Interactive Dynamic Volume Illumination with Refraction and Caustics

Jens Gåsemyr Magnus, author

Stefan Bruckner, supervisor

Master Thesis

University of Bergen
Department of Informatics

Acknowledgments

I would like to thank Professor Stefan Bruckner for supervising this thesis and for the great discussions we frequently had giving rise to the solutions presented in this thesis. I would also like to thank Itai Kallos for informative discussions giving insight in the physics behind our light model.

Abstract

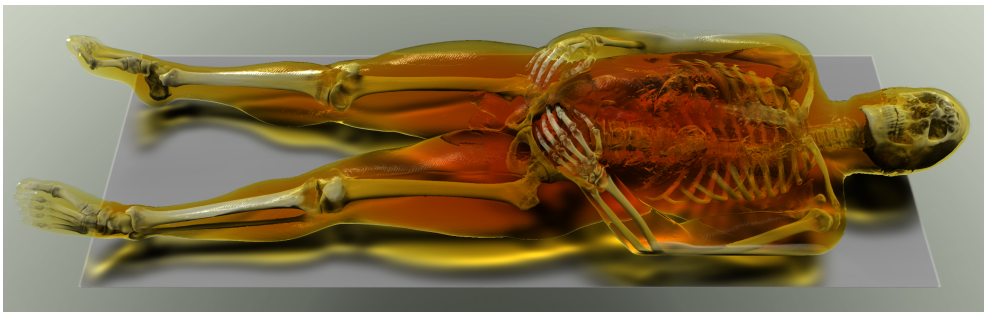


Figure 0.1: The visible human CT dataset rendered using the proposed technique – the skeleton is embedded in a colored transmissive medium with higher refractive index than its surroundings.

In recent years, significant progress has been made in developing high-quality interactive methods for realistic volume illumination. However, refraction – despite being an important aspect of light propagation in participating media – has so far only received little attention. In this thesis, we present a novel approach for refractive volume illumination including caustics capable of interactive frame rates. By interleaving light and viewing ray propagation, our technique avoids memory-intensive storage of illumination information and does not require any precomputation. Propagation of refracted illumination is realized by employing a Semi-Lagrangian backward integration scheme, inspired by texture advection from the field of texture-based flow visualization. It is fully dynamic and all parameters such as light position and transfer function can be modified interactively without a performance penalty.

Contents

Chapter 1

Introduction

1.1 Motivation

Interactive volume rendering refers to the rendering and visualization of 3D scalar fields, also called or volumes, at frame rates high enough as to allow user interactions such as changing camera positions. Volume rendering is widely used the field of scientific visualization and for rendering volumetric visual phenomena, such and atmospheric effects and smoke. Commonly, volume rendering is done by assigning different optical properties to the values of a scalar field. In the context of medical visualization, scalar values in volumes often represent physical materials, such as bone or soft tissues. If the aim of a visualization is to show bone structures, one could assign visible optical properties to the scalar values associated with bone.

Advanced illumination effects, such as shadowing, ambient occlusion and light scattering, can provide important visual cues for the perception of complex spatial structures [1]. However, for the sake of performance and to allow interactive frame rates, simplified illumination models are often employed. This is particularly true in the context of rendering volume data where, due to the lack of discrete homogeneous objects, illumination contributions must in principle be evaluated and propagated at every point in space. Research in recent years, partly fueled by the performance and flexibility advances of current GPUs, has successfully developed efficient high-quality methods for interactively rendering volume data with advanced illumination effects. However, one aspect that has been neglected so far is refraction, i.e., directional changes in light propagation due to differences in the speed of light between transmission media. Despite the fact that translucency is commonly used in the visualization of volume data, refraction effects are typically ignored and all rays are treated as straight. In this thesis we aim to enable interactive rendering of

refraction and present an approach capable of rendering refracted illumination for volume rendering at interactive frame rates.

1.2 Refraction

Light is refracted when the speed at which it can propagate in a medium changes. When transitioning to a medium where the propagation speed is slower, the light will change direction toward the the slower medium. Refraction is responsible for a wide range of optical phenomena that often significantly affect the appearance of transparent materials. For instance, the directional change towards the normal of a light ray entering a convex glass lens surrounded by air, which has a higher speed of light, will cause a magnifying behavior. Caustics are a well-known phenomenon caused by refraction. Caustics refer to the shadow patterns emerging when light passing through a curved surface is concentrated in some areas and dispersed in others. An example of this are the dancing patterns seen underwater due to a wavy water surface.

1.3 Problem and Contribution

As mentioned earlier, caustics form when photons are focused in some regions, leaving other regions shadowed. When rendering refraction and caustics, both the viewing rays and light rays are curved. This is problematic because tracking which light rays that contribute to the illumination of some point along a viewing ray is not trivial. Offline techniques, such as volumetric photon mapping presented by Gutierrez [2], aim to simulate this phenomenon directly, by shooting photons through a scene and storing illumination contributions where they land. These techniques require large amounts of photons to be simulated for the result to appear continuous, and are thus prohibitively expensive for interactive applications. Additionally, if particles are forward integrated like this, there is no guarantee that the result will be continuous. Regions entirely devoid of particles may appear.

We draw inspiration from the field of texture-based flow visualization, who was faced with a similar issue, and adopt the use of a semi-Lagrangian backward integration scheme. Rather than forward integrating the photons and depositing their values where they land, the values for each pixel are found by looking backward in time, in the direction of the photon movement, to find the value that is to be transported here. This approach guarantee a continuous result. Light is backward integrated and propagated through the volume in a plane-by-plane manner. By simultaneously propagating the light and viewing rays, illumination does only need to be stored for the current plane, we thus

avoid the need for an intermediate illumination volume. Our approach supports soft shadows and caustics, as demonstrated in Figure 0.1, at interactive frame rates. To the best of our knowledge, our work is the first fully interactive method capable of rendering volume data using advanced refractive effects. Furthermore, our technique was designed to avoid precomputation allowing fully dynamic manipulation of all rendering parameters.

1.4 Thesis Structure

First, in the following chapter (Chapter 2), we give an overview of scientific work related to this thesis. In Chapter 3, we cover the basics and theoretical background of GPU-based interactive volume rendering based on raycasting and slicing approaches. The main contribution of the thesis is covered in Chapter 4. Here we describe ray propagation schemes, light calculations and compositing. Following this, we discuss implementation details in Chapter 5. In Chapter 6, we go through a selection of images rendered with our implementation showing the capabilities of our approach, as well as performance tests. The final chapters, 7 and 8, are devoted to discussions and limitations, and conclusions and future work, respectively.

Chapter 2

Related Work

In this chapter we will cover previous work on advanced illumination techniques. First we go through previous technique for volume illumination, followed by techniques for rendering refractive effects and finally nonlinear raytracing, such as the modeling of mirages caused by atmospheric refraction. In principle, light transport in participating media is guided by the radiative transfer equation as described by Chandrasekhar [3]. However, due to the high computational costs of physically-based rendering, interactive techniques typically employ simplified optical models. As discussed by Max [4], common simplifications include the emission-absorption model which disregards scattering effects or approaches that only consider single scattering. In recent years, a number of interactive volume rendering methods have been presented that incorporate more advanced effects as comprehensively discussed in the survey by Jönsson et al. [5].

2.1 Volume Illumination, Shadowing and Occlusion

Ambient occlusion techniques aim to calculate shadowing from ambient light sources. The addition of ambient occlusion in a rendered image does not only help making images look more realistic, but can also help to make spatial structures more apparent by providing visual cues [1]. To enable dynamic ambient occlusion, Ropinski et al. [6] used local data histograms to speed up the computation when the transfer function is changed. The histograms provide information about the data value distribution in the neighbourhood of a voxel. Because holding one local histogram for each voxel would require large amounts of memory, the histograms are sorted into clusters using vector quantization. A 3D texture is used to map voxels to the cluster the particular voxels are associated with. The local histograms for each cluster are stored as

rows in a 2D texture that is used during interactive rendering. The histograms are computed independently of the current transfer function, allowing the transfer function to be specified without the need for further preprocessing. The histograms are used during interactive rendering to approximate ambient occlusion, subsurface scattering, color bleeding, and glow.

Hernell et al. [7] proposed a local variant of ambient occlusion which is calculated in a spherical neighborhood of a voxel. Their algorithm works by progressively accumulating ambient light contributions for each voxel from discrete directions in a spherical pattern. For each voxel they calculate the ambient light contributions from one direction at every frame and add it to the accumulated ambient light. The ambient light calculations are interleaved with the main rendering pass, allowing for user interaction although the ambient light computation may not be finished.

Schlegel et al. [8] used 3D summed area tables (SATs) over extinction coefficients for interactive directional soft shadows. To calculate the shadows they use the SAT to approximate the light extinction in a cone by summing over a series of cuboids. If the light source is moved, the light cache is recalculated. Their method also allows for the computation of ambient occlusion and color bleeding. This is done by calculating the extinction coefficients, using the summed area table, for progressively larger boxes around the point to be shaded, weighting the extinction coefficients by the inverse square of the corresponding radius. These calculations are independent of the light positions. Ament et al. [9] solve full light transport within finite spherical regions stored in a preintegration table and combine this information with local illumination and ambient occlusion. They build upon the observations made by Schlegel et al. that the spatial distribution of extinction coefficients has little effect on the final results. Additionally scattering effects from far away have negligible effects on a given location due to their exponential nature. This means that the spherical region that is preintegrated over can be small. Light transport is computed by Monte-Carlo simulation and stored in the preintegration table. The preintegration table is independent of specific data sets and transfer functions. In later work [10], Ament et al. also presented a technique that uses summed area tables to compute soft shadows for multiple directional and point light sources. Zhang and Ma [11] presented a method to compute light transport by solving a convection-diffusion equation. They also developed an approach for decoupled shading that separates global lighting evaluation from per-sample material shading [12].

Sunden et al. [13] proposed a selective update scheme to minimize computations when light settings are changed. Multiple light sources are propagated

through the volume using a technique named *coherence-base light propagation*. In this technique, lights are sorted in groups based on which principle axis their directions are the closest to. Lighting information is stored, for all lights, in a single illumination volume. By using a higher precision floating-point texture to store the illumination volume, energy from one or more light source can be added or subtracted freely without having to recompute the volume for all lights. This way updating illumination from one light source can be done simply by first subtracting the old illuminate cast by this light before reintroducing it with the new settings.

Kniss et al. [14] proposed an approach that allows for the interleaving of light and viewing computations, thereby eliminating the need for an intermediate illumination volume. They introduced the concept of half angle slicing, where slices are angled half way between the view direction and the light direction. Their method supports phase functions and direct illumination and was the first interactive volume rendering solution to incorporate multiple scattering [5]. Schott et al. [15] used a similar approach for a view-dependent approximation of ambient occlusion using incremental convolution. In their setup slices are aligned with the view plane. Occlusion for the current slice is calculated by averaging samples from the previous plane in a conical neighbourhood around the current fragment. Šoltészová et al. [16] extended this approach to enable user-specified light positioning. This was later extended by Patel et al. [17] who proposed an efficient convolution-based approach capable of generating dynamic soft shadows. They introduce a stochastic elliptical Dirac filter kernel achieving results similar to that of a Gaussian kernel with very few samples. Sunden et al. [18] proposed a plane sweep approach that also enables incremental lighting computations with a small memory footprint. Their method enables advanced illumination effects such as shadowing and scattering in a raycasting-based solution. While many efficient methods for interactive advanced volume illumination have been developed, none of the above approaches supports refraction.

2.2 Rendering of Refraction

Several methods for the real-time rendering of refractive effects on surfaces have been presented. Wyman [19] used an image-space approach to approximate refraction of a distant environment through two interfaces. They approximate the position of the second interface by estimating the depth along the refracted ray. The estimation is done by interpolating between the mesh depth along the view vector and along the surface normal. The interpolation is weighted based on the ratio between the angle of the incident ray to the normal and the angle of the refracted ray to the normal. The normals of the back faces are stored in a buffer, and can thus be retrieved at the estimated po-

sition of the second interface. In later work they extended this approximation to also handle caustics through either photon emission or photon gathering on the GPU [20]. Oliveira and Brauwers [21] used a similar method to enable the rendering of refractive deformable objects. Rather than approximating the intersections points of noninitial interfaces based on the angles of incidence and transmittance, they approximate them using a ray perspective-depth-buffer intersection solution. The intersection is done against the depth buffer of the back-faces. The technique by Davis and Wyman [22] additionally handles total internal reflection. They, like Wyman [19] and Oliveira and Brauwers [21], employ a screen space solution. They also intersect against the depth buffer, but do so using a binary search in the depth buffer to approximate, within a threshold, the intersection points in logarithmic time.

The rendering of refraction effects on rough objects was investigated by Walter et al. [23], who showed how existing microfacet models could be extended to simulate materials such as etched glass. They presented an offline rendering solution using importance sampling to speed up computations. De Rousiers et al. [24] proposed a real-time technique capable of rendering rough refractive materials using a combination of cone tracing and macro geometry filtering together with a pre-convolved environment map. Their cone tracing is done against back face depth buffers using the ray perspective-depth-buffer intersection described by Oliveira and Brauwers [21], but is done on four rays bounding a spherical Gaussian lobe. The back face buffers are downsampled and filtered such that this tracing allows them to approximate the refracted Gaussian lobe leaving the mesh, which is used for sampling the environment map.

2.3 Nonlinear Raytracing

An early approach to approximate ray propagation in media of non-constant refractive indices by repeated application of Snell's law was used by Berger et al. [25] to render atmospheric effects such as mirages. They simulate mirages by tracing rays through a *mirage box* consisting of multiple layers of varying index of refraction. Gröller [26] presented a general approach to nonlinear ray tracing for the visualization of mathematical and physical systems. Weiskopf et al. [27] developed a GPU-based technique for nonlinear ray tracing in the context of relativistic visualization, modeling phenomena such as gravitational lensing. Stam and Languenou [28] extended a standard ray tracing algorithm to handle non-constant media by employing the eikonal equation, for example, to model continuous variations of the refractive index in air causing mirages. Full global illumination of continuously refracting participating media can be performed using volume photon mapping, as proposed by Gutierrez et al. [2]. Their approach simulates curved light paths and inelastic scattering for the

rendering of atmospheric effects.

