *ACM Trans. Graph.*, *33*(4). Retrieved from https://doi.org/10.1145/2601097.2601139 doi: 10.1145/2601097.2601139

Karis, B. (2013). *Real shading in unreal engine 4* (Tech. Rep.). Epic Games. Retrieved 2016-02-05, from http://blog.selfshadow.com/publications/s2013-shading-course/karis/s2013_pbs_epic_notes_v2.pdf

Lagarde, S., & De Rousiers, C. (2014). Moving frostbite to physically based rendering 3.0. In *Proceedings of physically based shading in theory and practice, siggraph 2014 course notes*. Retrieved from https://seblagarde.wordpress.com/wp-content/uploads/2015/07/course_notes_moving_frostbite_to_pbr_v32.pdf

Laine, S., Karras, T., & Aila, T. (2013). Megakernels considered harmful: wavefront path tracing on gpus. In *Proceedings of the 5th high-performance graphics conference* (p. 137–143). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2492045.2492060 doi: 10.1145/2492045.2492060

Mack, C. (2007). Appendix c: The dirac delta function. In *Fundamental principles of optical lithography* (p. 495-500). John Wiley Sons, Ltd. Retrieved from https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470723876.app3 doi: https://doi.org/10.1002/9780470723876.app3

Naty, H. (2009). *Deferred lighting approaches*. Retrieved from https://www.realtimerendering.com/blog/deferred-lighting-approaches/

Pharr, M., & Humphreys, G. (2023). *Physically based rendering, fourth edition: From theory to implementation* (4th ed.). San Francisco, CA, USA: MIT Press.

Randrianandrasana, J., Callet, P., & Lucas, L. (2021, jul). Transfer matrix based layered materials rendering. *ACM Trans. Graph.*, *40*(4). Retrieved from https://doi.org/10.1145/3450626.3459859 doi: 10.1145/3450626.3459859

Stanford. (1996). *Stanford 3d scanning repository*. Stanford University Computer Graphics Lab. Retrieved from http://graphics.stanford.edu/data/3Dscanrep/

Walter, B., Marschner, S. R., Li, H., & Torrance, K. E. (2007). Microfacet models for refraction through rough surfaces. In *Proceedings of the 18th eurographics conference on rendering techniques* (p. 195–206). Goslar, DEU: Eurographics Association.

Weidlich, A., & Wilkie, A. (2007). Arbitrarily layered micro-facet surfaces. In *Proceedings of the 5th international conference on computer graphics and interactive techniques in australia and southeast asia (graphite '07)* (p. 171-178). New York, NY, USA. Retrieved from `https://doi.org/10.1145/1321261.1321292`