Ihrke et al. [29] achieved real-time frame rates for the rendering refractive surface models by utilizing a voxelized version of the model and precomputing light using wavefront tracking based on the eikonal equation. In their method, light is propagated incrementally as particles, bundled into packets of four forming a wavefront patch. At each step the intensity of the light is calculated based on the size of the wavefront patch, using the intensity law of geometric optics, and deposited into an illumination volume by rendering the particles as point primitives. Refraction is calculated based on the ray equation of geometric optics (derived from the eikonal equation) which models refraction in non-constant media. Sun et al. [30] proposed a similar solution for rendering voxelized surface models, but additionally allowing lighting parameters to be changed at interactive frame rates. They adopt the ray equation of geometric optics Ihrke et al. used. They employed an octree decomposition of the refractive index field to enable adaptive ray marching, allowing them to take larger steps when computing the illumination volume. Cao et al. [31] presented a raytracing approach that like Ihrke et al. and Sun et al.'s methods handle non-constant refractive media. However, instead of basing their refraction calculations on the ray equation of geometric optics, they employ the ray equation of gradient-index optics. This allows them to do larger steps when traversing the media in comparison to those based on geometric optics, and thus gaining performance.

Ament et al. [32] introduced the refractive radiative transfer equation which models the continuous bending of light rays for the physically-based rendering of participating media with spatially varying index of refraction. Their approach uses photon mapping to accurately simulate continuous refraction and multiple scattering. In the context of volume rendering, only few attempts have been made to incorporate refraction. Li and Mueller [33] used a spline-based filter for the high-quality reconstruction of gradients to improve the appearance of refraction effects and employed an octree to speed up computations [34], but they also focus discretely sampled surface-based objects. Rodgman and Chen [35] presented a refractive volume rendering framework based on discrete ray tracing. They use anisotropic nonlinear diffusion to filter the data in order to reduce noise in the refractive indices. None of these approaches, however, is capable of rendering refractive volumetric scalar fields at interactive frame rates.

Chapter 3

Interactive Volume Rendering

In this section we give an overview of GPU-based direct volume rendering. The method presented in this thesis is a direct volume rendering solution. Volume rendering is a collective noun referring to rendering methods that produce 2D images from 3D volumetric data. Rendering of volumetric data is important in a number of situations such as scientific visualization, medical imaging and in computer graphics to model phenomena not easily represented by surfaces. In the context of scientific visualization, volume rendering methods are often applied to visualize volumetric data gathered from either simulations or experiments. The field of medical imaging field is often concerned with visualizing volumetric data gathered from, for example, CT (computed tomography) scans or MRI (magnetic resonance imagery).

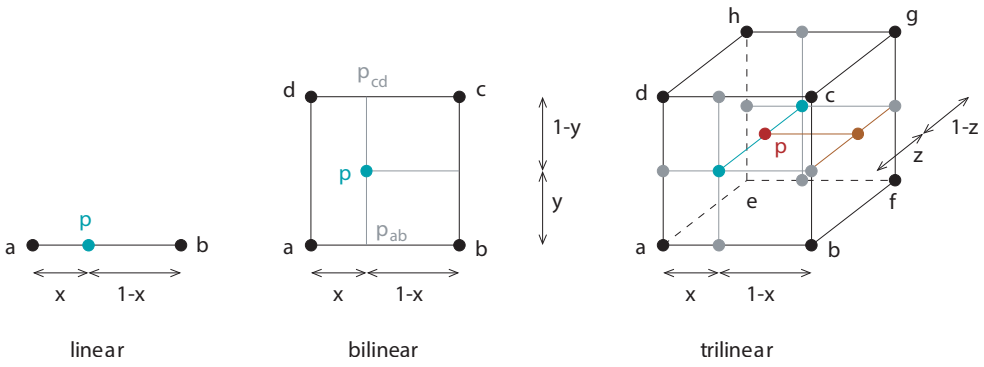


Figure 3.1: Linear (left), bilinear (middle) and trilinear (right) interpolation. Image taken from [36].

3.1 Reconstruction

Volumetric data is generally represented by discretized grids, which is in turn represent continuous 3D scalar fields $f : \mathbb{R}^3 \to \mathbb{R}$, which we refer to as volumes. For the purpose of this thesis we assume that the volumetric data is stored in discrete regular grids, that is the cells of the grid are cuboid. Volumetric data, in the form of discrete regular grids, can be conveniently stored in GPU memory as 3D textures. For the remainder of this thesis, we assume that the volumes are stored in 3D textures. Because the data is discretized, the continuous function must be reconstructed from the discrete grid. There are various reconstruction filters that can be applied in order reconstruct continuous scalar fields from discrete ones. Examples of such filters are nearest neighbour filtering, trilinear interpolation or more accurate, but expensive, filters such as cubic B-spline filtering [36]. Trilinear filtering of textures is supported by graphics hardware, which means that by representing the scalar field as a 3D texture the continuous field can be reconstructed with small performance impacts. The reconstructed volume function can then be accessed by simple texture lookups in shaders. Figure 3.1 illustrates the expansion from linear filtering to bilinear and to trilinear filtering.

3.2 Transfer Functions

In the context of direct volume rendering, the scalar values in the volume are assigned optical properties. This is done by applying a transfer functions to the volume. Transfer functions are typically used to map the sampled values from the volume to RGB colors and alpha, corresponding to a mapping $tf : \mathbb{R} \to \mathbb{R}^4$. Transfer functions are stored in GPU memory as texture lookup tables where the sampled values are used as texture coordinates. Transfer functions can be applied either directly to the elements of the discrete scalar field before reconstruction (pre-interpolative), or they can be applied to the reconstructed values sampled from the volume (post-interpolative) [36]. In Figure 3.2 we can see the effects of applying the transfer function before and after reconstruction. The post-interpolative method yields the better results.

3.3 Volume Rendering Integral

In general, the mathematical formulation of light transport in a participating medium is given by the radiative transport equation which describes the total radiance $L(x, \omega)$ at a position x in direction ω [4]:

$$L(x, \omega) = T(x_0, x) \cdot L_0(x_0, \omega) + \int_{x_0}^{x} T(x', x) \cdot \sigma(x') \cdot L_s(x', \omega)\, dx', \qquad (3.1)$$

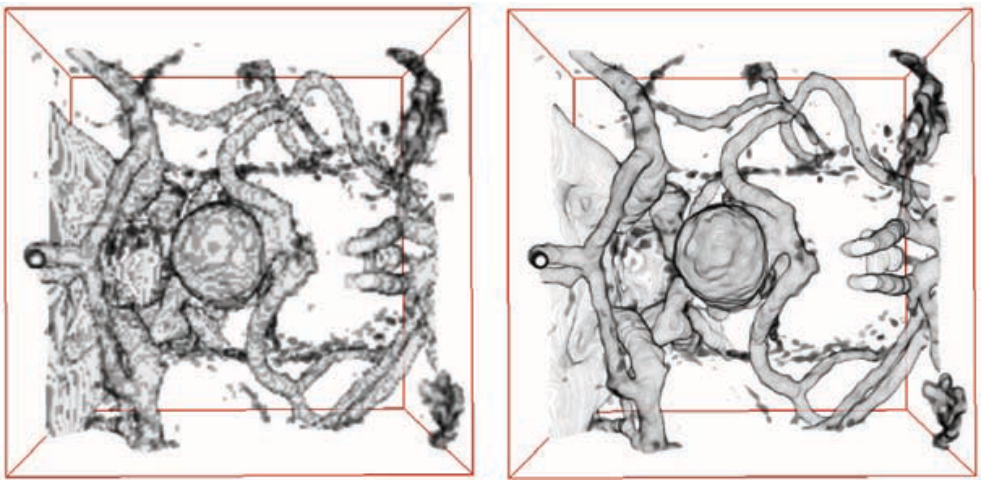


Figure 3.2: Pre-interpolative (left) and post-interpolative (right) transfer function application. Applying the transfer function before reconstruction causes detrimental artifacts. Image taken from [36].

where $L_0(x_0, \omega)$ is the background radiance from a boundary position x_0 and $\sigma(x)$ is the extinction coefficient at a position x. The transmittance $T(x_i, x_j)$ between any two points x_i and x_j is:

$$T(x_i, x_j) = e^{-\tau(x_i, x_j)}, \tag{3.2}$$

where $\tau(x_i, x_j)$ is the optical depth defined as:

$$\tau(x_i, x_j) = \int_{x_i}^{x_j} \sigma(x')\, dx'. \tag{3.3}$$

$L_s(x, \omega)$ corresponds to the amount of radiance arriving from all directions at a point x in the direction ω:

$$L_s(x, \omega) = \int_{\Omega} P(x, \omega', \omega) \cdot T(x_b, x) \cdot L_0(x_0, \omega')\, d\omega', \tag{3.4}$$

where Ω is the sphere of all directions. Scattering probability is described by the phase function $P(x, \omega', \omega)$, which is the probability density of radiance being scattered from an incident direction ω' to direction ω.

3.4 Emission-Absorption Integral

In practice, the costly integration over the sphere is often avoided and replaced by simpler models. The most commonly employed model is the emission-absorption model. In this model scattering is ignored. Instead of the scattering

term $L_s(x', \omega)$, we use only the emissive term $L_e(x')$, which is independent of direction. The background radiance $L_0(x_0, x)$ is usually assumed to be 0 for volume visualization leaving us with the following integral[37]:

$$L(x) = \int_{x_0}^{x} T(x', x) \cdot \sigma(x') \cdot L_e(x')\, dx' \tag{3.5}$$

This integral can be discretely approximated with a Riemann sum:

$$L(x) \approx \sum_{i=1}^{n} T(x_i, x) \cdot \underbrace{\sigma(x_i) \cdot L_e(x_i)\, \Delta x_i}_{c(x_i)} \tag{3.6}$$

where $c(x_i)$ denotes the radiance at x_i. Later, when compositing is discussed in Section 3.6, we refer to this radiance as the premultiplied color, that is the color at x_i multiplied with the associated opacity. Substituting in c_i (for convenience) and the definition of $T(x_i, x)$ we can also approximate the transmittance term:

$$\begin{aligned} L(x) &\approx \sum_{i=1}^{n} c(x_i) \cdot \exp\left(- \int_{x_i}^{x} \sigma(x')\, dx' \right) \\ &\approx \sum_{i=1}^{n} c(x_i) \cdot \exp\left(- \sum_{j=1}^{i-1} \sigma(x_j)\, \Delta x_j \right) \\ &= \sum_{i=1}^{n} c(x_i) \cdot \prod_{j=1}^{i-1} \exp\left(-\sigma(x_j)\, \Delta x_j\right) \\ &= \sum_{i=1}^{n} c(x_i) \cdot \prod_{j=1}^{i-1} \left(1 - \alpha(x_j)\right) \end{aligned} \tag{3.7}$$

with opacity α_j at position x_j defined as:

$$\alpha(x_j) = 1 - \exp(-\sigma(x_j)\Delta x_j) \tag{3.8}$$

Generally the premultiplied color $c(x_i)$ and opacity $\alpha(x_i)$ are retrieved by applying a transfer function to the scalar values associated with position x_i.

3.5 Volume Rendering Algorithms

In this section we will go through the two volume rendering algorithms most related to the method presented in this thesis, namely raycasting and view-aligned slicing.

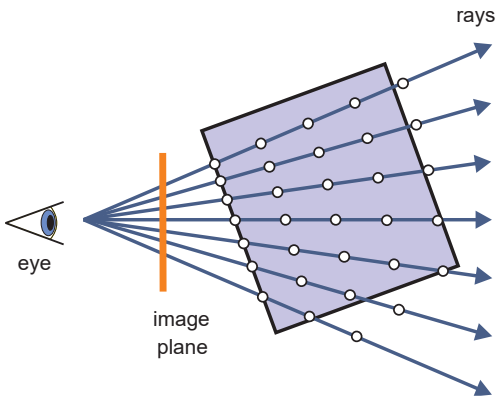


Figure 3.3: In the raycasting method rays are traversed from the eye along rays through the volume. Image taken from [36].

Raycasting

Raycasting is an important technique that many volume renders and volume rendering techniques build upon. Raycasting aims to evaluate the volume rendering integral directly by sampling the volume at discrete intervals along rays leaving the camera in a manner that essentially amounts to the Riemann sums we discussed in the previous section [36]. Each pixel in a rendered image represent a single ray (except when supersampling is used) leaving the camera as illustrated in Figure 3.3. To render an image by raycasting in a front-to-back manner, we start by initializing the pixels to a fully transparent color. After initialization we start sampling the volume along the ray associated with this pixel. At each sample along the ray, we apply the transfer function to the sampled value to retrieve the optical properties, such as the color, associated with this value. The color is then composited under the current color in the pixel as we will discuss in Section 3.6.

View-aligned Slicing

Another popular technique in volume rendering is view-aligned slicing [36]. The idea of this technique is to transverse the volume in a plane-by-plane manner, outputting the optical properties of the volume at the positions of the current plane and compositing them in the final output image. The slices are aligned with the view plane as seen in Figure 3.4. The planes can be represented as planar geometry that are swept through the volume. The slices can be processed in a front-to-back or back-to-front order. When using front-to-back traversal, the evaluated slice is composited under the previously process slices, and in the case of back-to-front it is composited over, using the under and over operators [38] as discussed in the next section.

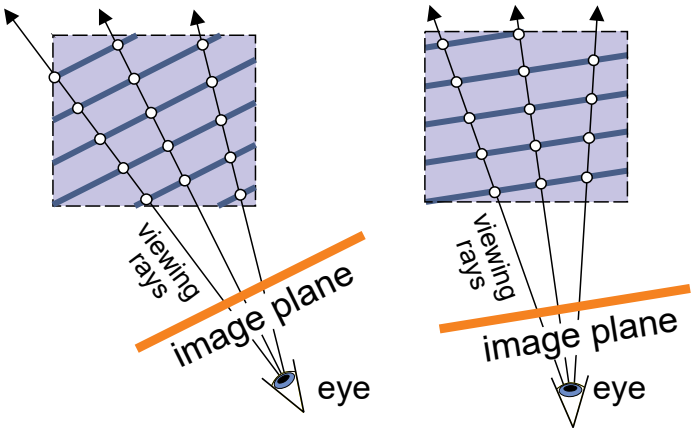


Figure 3.4: In the view-aligned slicing method the volume is processed in a plane-by-plane manner. Image taken from [36].

3.6 Compositing

Compositing is the mechanism used when iteratively evaluating the volume rendering integral, for example as done in raycasting. If we look back to the discretized volume rendering integral for emission and absorption, we can reformulate it to support recursive evaluation.

$$L(x) \approx \sum_{i=1}^{n} \underbrace{c(x_i)}_{c_i} \cdot \prod_{j=1}^{i-1} \underbrace{(1 - \alpha(x_j))}_{\alpha_i} \tag{3.9}$$

Here we define c_i as the premultiplied color at the current position x_i while α_i denotes the opacity. If we evaluate the integral in a front-to-back scheme, compositing is done with an *under* operation. The equations for this operation are:

$$\begin{aligned} C_i &= C_{i-1} + (1 - A_{i-1}) \cdot c_i \\ A_i &= A_{i-1} + (1 - A_{i-1}) \cdot \alpha_i \end{aligned} \tag{3.10}$$

where C_i and A_i are the accumulated color and opacity for index i. If the integral is evaluated in a back-to-front scheme the composition must be done with an *over* operation. The equations for this are:

$$\begin{aligned} C_i &= c_i + (1 - \alpha_i) \cdot C_{i-1} \\ A_i &= \alpha_i + (1 - \alpha_i) \cdot A_{i-1} \end{aligned} \tag{3.11}$$

3.7 Opacity Correction

Usually the transfer functions are specified in relation to a fixed sample distance. Our previous definition of opacity,

$$\alpha_i = \alpha(x_i) = 1 - \underbrace{\exp(-\sigma(x_i)\Delta x_i)}_{1-\alpha_i}, \tag{3.12}$$

assumes that the opacity is specified for a sample distance of Δx_i. If we want to change the sample rate the opacity must be corrected if the output is to appear similar. There are several reasons one might want to change the sample rates. Increasing the sample rate can for example be done in order to produce an image of higher quality, decreasing the sample rate can be done to increase performance. If we increase the sample rate without correcting the opacity, regions in the volume would end up appearing more opaque than they should. Opacity for a sample distance of $\Delta x_i'$, based on an opacity specified for a sample distance of Δx_i, can be calculated using the following equation [39]:

$$\begin{aligned}
\alpha_i' &= 1 - \exp\left(-\sigma(x_i)\Delta x_i'\right) \\
&= 1 - \exp\left(-\sigma(x_i)\Delta x_i \frac{\Delta x_i'}{\Delta x_i}\right) \\
&= 1 - \exp\left(-\sigma(x_i)\Delta x_i\right)^{\frac{\Delta x_i'}{\Delta x_i}} \\
&= 1 - (1-\alpha_i)^{\frac{\Delta x_i'}{\Delta x_i}}
\end{aligned} \tag{3.13}$$

The premultiplied color must also be corrected since it has been multiplied with the uncorrected opacity.

$$c_i' = c_i \frac{\alpha_i'}{\alpha_i} \tag{3.14}$$

3.8 Gradients and Lighting

Lighting is crucial for how humans perceive 3D objects. Without lighting discerning the shapes or curvatures of objects can be very difficult. Most illumination models commonly used in computer graphics compute lighting for surfaces based on the surface normals. In our case however we don't have defined surfaces but we want to apply the same illumination models. The typical solution to this issue is to use the gradient of the scalar field as the normal when shading points in the volume along rays. The gradient does in fact give us the normal of the isosurface passing through the point associated with the value of that point [36]. The gradient of a scalar field f is defined as:

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial z} \end{pmatrix} \tag{3.15}$$

Since we usually don't have the gradient of the volume as part of the data, we need to calculate an estimate. The simplest and most commonly used method of estimating the gradient is by calculating finite differences like the central difference:

$$\nabla f(x,y,z) \approx \frac{1}{2h}\begin{pmatrix} f(x+h,y,z) - f(x-h,y,z) \\ f(x,y+h,z) - f(x,y-h,z) \\ f(x,y,z+h) - f(x,y,z-h) \end{pmatrix} \tag{3.16}$$

where h denotes the distance of the samples. Using the gradient as the normal one can employ illumination models such as Lambertian reflectance which models diffuse reflection. The Lambertian reflectance states that the light intensity as reflected of a surface is proportional to the cosine of the angle between the light direction and surface normal.

3.9 Preintegration

In order to achieve interactive frame rates in the context of volume rendering, the number of volume samples that can be evaluated per frame is limited. The lower sample rates often chosen for volume rendering will cause artifacts. To remedy these artifacts, preintegration can be employed. In Figure 3.5 we can see these artifacts and the results of using preintegration. When preintegration is employed, evaluation of the volume rendering integral is done using the trapezoid rule of numerical integration rather than with a Riemann sum. Instead of sampling the transfer functions at only the currently evaluated scalar value, we can consider the integral between the current value and the previous value. By assuming a linear relationship between the two we can precalculate an approximation of the integral between all pairs of values. These precalculated transfer function colors are then stored in 2D or 3D lookup tables. Figure 3.6 shows two examples of images rendered using preintegration along with the lookup tables used for the retrieval of transfer function colors. We denote the alpha between points x_i and x_j by:

$$\begin{aligned} \alpha(x_i,x_j) &= 1 - \exp\left(-\int_{x_i}^{x_j} \sigma(x')\,dx'\right) \\ &\approx 1 - \exp\left(-\Delta x \int_0^1 \sigma(\underbrace{(1-\lambda)f(x_i) + \lambda f(x_j)}_{\Lambda(x_i,x_j,\lambda)})\,d\lambda\right) \\ &= \underbrace{1 - \exp\left(-\Delta x \int_0^1 \sigma(\Lambda(x_i,x_j,\lambda))\,d\lambda\right)}_{1-\alpha(x_i,x_j)} \end{aligned} \tag{3.17}$$

$\alpha(x_i,x_j)$ is thus a function dependent on x_i, x_j and Δx, or more specifically the scalar values from the volume function at position x_i and x_j and the

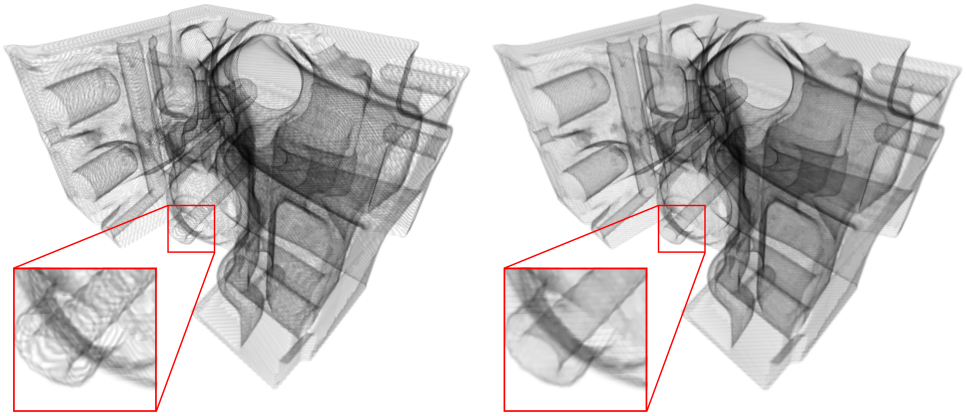


Figure 3.5: Volume rendering of an engine without preintegration on the left, and with on the right.

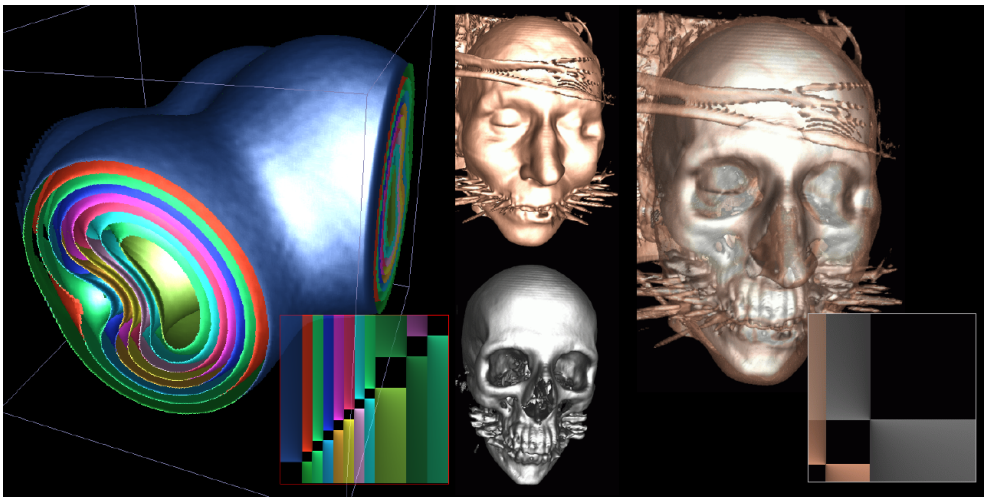


Figure 3.6: Isosurfaces rendered for two datasets with their corresponding preintegration lookup tables. Image taken from [40].

sample rate Δx [39, 40]. We approximate the premultiplied color in a similar fashion:

$$\begin{aligned} c(x_i, x_j) \approx & \Delta x \int_0^1 c\left(\Lambda(x_i, x_j, \lambda')\right) \\ & \cdot \exp\left(-\Delta x \int_0^\lambda \sigma(\Lambda(x_i, x_j, \lambda'))\, d\lambda'\right) d\lambda \end{aligned} \tag{3.18}$$

Like $\alpha_{i,j}$, $c_{i,j}$ is a function dependent on the scalar values at x_i, x_j and the sample rate Δx. Alternatively, one can assume the sample rate Δx to be

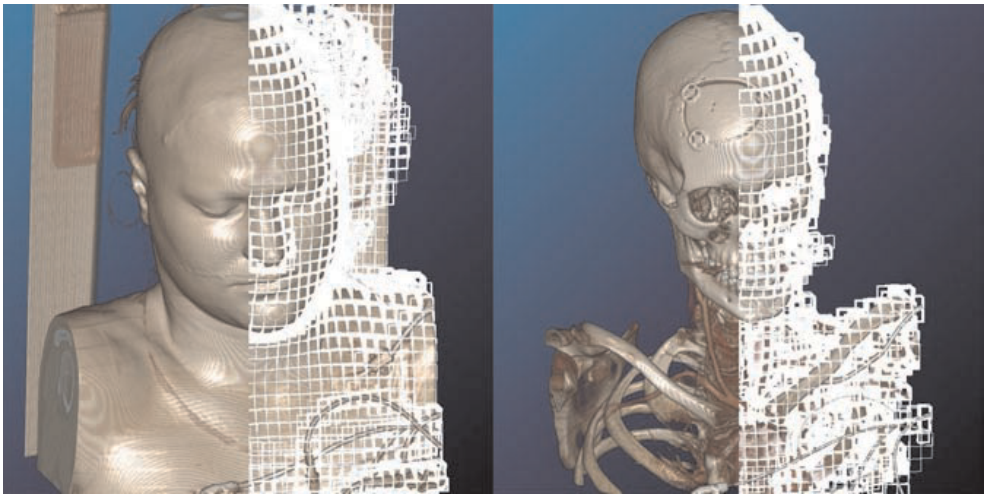


Figure 3.7: Visible regions are shown as white cubes. The volume is divided in uniformly sized blocks. In the left image roughly 40% of the fragments where skipped. In the right image around 80% of the fragments where skipped [36]. Image taken from [36].

constant and correct the opacity after the preintegration. The benefit of doing this is that the lookup texture can be 2D instead of 3D. The opacity corrected version of $\alpha'_{i,j}$ is then:

$$\alpha(x_i, x_j)' = 1 - (1 - \alpha(x_i, x_j))^{\frac{\Delta x'}{\Delta x}}, \tag{3.19}$$

using the same logic as in section 3.7. These functions are approximated for all combinations of values in the preintegraion table, typically in the range $[0, 1]$. They can be solved either analytically as the integrals are now over linear functions, or numerically using Riemann sums.

3.10 Empty Space Skipping

Extensive regions of volumes might not contribute to the final rendering [36]. Consider the case where all points in a region map to fully transparent colors via the transfer function, the region will be fully invisible. Performance can be gained by skipping these regions during rendering. The conventional approach for this is to partition the regions of the volume into visible and invisible regions. The regions can be partitioned, for example, by constructing an octree around the visible regions of the volume, or dividing it into uniformly sized blocks thus dividing the volume based on visibility [41]. In the context of raycasting, the empty space can be skipped by starting the ray at the nearest visible region. In Figure 3.7 we can see such a partitioning. The empty regions

of the volume are invisible, whilst the regions contributing the the image are marked as white-outlined cubes.

Chapter 4

Interactive Volume Refraction

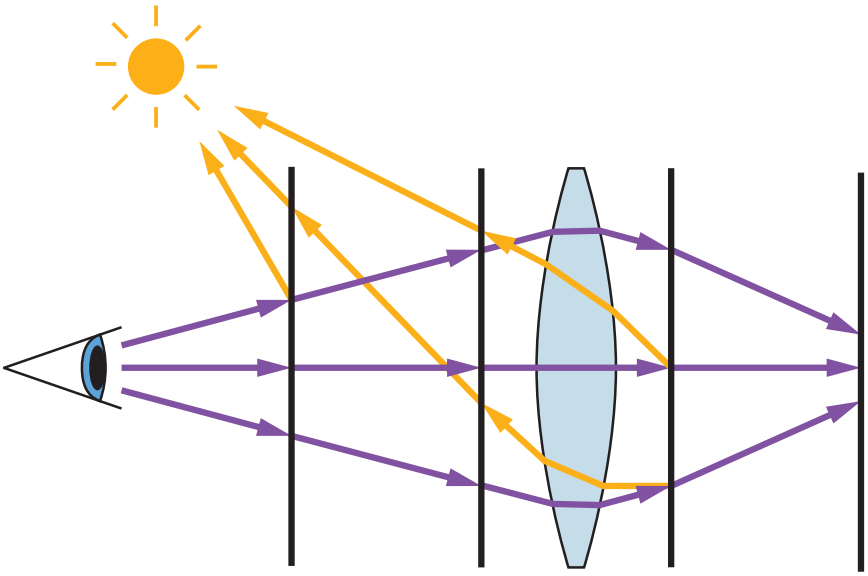


Figure 4.1: Viewing rays (—) are forward integrated while light rays (—) are backward integrated. The paths of the light rays are reconstructed by looking back to the previous illumination plane.

The goal of our method is to render volumetric refractive participating media with illumination at interactive frame rates while avoiding precomputation. In particular, we want to avoid the explicit generation of an illumination volume as, in the context of refraction, such a discretization is problematic since the curved light rays greatly complicate sampling on a regular grid. For non-interactive applications, techniques such as volumetric photon mapping [2, 32] have been employed to render scalar fields with refractive media. These techniques are, however, prohibitively expensive for interactive applications as the amount of photons needed for the result to appear continuous is too

large. Rather than shooting individual photons, our technique propagates light in a plane-by-plane manner while simultaneously advancing viewing rays as illustrated in Figure 4.1. The volume is traversed in planes parallel to the image plane and, in the following description, we assume a single directional light source. Before discussing the details of light and viewing ray propagation in Sections 4.2, 4.3, and 4.4, we will first briefly outline the foundations of our model in Section 4.1.

4.1 Model

We assume a continuous scalar-valued volumetric function $f : \mathbb{R}^3 \to \mathbb{R}$ as well as an additional field $\eta : \mathbb{R}^3 \to \mathbb{R}$ that defines the refractive index at every point. Based on the ray equation of geometric optics [28], we can then describe the path of a light ray in this field as:

$$\frac{d}{ds}\Big(\eta\frac{dx}{ds}\Big) = \nabla\eta \tag{4.1}$$

where ds denotes an infinitesimal step in the direction tangential to the curved ray. We adopt the discretized version of this equation described by Ihrke et al. [29] and later used by Sun et al. [30]. They define the ray direction as $v = \eta\frac{dx}{ds}$ and rewrite the equation as a system of first-order differential equations:

$$\frac{dx}{ds} = \frac{v}{\eta}, \qquad \frac{dv}{ds} = \nabla\eta, \tag{4.2}$$

which can be discretized as:

$$x_{i+1} = x_i + \frac{\Delta s}{\eta}v_i, \qquad v_{i+1} = v_i + \Delta s\nabla\eta, \tag{4.3}$$

where x_i and x_{i+1} correspond to the previous and new position, respectively, along the ray, v_i and v_{i+1} are its previous and new directions, and Δs is the step size. With this we can model the changes in ray direction caused by refraction, as $\nabla\eta$ can also be discretized using finite differences. In practice, it is usually convenient to specify η as a function of the scalar value (and/or other attributes), as is commonly done for other optical properties such as color and opacity. We therefore, in addition to the color transfer function c_{tf} and the opacity transfer function α_{tf}, provide a refraction transfer function η_{tf}. Volume rendering commonly uses the particle model of Porter and Duff [38]. However, when introducing refraction it is convenient to also provide explicit control over the color of the transmissive medium. We therefore additionally introduce a medium color transfer function m_{tf}.

Following the physically-based color model by Oddy and Williams [42], we can interpret the values provided by these functions as pigment particles suspended

in a filter-like medium. The color of the pigments is specified by c_{tf} while the color of the medium is given by m_{tf}. Intuitively, the value of α_{tf} controls the proportion of medium vs. particles, i.e., $\alpha_{tf} = 0$ means that no reflective particles are present, while $\alpha_{tf} = 1$ corresponds to densely packed particles that do not permit transmittance. If the medium color is constant white, this exactly corresponds to the standard Porter-Duff model. For varying medium colors, however, it implies that in this model it is possible to have a visible contribution of the volume to the final image even if α_{tf} is constant zero. An example of this would be (idealized) tinted glass which only exhibits filtering but not reflective behavior.

4.2 Light Propagation

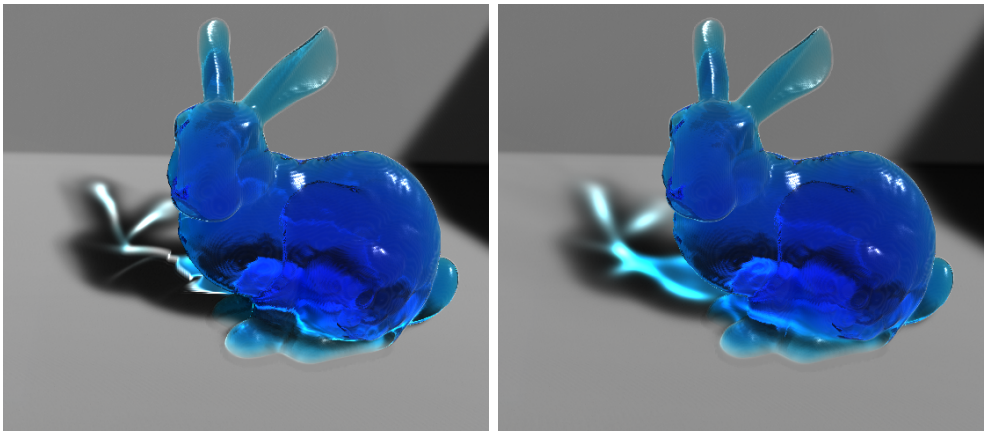
As discussed in the previous section, we can discretize light paths in a refracting medium by forward-integrating them using the gradient of the refractive index field. The major disadvantage of this approach is that it makes it difficult to efficiently store illumination information, which is needed during viewing ray traversal. We draw inspiration from the field of texture-based flow visualization where a similar issue occurs. When forward-advecting a texture in an unsteady vector field in a Lagrangian manner, holes may appear and a dense coverage of the domain is difficult to maintain. A solution is to use a hybrid Lagrangian-Eulerian scheme [43], where backward integration on a regular grid is employed. Given the similarity between the two scenarios, we adopt an analogous approach for light propagation in refractive media.

Light is propagated in a plane-by-plane manner, storing its direction and radiance in 2D buffers. For every point in the light buffer we backward-integrate the light direction and intersect it with the previous light plane. This gives us the incoming radiance, which is then attenuated and filtered based on the particle and medium contributions between the two planes. The gradient of the refractive index field $\nabla\eta$ is then used to update the light ray direction. More formally, for every pixel position on the current light buffer L_i, we compute:

$$L_i = L_{i-1} I_i (1 - \alpha) m, \tag{4.4}$$

where L_i denotes the light color on light plane i and I_i is its intensity (see the discussion on intensity correction below). L_0 is initialized with the color of the light source. Light is attenuated by the opacity, α, and filtered by the medium color m, of the particles between the planes $i - 1$ and i. We use preintegration tables based on the corresponding transfer functions, which are updated whenever they are changed. The light direction is updated using the ray equation of geometric optics discussed earlier:

$$ld_i = ld_{i-1} + \Delta s \nabla\eta. \tag{4.5}$$



(a) (b)

Figure 4.2: Effects of light filtering. (a) No filtering. (b) Filtering of light and light direction.

The ray direction changes in the direction of the gradient of the field of indices of refraction and ld_0 is the direction of the light prior to any refraction events. As with L_{i-1}, ld_{i-1} is obtained by intersecting back to the previous light plane in the direction of the light.

Light Filtering

This backward integration scheme as-is models a directional light source. However, following the approach of Patel et al. [17], we can use incremental convolution to efficiently support distant area light sources with controllable softness with little additional costs. The intuition of this is that by applying the blurring kernel to the previous light plane at every iteration, earlier light events, like shadowing, will become increasingly diffuse as light progresses. Their elliptical convolution kernel which uses a randomized rotational offset can be straightforwardly employed instead of a simple texture lookup, and requires only two additional texture fetches (for the three-sample kernel, which is used throughout the thesis). Additionally, we use the same kernel to filter the ray directions to remedy artifacts caused by the backward mapping approach. The artifacts are caused by the fact that refracted rays tend to move toward the refractive media. The backward mapping will have a bias towards light near the boundary of the media, where light is dispersed, because the backward ray direction will tend to point into the direction of the boundary. The filtering counteracts this bias to some degree. The filtered versions of light color and direction are then simply used instead of L_{i-1} and ld_{i-1} in Equations 4.4 and 4.5. An example of the effect of applying this filtering is shown in Figure 4.2.

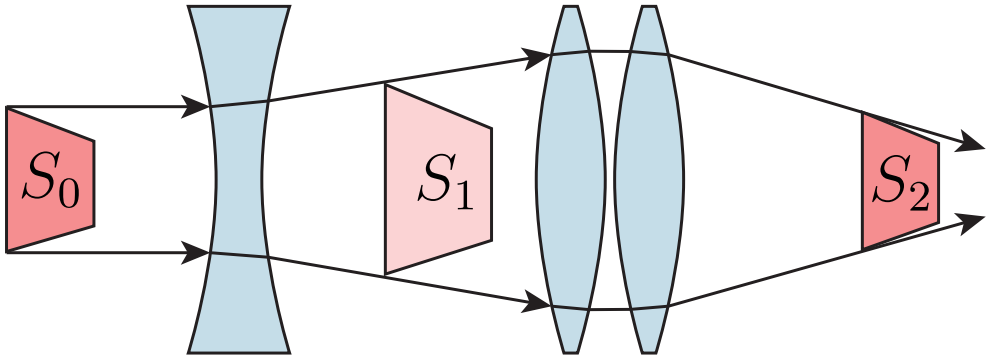


Figure 4.3: Light intensity is dimmed from S_0 to S_1, and brightened again from S_1 to S_2.

Intensity Correction

Caustics are caused by the concentrations and dispersal of light and represent an important aspect of light transfer in refractive media. When used naively, the backward mapping approach lets us propagate and diffuse light, but does not capture caustics. The intensity law of geometric optics states that the energy within an infinitesimal stream tube formed from rays leaving a wavefront element of size dS_1 and hitting one of size dS_2 is constant [44]:

$$I_1 dS_1 = I_2 dS_2, \tag{4.6}$$

where I_1 denotes the light intensity on dS_1 and I_2 the intensity on dS_2. In our approach, we use a discretized version of the intensity law:

$$I_i = \frac{I_{i-1} S_{i-1}}{S_i}, \tag{4.7}$$

where I_i is then the intensity of the light on an element with an area of S_i originating from an element with an area of S_{i-1} and intensity I_{i-1}. Figure 4.3 illustrates the intensity law. Light leaving S_0 is dimmed as it travels to S_1, and is again brightened as it travels from S_1 to S_2.

In principle, we could compute the required areas by backward-integrating the vertices of a regular shape. However, a more efficient solution is to approximate them using screen-space partial derivatives of the intersection between the light ray and the current and previous line planes as modern GPUs offer built-in functions to compute these derivatives. The areas S_1 and S_2 in Equation 4.7 are then given by:

$$S_i \approx \left| \frac{d}{dx} p_i \right| \left| \frac{d}{dy} p_i \right|, \tag{4.8}$$

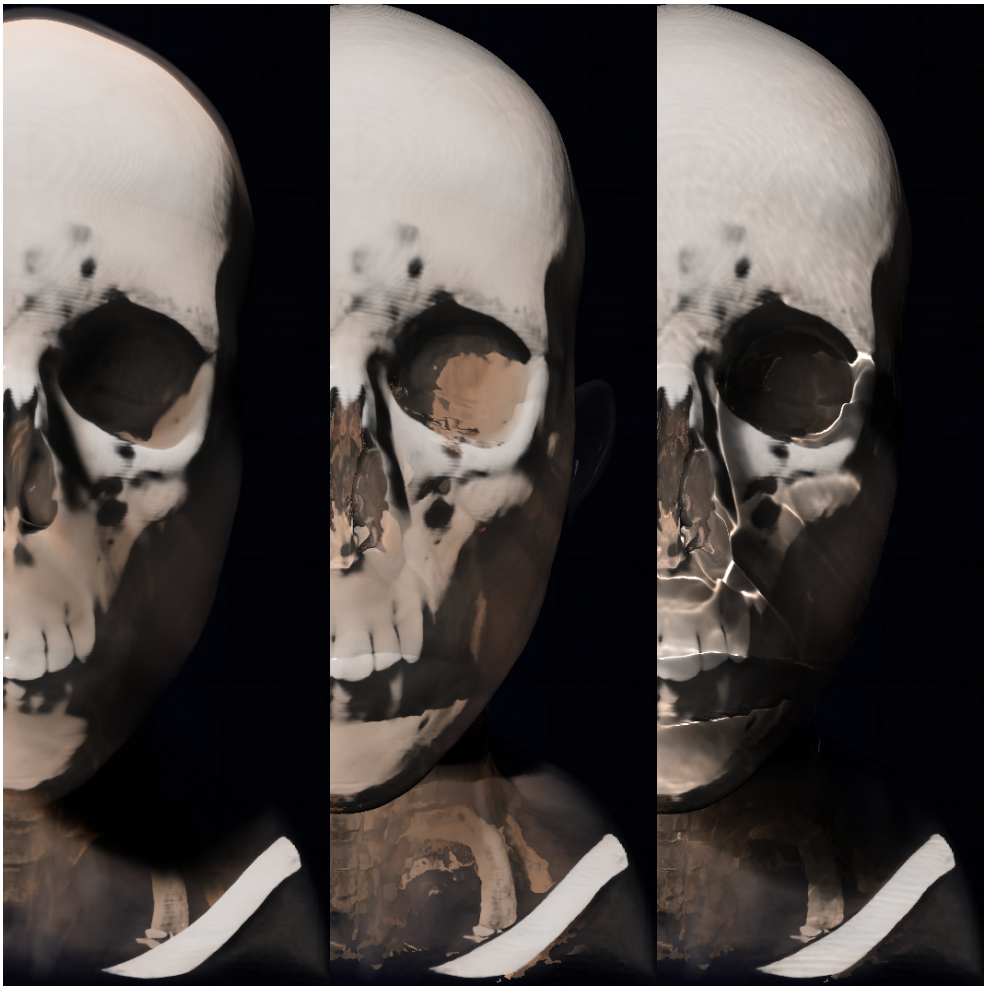


Figure 4.4: Effects of light intensity correction. Left: no refraction. Middle: no intensity correction. Right: intensity correction recovers caustics.

where p_i is the intersection point of the light ray and the corresponding plane. This approximation is then used in Equation 4.7 to calculate the caustics.

Figure 4.4 shows the importance of caustics for proper shadowing of refracted light. The shadows disappear in the middle image because the light is bent around the chin, but because caustics are not considered the light intensity in the created gap remains the same. When caustics are considered the shadow returns because the dispersal of the light causes its intensity be to lowered.

4.3 Viewing Ray Propagation

In contrast to light propagation, viewing rays are advanced using regular forward integration, i.e., we store their position, direction, and accumulated color in a set of 2D buffers. To enable the distinction between reflective and filtering behavior as outlined in Section 4.1, we also need an additional intermediate buffer for the medium color. Each pixel in these buffers corresponds to one viewing ray. In order to propagate the viewing rays together with the light, we intersect the viewing rays with the current illumination plane.

Shading

The incoming diffuse illumination is accessed by looking up the value in the light buffer at the current location of the viewing ray. We also calculate a specular component analytically using a Cook-Torrance BRDF [45] with GGX [23]. This calculation is done based on the direction of the light, the viewing ray, and the normal. We use the gradient of the scalar field as the normal, and calculate the reflectivity of the interface based on the relative index of refraction between the current and previous illumination planes. The Cook-Torrance BRDF specular reflectance model defines the specular reflection to be:

$$R_s = \frac{D(h)F(v,h)G(l,v,h)}{4(n \cdot l)(n \cdot v)} \tag{4.9}$$

where D denotes the facet slope distribution function, G the geometric attenuation function and F the Fresnel term. The v, l and h terms denote the view, light, and halfway vectors, respectively. There are multiple choices that could be used for the D, G, and F terms. Karis investigated various combinations of terms when working on the shading model for Unreal Engine 4 (UE4) [46]. We adopt the same D and G terms as chosen for UE4, namely Trowbridge-Reitz GGX [47] for the D term and a modified version of Schlick's model [48] for the G term. The Trowbridge-Reitz GGX D term is given by:

$$D(h) = \frac{\alpha^2}{\pi((n \cdot h)^2(\alpha^2 - 1) + 1)^2} \tag{4.10}$$

Note that for these equations we use the α from the cited papers denoting roughness and not opacity. We, like Karis, also adopt Disney's remapping of $\alpha = \text{roughness}^2$ [49]. The modified Schlick model for G is defined as:

$$\begin{aligned} G(l,v,h) &= G_1(l)G_1(v) \\ G_1(v) &= \frac{n \cdot v}{(n \cdot v)(1-k)+k} \\ k &= \frac{(\text{roughness}+1)^2}{8} \end{aligned} \tag{4.11}$$

For the F term we use the one described by Cook and Torrance [23, 45]:

$$\begin{aligned} F(v,h) &= \frac{1}{2}\frac{(g-c)^2}{(g+2)^2}\left(1+\frac{c(g+c)-1}{c(g-c)+1}\right) \\ \text{where } g &= \sqrt{\frac{\eta_t^2}{\eta_i^2}-1+c^2} \text{ and } c = v \cdot h \end{aligned} \tag{4.12}$$

Here η_t denotes the index of refraction on the transmitted side, and η_i the index of refraction on the incident side. We chose this Fresnel term because its dependent on the incident and transmitted indices of refraction rather than some term dependent on the refractive index relative to air, making it more convenient when used with scalar fields.

Compositing

Given the fact that we explicitly handle reflective and transmissive behavior, the compositing step differs from standard volume rendering. Substituting into the equations by Oddy and Willis [42, Sections 4.2 and 4.3], we obtain:

$$\begin{aligned} C_i &= C_{i-1} + (1-A_{i-1}) \cdot M_{i-1} \cdot (\alpha \cdot c \cdot i_d + i_s) \\ A_i &= A_{i-1} + (1-A_{i-1}) \cdot \alpha \\ M_i &= M_{i-1} \cdot m \end{aligned} \tag{4.13}$$

where C, A, and M, correspond to the previous (index $i-1$) and new (index i) values of particle color, opacity, and medium color, c, α, and m denote the contributions of the ray segment, and i_d and i_s correspond to the diffuse and specular illumination contributions. This equation has several differences from standard compositing in volume rendering. First, we note that in the original model by Oddy and Willis [42] the color was multiplied by the medium color M_{i-1} twice because they model the light as passing through the medium and being filtered once before getting reflected of the particles and passing through the medium again on the way back to the viewer. In our model, the light used for the compositing has already been filtered by the medium, so we disregard the second multiplication. Also, the specular lighting contribution is not multiplied by the opacity, meaning that purely transmissive parts of the volume without opacity contribution may still exhibit specular reflections, as is desired in the case of materials such as glass.

We use preintegration in our solution to improve the quality of the output images without increasing the sample rate. Oddy and Willis' model must therefore be adapted to support preintegration. We approximate the integral between two points by assuming a linear relationship and storing the preintegrated values in a lookup table. Note that the compositing of the medium

color is fully multiplicative, we therefore utilize the multiplicative integral of multiplicative calculus [50, 51] given by:

$$\prod_a^b f(x)^{dx} = \lim_{\Delta x_i \to 0} \prod_{i=0}^{n-1} f(\xi_i)^{\Delta x_i} = \exp\left(\int_a^b ln(f(x))\,dx\right) \qquad (4.14)$$
$$\text{where } \xi_i \in [x_{i-1}, x_i] \text{ and } x_0 = a, x_{n-1} = b$$

The alpha compositing of Oddy and Willis' model is identical with the standard alpha compositing, we can therefore simply use the approximation discussed in Section 3.9:

$$\begin{aligned}
\alpha(x_i, x_j) &= 1 - \exp\left(-\int_{x_i}^{x_j} \sigma(x')\,dx'\right) \\
&\approx 1 - \exp\left(-\Delta x \int_0^1 \sigma(\underbrace{(1-\lambda)f(x_i) + \lambda f(x_j)}_{\Lambda(x_i,x_j,\lambda)})\,d\lambda\right) \\
&= 1 - \underbrace{\exp\left(-\Delta x \int_0^1 \sigma(\Lambda(x_i, x_j, \lambda))\,d\lambda\right)}_{1-\alpha(x_i,x_j)}
\end{aligned} \qquad \text{(3.17 revisited)}$$

We express the medium color between two points using a multiplicative integral:

$$\begin{aligned}
m(x_i, x_j) &= \prod_{x_i}^{x_j} m_{tf}(f(x'))^{dx'} \\
&\approx \left(\prod_0^1 m_{tf}(\Lambda(x_i, x_j, \lambda))^{d\lambda}\right)^{\Delta x}
\end{aligned} \qquad (4.15)$$

Note that the medium color can be opacity corrected for smaller sample distances by rising it to the power of $\Delta x'$. The color compositing is identical to the standard one, as discussed in Section 3.9, with the exception that the medium color between the two points must be taken into account. The color is multiplied by the medium color term:

$$\begin{aligned}
c(x_i, x_y) \approx \Delta x \int_0^1 & c(\Lambda(x_i, x_j, \lambda)) \\
& \cdot \exp\left(-\Delta x \int_0^\lambda \sigma(\Lambda(x_i, x_j, \lambda'))\,d\lambda'\right) \\
& \cdot \left(\prod_0^\lambda m_{tf}(\Lambda(x_i, x_j, \lambda'))^{d\lambda'}\right)^{\Delta x} d\lambda
\end{aligned} \qquad (4.16)$$

We approximate and store these equations for a fixed sample distance Δx for all combinations of values in a 256 by 256 lookup table using a compute shader. In practice it is easiest to use the compositing equations with opacity corrections directly to compute the preintegration table. For each pair of values we take 256 samples interpolated between the two, compositing each sample in the final value and opacity correcting them for a sample distance of $\Delta x' = 1/256$. Because we assume a fixed sample distance, the preintegrated values must also be opacity corrected should the sample distance not be Δx.

4.4 Environment Mapping

In contrast to common volume rendering models, where transparent structures are represented by reflecting matter, our approach can also generate visible contributions without the presence of opaque material along a viewing ray due to changes is the ray direction caused by refraction. The refracted rays will distort the background, making the shape of the distorting media more apparent. However, this effect will not be visible on a solid background. For this reason, after ray traversal has finished, we apply an optional environment mapping step where the final viewing ray directions are used to retrieve the background color from a texture that is then blended with the volume rendering based on medium and particle color and opacity. For simplicity, we employ a simple equiangular environment map, but other approaches such as a cube maps could be used as well. As neighboring rays that have undergone refraction may exhibit large differences in direction, we use summed area tables to anti-alias the environment map as to avoid noisy results. In Figure 4.5 we can see this aliasing, even with mipmapping, and that by using summed area tables, the noise is greatly reduced.

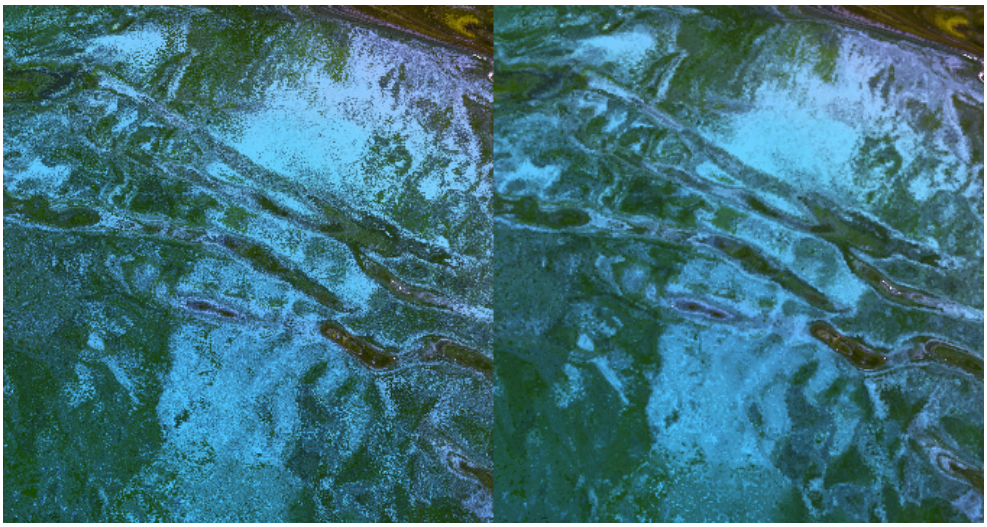


Figure 4.5: Distorted environment map behind a refractive medium with mipmapping anti-aliasing (left) and summed area table anti-aliasing (right).

Chapter 5

Implementation

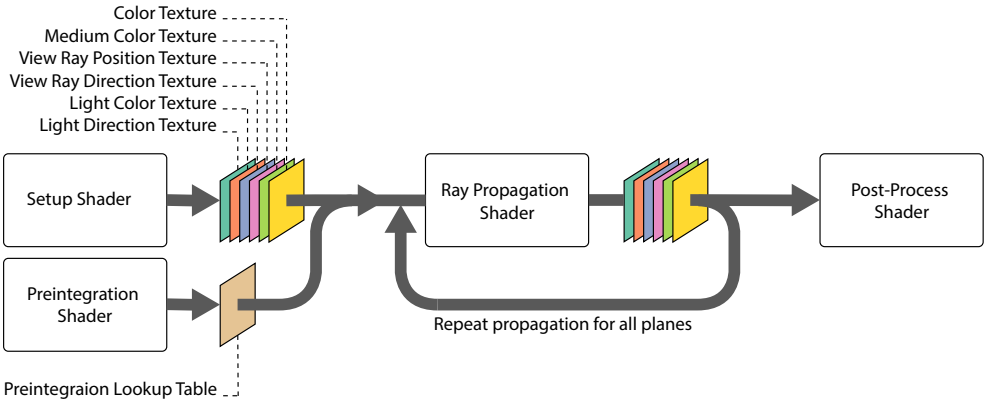


Figure 5.1: Overview of our algorithm. Initially the preintegration table is recomputed using a compute shader if the transfer function has changed. The setup compute shader clears out and initializes the buffers. In the ray propagation step, viewing rays and light are propagated front-to-back between all illumination planes. Finally, an environment map is composited behind the model, using the color, medium and ray direction buffers, and the final image is rendered to the screen.

The presented method was implemented using OpenGL 4.5. All the major steps of our algorithm are executed on the GPU. Figure 5.1 shows an overview of the major steps in our algorithm. All buffers are stored in layered 2D textures with two layers each. These layers are written to and read from in a ping-pong fashion. Modern OpenGL allows for a cheap method of ping-ponging using multi-layer framebuffer attachments. By writing the gl_Layer built-in variable, a geometry shader can specify which layer of the attachment is written. Synchronization is realized with glTextureBarrier, which assures that texture writes have been completed and that texture caches are invalidated. Our algorithm uses 6 layered textures: the light color texture, the

light direction texture, the view ray position and direction textures, and the accumulated color and medium texture. These textures are initialized using a compute shader. This initialization shader clears the color and medium texture, calculates the view ray directions and initial positions based on the viewing transformation matrix and initializes the light and light direction textures with the color and direction of the light source. Directions and positions are stored in viewing coordinates.

5.1 Preintegration Lookup Table

When the transfer function is changed, the preintegration lookup table is recomputed in a compute shader. The four transfer functions used in our algorithm require in total eight channels, three for the particle color, three for the medium color, one for opacity and one for indices of refraction. We preintegrate the paricle color, medium color and alpha. The index of refraction transfer function is not preintegrated. The preintegration lookup table must therefore hold seven channels. To facilitate this we store it in a 256 by 256 RGBA texture with two layers. This leaves us with a spare channel that could in principle hold preintegrated values for indices of refraction, but in our implementation we do not use this.

5.2 Ray Propagation

Light and viewing ray propagation are then performed in a single shader program executed for every light plane. A geometry shader first constructs the light plane as a triangle strip. The fragment shader carries the main workload of the algorithm and consists of the two stages described in Sections 4.2 and 4.3. A more detailed outline of the entire algorithm is given in Algorithm 1. During our ray propagation stage, no state changes take place leading to very little driver overhead. At every iteration we first call glTextureBarrier to make sure the work from the previous iteration is completed, followed by one draw call, with one vertex. Which plane the shader is working on is determined by the vertex data of the single vertex supplied in the draw call.

Light Propagation

Light propagation from the previous plane is performed using backward integration. The values to be propagated to the current location are accessed by intersecting, in the direction of the light at the current position, with the previous illumination plane. The intersection point is transformed to screen coordinates, which are used as texture coordinates for looking up the light color and direction. The elliptical convolution kernel by Patel et al. [17] is

used to filter the sampled light, using a pseudorandom rotational offset. The light intensity calculation is performed using GLSL's built-in derivative functions. The ratio between the differences of the current light positions and the intersection points gives us a cheap approximate of the intensity at the current position. The contributions of the volume between the two planes are determined using the preintegration tables for the medium and particle color and opacity and used to compute the new light color. The index of refraction gradient is computed on-the-fly, by applying η_{tf} to the neighboring data values before computing the central differences. It is then used to update the light direction.

Viewing Ray Propagation

The incoming illumination is retrieved from the light texture and specular shading is computed using the volume gradient (which is also computed on-the-fly) and the BRDF model described in Section 4.3. The light texture is accessed by transforming the current position of a viewing ray to screen coordinates, giving us the appropriate sample location. As in the light propagation stage, the contributions of the current ray segment are retrieved from the preintegration tables and are used to compute the new opacity, particle color and medium colors. The ray direction is updated using the refractive index gradient. Finally, the ray position is advanced by intersecting it with the next light plane.

5.3 Post Processing

Finally, after the ray propagation stage, the post process shader is executed to finalize the image and render it to the screen. This stage composites the environment map behind the rendered volume. The sample locations of the environment map are determined by the ray directions in the view ray direction texture. To reduce noisy aliasing of the environment map (due to incoherent ray directions) we store the environment map as a summed area table and use this to calculate the environment color in the polygon formed from the ray directions of the current fragment and three of its neighbours. The environment map is composited behind the rendered volume using Oddy and Willis' model [42], taking into account the alpha and medium color overlaying it.

Data:
lb : light buffer
ldb : light direction buffer
cb : color buffer
mb : medium buffer
vpb : viewing ray position buffer
vdb : viewing ray direction buffer
Δs : sample distance

`initializeBuffers(`*lb,ldb,cb,mb,vpb,vdb*`)`

foreach *plane* $p_i \in$ *planes* **do**
  writelayer $= i \bmod 2$
  readlayer $= 1 - writelayer$
  foreach *fragment* $x \in$ *fragments* **do**
    light propagation
      $ld_i =$ `texture(`*ldb, x, readlayer*`)`
      $lp_{i-1} =$ `intersect(`p_{i-1}*,x,-*ld_i`)`

      $L_{i-1} =$ `filter(`*lb,* lp_{i-1}*, readlayer*`)`
      $ld_{i-1} =$ `filter(`*ldb,* lp_{i-1}*, readlayer*`)`

      $S_i = |$`dFdx(`x`)`$| \cdot |$`dFdy(`x`)`$|$
      $S_{i-1} = |$`dFdx(`lp_{i-1}`)`$| \cdot |$`dFdy(`lp_{i-1}`)`$|$
      $I_i = S_{i-1} \,/\, S_i$

      $[\alpha, m] =$ `integrationTable(`lp_{i-1}*,x*`)`
      $L_i = L_{i-1} \cdot I_i \cdot (1 - \alpha) \cdot m$
      $ld_i = ld_{i-1} + \Delta s \nabla \eta$

      `store(`*lb,* L_i*, x, writelayer*`)`
      `store(`*ldb,* ld_i*, x, writelayer*`)`
    end
    view propagation
      $vp_i =$ `texture(`*vpb, x, readlayer*`)`
      $vd_i =$ `texture(`*vdb, x, readlayer*`)`
      $[C_{i-1}, A_{i-1}] =$ `texture(`*cb, x, readlayer*`)`
      $M_{i-1} =$ `texture(`*mb, x, readlayer*`)`

      $i_d =$ `texture(`*lb, x, readlayer*`)`
      $i_s =$ `specularBDRF(`ld_i*,*vd_i*,*∇f`)`

      $[c, \alpha, m] =$ `integrationTable(`vp_{i-1}*,*vp_i`)`
      $C_i = C_{i-1} + (1 - A_{i-1}) \cdot M_{i-1} \cdot (M_{i-1} \cdot \alpha \cdot c \cdot i_d + i_s)$
      $A_i = A_{i-1} + (1 - A_{i-1}) \cdot \alpha$
      $M_i = M_{i-1} \cdot m$

      $vd_{i+1} = v_i + \Delta s \nabla \eta$
      $vp_{i+1} =$ `intersect(`p_{i+1}*,*vp_i*,*vd_{i+1}`)`

      `store(`*vpb,* vp_{i+1}*, x, writelayer*`)`
      `store(`*vdb,* vd_{i+1}*, x, writelayer*`)`
      `store(`*cb,* $[C_i, A_i]$*, x, writelayer*`)`
      `store(`*mb,* M_i*, x, writelayer*`)`
    end
  end
end

Algorithm 1: Pseudocode of our algorithm. For brevity, sampling of the volume and gradient reconstruction have been omitted.

Chapter 6

Results

In order to demonstrate the capabilities of our algorithm, we show several examples of volumetric datasets with refractive properties specified using transfer functions for particle and medium color as well as refractive index. In practice, the specification of the refractive index transfer function η_{tf} and the medium color transfer function m_{tf} is performed using an additional user interface widget, which allows for the specification of η_{tf} as a curve, while m_{tf} is specified using a color gradient. Initially, η_{tf} is constant one while m_{tf} is constant white, amounting to conventional volume rendering without refraction. While any added complexity to the already non-trivial process of transfer function specification is potentially problematic from a usability point of view, we found these functions surprisingly easy to control.

Figure 6.1 shows a timestep of a supernova simulation lit from above. A ground plane was added to show the effects of refractions on the shadow. The outermost layer uses a purple medium color, but is otherwise transparent revealing inner structures. In Figure 6.1 (a) all refractive indices are equal and the difference between the refractive indices of the outer and inner layers increases from (b) to (d). In addition to the appearance of caustics, the increasing distortion is also clearly visible in the zoom-ins on the bottom of the figure.

The piggy bank dataset, depicted in Figure 6.2, exhibits complex light patterns due to due to its curved nature. We use a combination of white reflective contributions and a yellow media color, to create the appearance of a semi-transparent material such as plastic. The effect of light source softness can be seen by comparing Figure 6.2 (a), (b), (c), and (d) – with increasingly softer light, the caustics smoothen out and become less prominent.

Figure 6.3 demonstrates the importance of refraction for the appearance of transparent structures. In Figure 6.3 (a) a volumetric scene of a glass filled with liquid is rendered without refraction using only reflective material properties (i.e., conventional Porter-Duff compositing without a medium color). Both the glass and the fluid were given low opacity values to make them semi-transparent. In Figure 6.3 (b), we show the same scene rendered using our method, but still with constant refraction indices, i.e., no refraction. However, we use more realistic material properties by specifying negligible absorption (i.e., opacity) for both the glass and the liquid. The colors are solely determined by the medium color. Figure 6.3 (c) specifies the refractive indices for both glass and liquid, but both have zero opacity simulating the appearance of a mostly transparent drink such as beer. Note the colored caustics on the ground and back planes and at the base of the stem. In Figure 6.3 (d), we increase the opacity of the liquid to recreate the appearance of a denser drink such as juice – in comparison to the previous figure, the liquid now absorbs most of the incoming light resulting in a dark shadow on the wall. Furthermore, a semicircular pattern caused by refraction at the rim of the glass becomes visible on the liquid. Finally, in Figure 6.3 (d) we change the medium color of the glass to a green tint and the medium color of the liquid to a red shade with similar refraction indices as in Figure 6.3 (c), resulting in a dark red appearance of the liquid similar to red wine in a green glass.

In Figures 6.4 and 6.5 we can see total internal reflection. Figure 6.4 shows a water like volume with a floor. Light shafts and caustics caused by the wavy surface can be seen. The medium color transfer function is set to a heavy blue-green color. Figure 6.5 shows a model of a ship encased in a refractive blue box-shaped medium and demonstrates total internal reflection. Viewing rays entering the volume are perfectly reflected when they hit the internal walls of the volume. It is worth noting that the total internal reflection is modeled inherently by the ray equation of geometric optics.

Figure 6.6 shows a CT angiography scan of a human head with different refractive indices for air and soft tissue. The medium color for the soft tissue is set to a shade of yellow, while the remaining tissues have mostly opaque reflective properties leading to an appearance similar to an amber trapping. A clipping plane has been specified to set the opacity in one half of the head to zero, but leaving all other material properties unchanged. The distortions due to refraction are clearly visible, as is the refraction on the boundary between the soft tissue and the air-filled lungs in the bottom left part of the image.

Figure 6.7 shows an MRI scan of a kiwifruit rendered with and without refraction. It is worth noting that the image rendered without refraction might

Dataset	Resolution	Render Time	FPS
Supernova	432x432x432	108.53 ms	9.21
Piggy bank	512x512x134	157.04 ms	6.37
Glass	768x768x800	217.20 ms	4.60
Hand	244x124x257	52.77 ms	18.95
Head	512x512x333	205.12 ms	4.88

Table 6.1: Rendering performance as measured on an Intel i5-6600K 3.50 GHz CPU equipped with an NVIDIA GeForce 1070 GTX GPU. Table of performance measurements. The measurements show the average render time over 100 frames with a viewport size of 768×768 and a sample distance of 1.

actually be more realistic than the one with. Even though the flesh of the kiwifruit is fairly translucent, light is scattered too much by the various materials for smooth refractions. It could potentially be argued that the refraction in this instance adds some depth to the material, and thus making features in the kiwi easier to distinguish.

In Figure 6.8, a CT scan of a human hand is shown with transfer functions that use both medium and particle color to highlight different structures. The difference in refractive indices between the individual tissue types give the blood vessels the appearance of colored glass and cause a complex interplay between light and shadow regions.

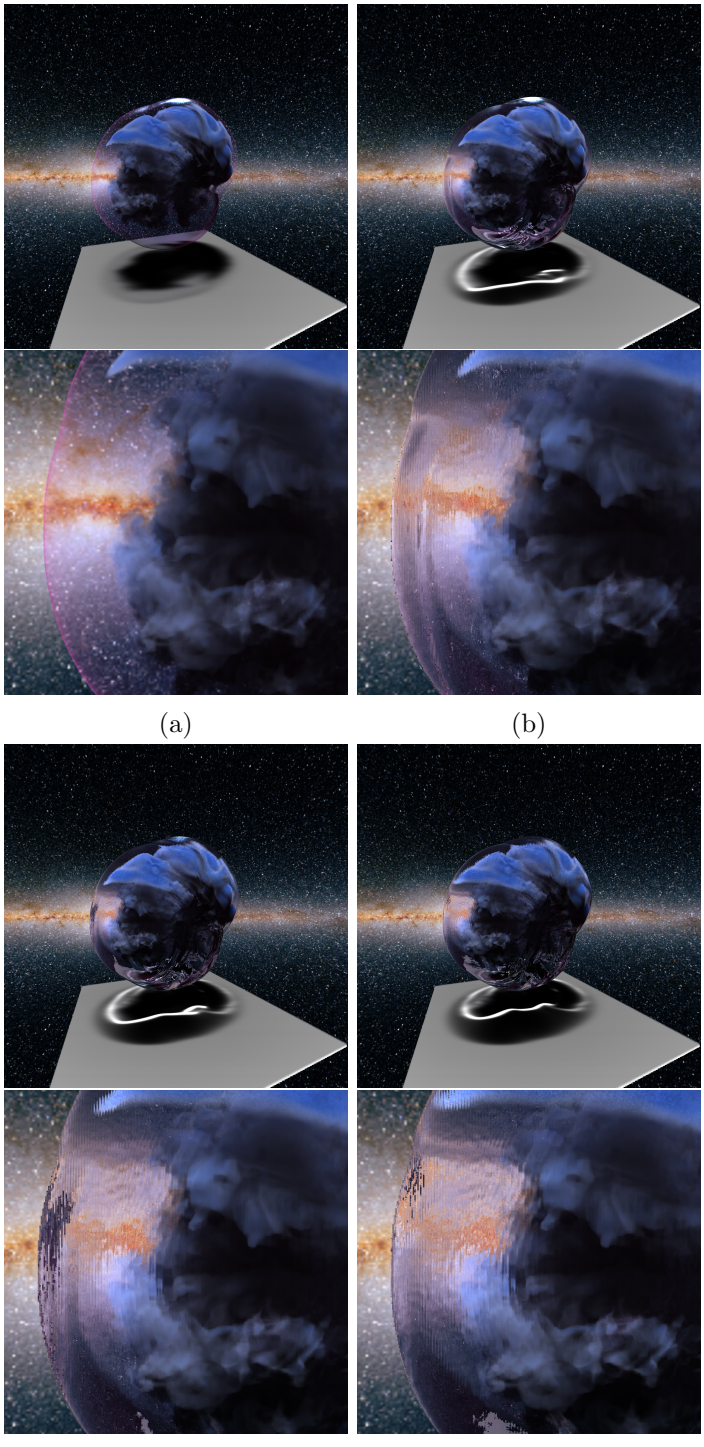
Finally, while it is not our goal to accurately simulate light transport in participating media, but rather to achieve plausible results at interactive frame rates, we show a comparison between our approach and a physically-based renderer in Figure 6.9. Figure 6.9 (a) shows a simple scene rendered with our technique, while Figure 6.9 (b) uses NVIDIA's Iray, a state-of-the-art physically-based global illumination engine. Image generation with Iray took approximately three minutes. While there are obviously a number of differences between the two images, partially caused by different material models, but also due to the fact that our renderer uses a voxelized version of the scene while the original triangle mesh was used in Iray, it can be seen that the main refraction characteristics including the caustics are approximated quite well.

To evaluate the performance of our method, we conducted measurements on an Intel i5-6600K 3.50 GHz CPU and an NVIDIA GeForce 1070 GTX GPU for five of the datasets. Performance was measured on an Intel i7-2600K 4.20 GHz CPU and an NVIDIA GeForce 980 GTX GPU for the three last. In our implementation the sample distance, i.e., the distance between light planes, is specified as a multiplier for the minimum dimensions of a voxel – a sample

Dataset	Resolution	Render Time	FPS
Water	256x256x128	327.31 ms	3.05
Ship	256x256x385	443.69 ms	2.25
Kiwi	256x256x128	155.15 ms	6.45

Table 6.2: Rendering performance as measured on an Intel i7-2600K 4.20 GHz CPU equipped with an NVIDIA GeForce 980 GTX GPU. Table of performance measurements. The measurements show the average render time over 100 frames with a viewport size of 768×768 and a sample distance of 1.

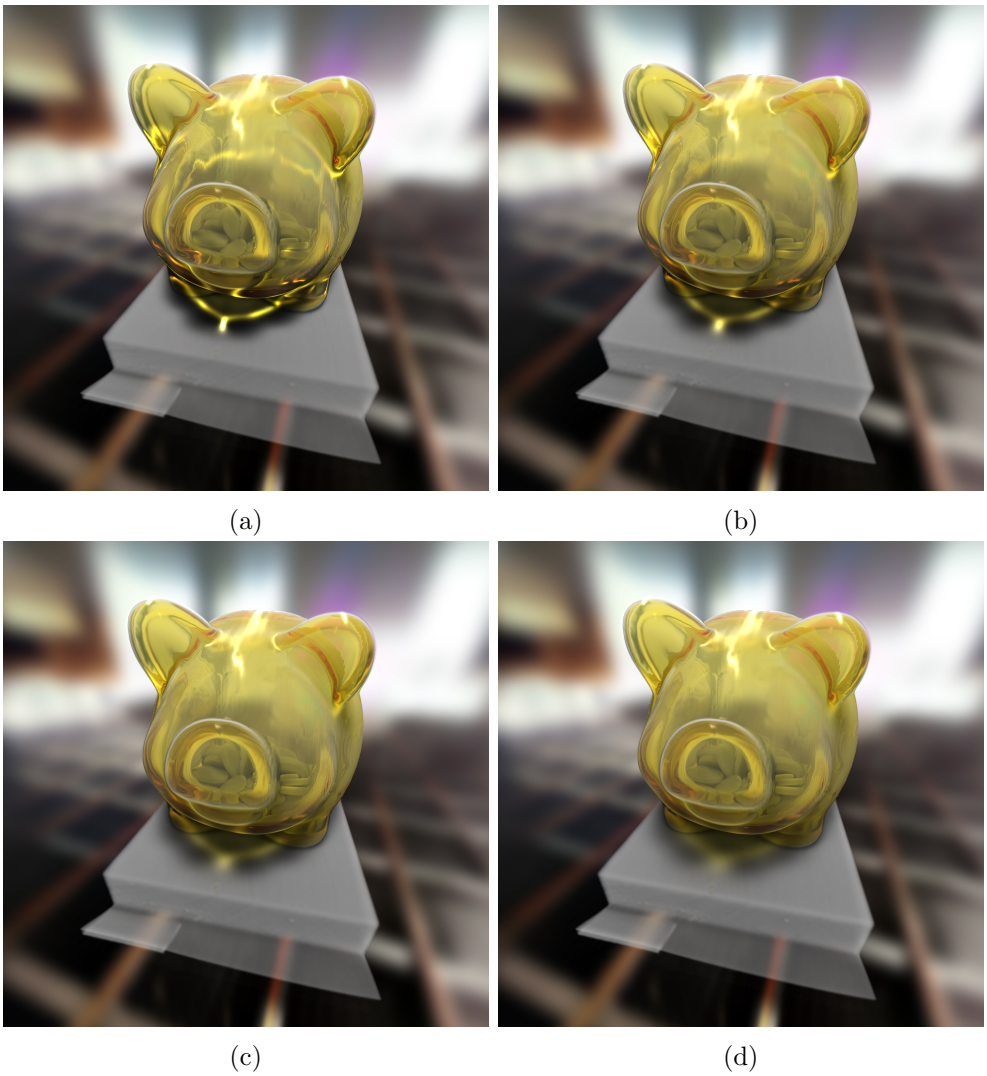
distance of one ensures a minimum of one sample per voxel for all orientations. All results and images in this thesis were generated using a sample distance of one. Table 6.1 lists performance measurements, run on the 1070 GTX machine, of the algorithm on the datasets used to produces the images in this section, while Table 6.2 lists performance measurements run on the 980 GTX machine. Higher resolution datasets are rendered using a higher number of illumination planes and thus generally take longer to render. While naturally, the incorporation of refraction comes at a cost, our algorithm produces high-quality images at subsecond frame rates with all computations performed on-the-fly.



(a) (b)

(c) (d)

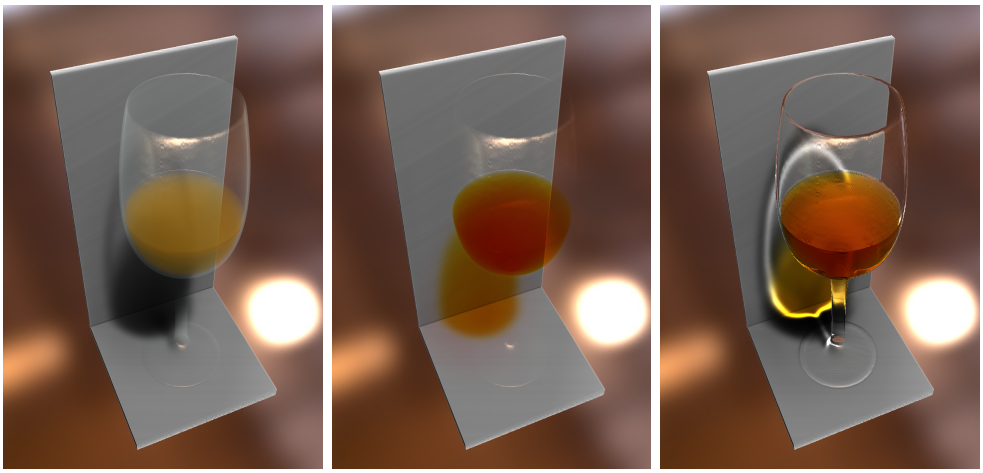
Figure 6.1: One timestep of a supernova simulation with increasing refractive index of the outermost layer from (a) to (d). The bottom row shows zoom-ins of the middle left parts of the top row images.



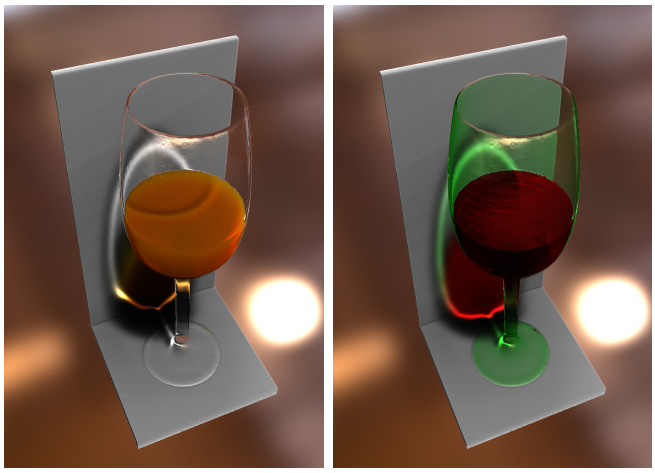
(a) (b)

(c) (d)

Figure 6.2: CT scan of a piggy bank with refraction and combination of transmissive and reflective material properties and increasing light source softness from (a) to (d).



(a) (b) (c)



(d) (e)

Figure 6.3: A volumetric dataset of a filled glass rendered with different material properties. (a) Constant refractive index and only reflective colors, with a constant white medium color. (b) Constant refractive index with varying medium colors. (c) Different refractive indices for glass and liquid with varying medium colors ("beer"). (d) Higher absorption for the liquid ("juice"). (e) Different medium colors for glass and liquid ("red wine in green tinted glass").

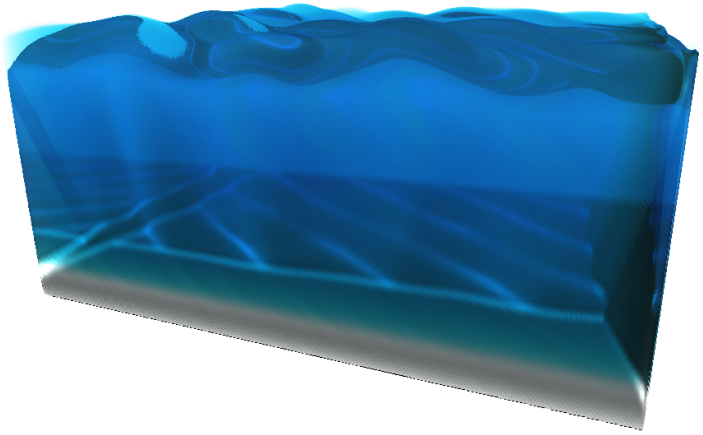


Figure 6.4: Caustics under a wavy surface.

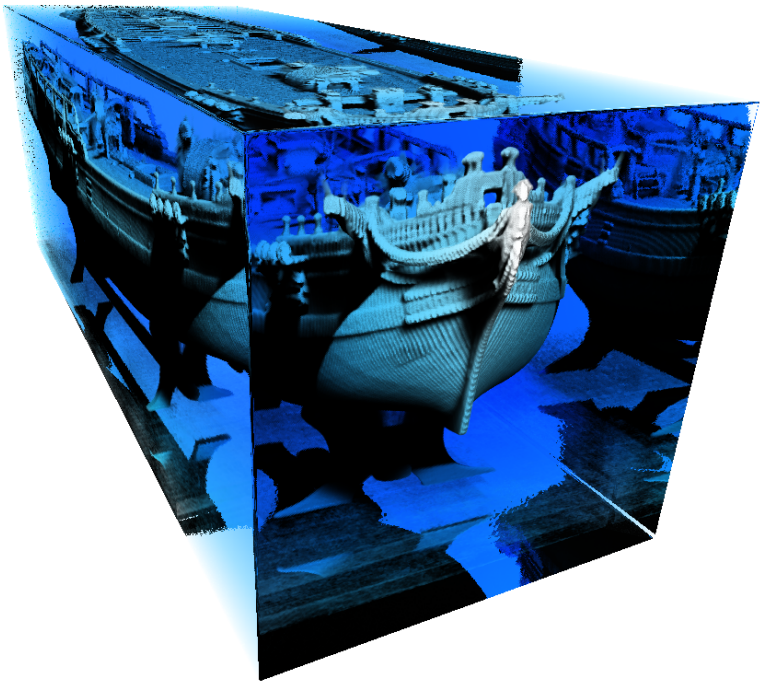


Figure 6.5: Model of a ship in a refractive box.

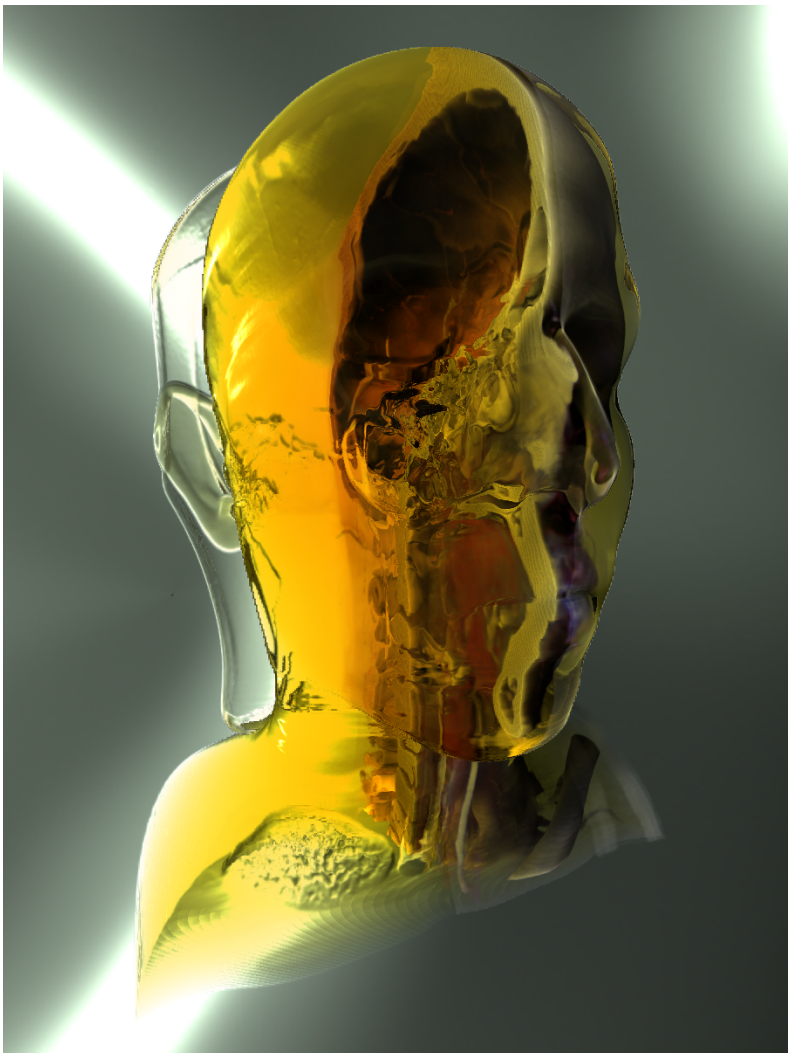


Figure 6.6: CT scan of a human head with different refractive indices for air, soft tissue, and other tissues. Only the denser tissues have opacity contributions, but an additional clipping plane has been specified to set then to zero in parts of the dataset without affecting the transmissive properties.

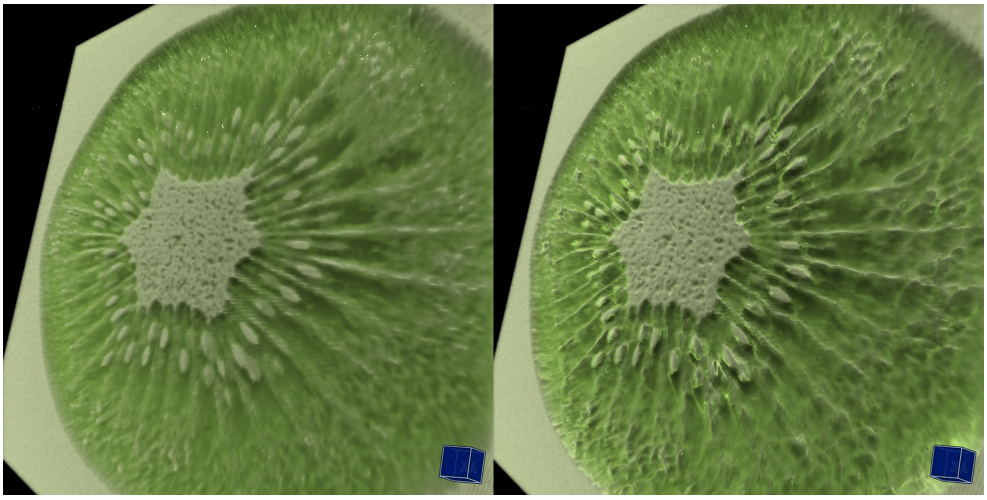


Figure 6.7: MRI of a Kiwifruit. The left image is rendered without refraction while the right is rendered with.

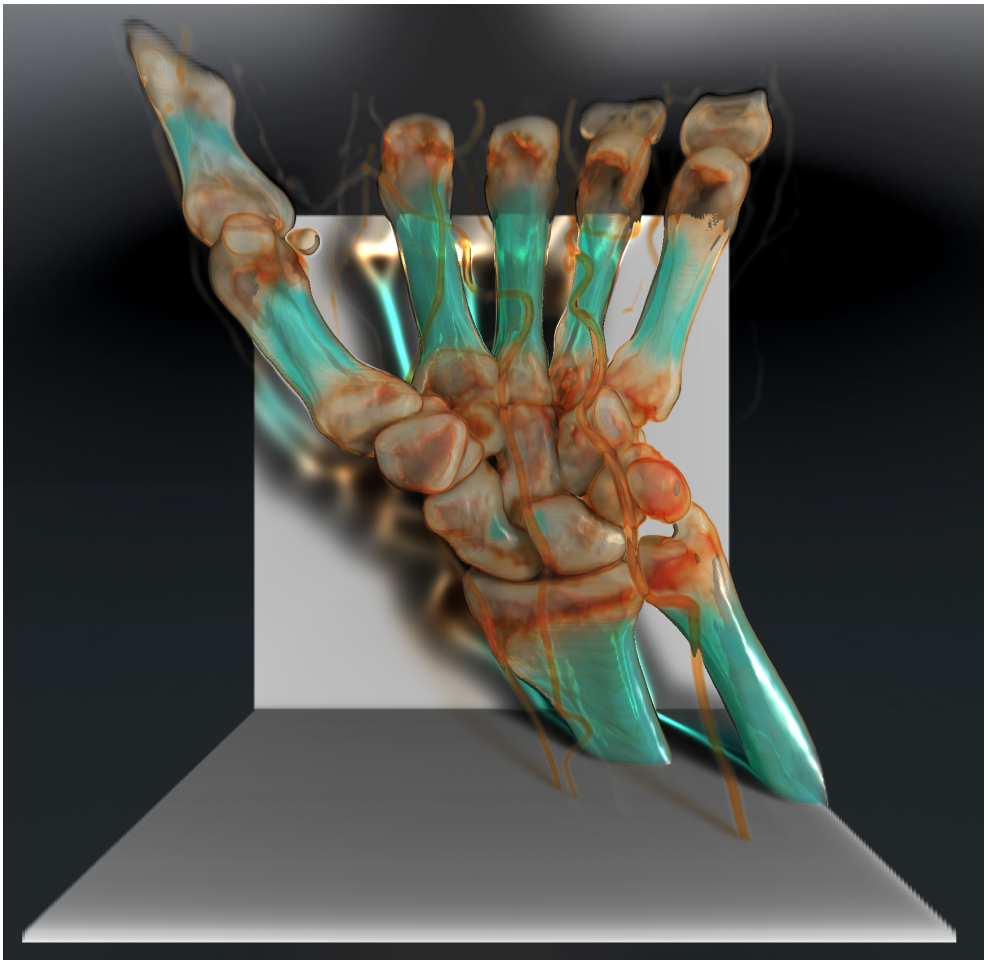
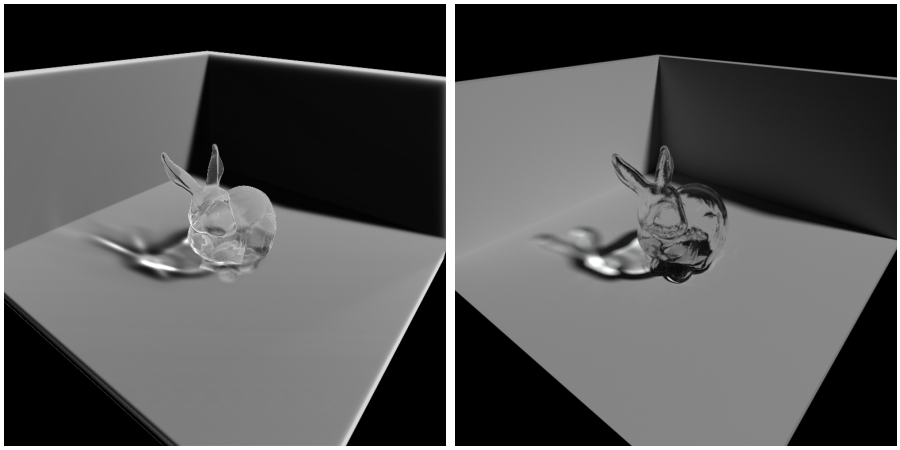


Figure 6.8: CT scan of a human hand with a mixture of transmissive and reflective properties and varying refractive indices.



(a) (b)

Figure 6.9: Comparison of our method to a physically-based offline renderer. (a) Our technique. (b) NVIDIA's Iray.

Chapter 7

Discussion and Limitations

While we have demonstrated that our technique is capable of rendering plausible refractive effects for volumetric data at interactive frame rates, there are also several limitations. At present, our implementation only supports a single distant light source. While, in principle, multiple light sources could be handled, current GPU restrictions on the number of concurrent render targets constrain the number of light buffers. Even without these constraints, however, due to the need of a single coherent traversal direction, all light sources would have to lie on the same hemisphere.

While we currently choose to orient the light plane to coincide with the image plane, this is not a restriction of our approach. An arbitrary orientation could be selected straightforwardly, so it would also be possible two use the half angle plane between view and light direction as proposed by Kniss et al. [14]. While this approach would avoid artifacts at a 90 degree angle between light source and viewing direction, it also results in a slight undersampling of the light buffer for the viewing rays. This could of course be remedied by increasing the light buffer resolution, but this would mean that separate framebuffer objects would have to be used as OpenGL requires all attachments to have the same dimensions incurring a performance overhead. In our experiments, the current solution has proven to be a good trade-off in the majority of the cases and artifacts only appear very close to the 90 degree angle between viewing and light direction. The fact that the light is propagated from slice to slice means that light cannot be refracted more than 90 degrees. This is not a major issue, however, because in most cases such large changes of ray direction do not happen, and in the case where they could occur, the result would likely appear noisy.

Our technique tend to produce inaccurate results when the light source is

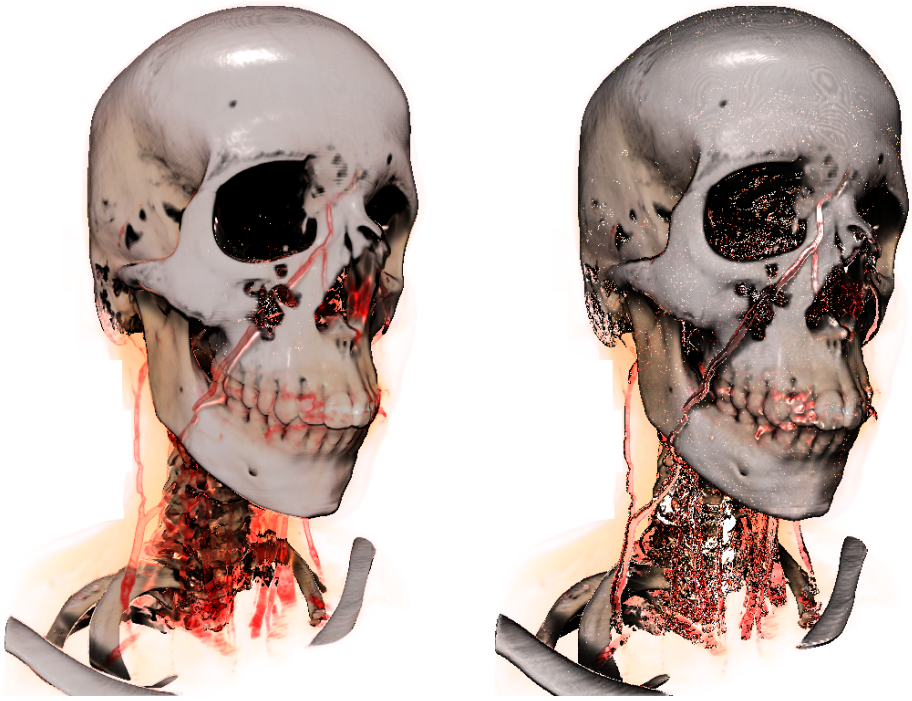


Figure 7.1: Rendering of CT scan of a human head with mild refractive index on the left and high refractive index on the right.

small (sharper shadows). Figure 4.2 from Section 4.2 illustrates this. When the light source is small, artifacts from the backward mapping, the bias toward rays near the boundary of the medium, is more prominent. With a larger light source this effect is diminished as the elliptical filter allows for more light to propagate from the center and spread through the volume.

The way we calculate caustics is prone to cause excessively bright spots when rendering high indices of refraction, often bright enough to pass through nearly opaque media. They occur when the light intensity for the current fragment is calculated from neighbouring fragments whose ray directions vary greatly. This will cause the wavefront element leaving to be very large on the previous plane, causing high intensity as it is concentrated on the small wavefront element around the evaluated fragment. Figure 7.1 shows these bright spots. In the right image, there are several erroneous bright spots, particularly visible in the eye.

Due to the fact that lighting information has to be propagated throughout the volume, our implementation currently only incorporates a limited form of empty space skipping that stops ray traversal as soon as the bounds of the

contributing volume (either through opacity or through transmission) have been reached. While viewing ray computations can be skipped as soon as full opacity has been reached, illumination has to be further propagated and, due to the inherently parallel nature of execution on the GPU, this only leads to limited performance gains. In the future, we plan to investigate more effective strategies for both empty space skipping and early ray termination using features such warp voting.

A further limitation is that our approach treats the index of refraction as a scalar quantity, disregarding the fact that physical refraction also affects the wavelength of light (dispersion). This means that our method is therefore not capable of capturing wavelength-dependent phenomena such as prisms. In principle, this could be handled by specifying separate refractive indices for the spectral samples, at the cost of making their specification more complex. Alternatively, such a spectral refractive index could potentially be derived as a function of the medium color, but we have not yet experimented with this.

For many visualization purposes, such as medical diagnosis, some of the effects of refraction are clearly undesirable. Refracting structures act as lenses, distorting the appearance of other objects. If accurate judgment of sizes or angles is important, such effects should be avoided. On the other hand, similar to shadows, refraction effects may aid in the perception of shapes, supporting an improved perception of properties such as curvature. In fact, distortion lenses as a focus+context tool have a long tradition in many distinct areas of visualization. While a detailed study of the perceptual implications of refraction effects is beyond the scope of this thesis, we believe that this could be an interesting subject for further empirical research expanding on the work of Lindemann and Ropinski [1]. One interesting aspect of refraction, in particular when combined with control over the medium color, is that it can provide information about the presence of objects without introducing occlusion. Hence, a limited degree of refraction could represent an interesting approach for sparsely encoding the presence of contextual structures. In this context, we would also like to note that our method does not constrain the source of the refractive index field, meaning that instead of the original scalar field – as done for all the results in presented in this thesis – a different volume, e.g., in the case of multimodal data, could be used providing an additional optical property that can represent additional information [52]. Moreover, with respect to visualization for presentation, as opposed to analysis or exploration, we believe that selectively used refractive effects can be a meaningful stylistic tool capable of creating compelling images that was previously missing from the volume visualization.

Chapter 8

Conclusions and Future Work

In this thesis, we have presented a novel technique for the rendering of volumetric data with refraction. By interleaving light and viewing ray propagation and using a semi-Lagrangian backward integration scheme for illumination propagation, we are able to generate fully dynamic volume illumination with advanced effects such as soft shadows and caustics. Our method does not use any precomputation and all rendering parameters can be changed interactively. We have demonstrated that our technique is capable of creating plausible approximations of complex refractive phenomena in participating media that were previously only possible in offline renderers.

In our method the volume is traversed in a plane-by-plane manner on view-aligned planes. Viewing rays are propagated by intersecting them with the next plane. The illumination at each plane is computed by backward integration. For each pixel in the illumination buffer, the illumination information are found by looking back to the previous plane in the direction of the light. Soft shadows are realizing by iteratively convolving the illumination buffer by an efficient blurring kernel, requiring only few additional samples. Caustics are calculated based on the intensity law of geometric optics. The optical properties between planes are retrieved from a preintegration lookup table.

As mentioned in chapter 7, our method is not capable of rendering dispersion, that is the splitting of white light into a spectrum of colors. For future work it could be interesting to investigate methods that could enable the rendering of dispersion. One possible way this could be achieved is to refract the red, green and blue channels separately according to their wavelengths. By iteratively convolving a blurring over the light buffer these channels would get mixed potential producing a convincing spectrum. Currently our method produces inaccurate caustics for small light sources, due to the bias the backward

mapping has toward light near boundaries. For larger light sources the bias is remedied by the filtering of the light buffer. Perhaps a different, or differently weighted filter could be employed to remedy the bias without softening the shadows and caustics. The filter would need to improve the amount of light pointing toward the currently evaluated point that gets propagated to this point. In the current implementation, specular reflections are not occluded. Rendering physically based reflections in refractive participating media would require refracted volumetric occlusion. If refracted volumetric occlusion could be accessed either by real-time computations or by looking up preprocessed data, it could be coupled with an environment mapping solution to produce more realistic reflections. Our model only supports refraction through smooth media. It could be interesting to investigate techniques that could allow our model to be expanded to allow the rendering or refraction through rough media. This would require efficient blurring of viewing rays progressing past the rough refraction event while also propagating the light further.

Bibliography

[1] F. Lindemann and T. Ropinski, "About the influence of illumination models on image comprehension in direct volume rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 1922–1931, 2011. DOI: 10.1109/TVCG.2011.161.

[2] D. Gutierrez, A. Munoz, O. Anson, and F. J. Seron, "Non-linear Volume Photon Mapping," in *Eurographics Symposium on Rendering (2005)*, K. Bala and P. Dutre, Eds., The Eurographics Association, 2005. DOI: 10.2312/EGWR/EGSR05/291-300.

[3] S. Chandrasekhar, *Radiative transfer*. New York: Dover Publications, 1960.

[4] N. Max, "Optical models for direct volume rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 1, no. 2, pp. 99–108, 1995. DOI: 10.1109/2945.468400.

[5] D. Jönsson, E. Sundén, A. Ynnerman, and T. Ropinski, "A survey of volumetric illumination techniques for interactive volume rendering," *Computer Graphics Forum*, vol. 33, no. 1, pp. 27–51, 2014. DOI: 10.1111/cgf.12252.

[6] T. Ropinski, J. Meyer-Spradow, S. Diepenbrock, J. Mensmann, and K. Hinrichs, "Interactive volume rendering with dynamic ambient occlusion and color bleeding," *Computer Graphics Forum*, vol. 27, no. 2, pp. 567–576, 2008. DOI: 10.1111/j.1467-8659.2008.01154.x.

[7] F. Hernell, P. Ljung, and A. Ynnerman, "Local ambient occlusion in direct volume rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 4, pp. 548–559, 2010. DOI: 10.1109/TVCG.2009.45.

[8] P. Schlegel, M. Makhinya, and R. Pajarola, "Extinction-based shading and illumination in GPU volume ray-casting," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 1795–1802, 2011. DOI: 10.1109/TVCG.2011.198.

[9] M. Ament, F. Sadlo, and D. Weiskopf, "Ambient volume scattering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 12, pp. 2936–2945, 2013. DOI: 10.1109/TVCG.2013.129.

[10] M. Ament, F. Sadlo, C. Dachsbacher, and D. Weiskopf, "Low-pass filtered volumetric shadows," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2437–2446, 2014. DOI: 10.1109/TVCG.2014.2346333.

[11] Y. Zhang and K.-L. Ma, "Fast global illumination for interactive volume visualization," in *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 2013, pp. 55–62. DOI: 10.1145/2448196.2448205.

[12] Y. Zhang and K.-L. Ma, "Decoupled shading for real-time heterogeneous volume illumination," *Computer Graphics Forum*, vol. 35, no. 3, pp. 401–410, 2016. DOI: 10.1111/cgf.12916.

[13] E. Sundén and T. Ropinski, "Efficient volume illumination with multiple light sources through selective light updates," in *IEEE Pacific Visualization Symposium*, 2015, pp. 231–238. DOI: 10.1109/PACIFICVIS.2015.7156382.

[14] J. Kniss, S. Premoze, C. Hansen, P. Shirley, and A. McPherson, "A model for volume lighting and modeling," *IEEE Transactions on Visualization and Computer Graphics*, vol. 9, no. 2, pp. 150–162, 2003. DOI: 10.1109/TVCG.2003.1196003.

[15] M. Schott, V. Pegoraro, C. Hansen, K. Boulanger, and K. Bouatouch, "A directional occlusion shading model for interactive direct volume rendering," *Computer Graphics Forum*, vol. 28, no. 3, pp. 855–862, 2009. DOI: 10.1111/j.1467-8659.2009.01464.x.

[16] V. Šoltészová, D. Patel, S. Bruckner, and I. Viola, "A multidirectional occlusion shading model for direct volume rendering," *Computer Graphics Forum*, vol. 29, no. 3, pp. 883–891, 2010. DOI: 10.1111/j.1467-8659.2009.01695.x.

[17] D. Patel, V. Šoltészová, J. M. Nordbotten, and S. Bruckner, "Instant convolution shadows for volumetric detail mapping," *ACM Transactions on Graphics*, vol. 32, no. 5, 154:1–154:18, 2013. DOI: 10.1145/2492684.

[18] E. Sundén, A. Ynnerman, and T. Ropinski, "Image plane sweep volume illumination," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2125–2134, 2011. DOI: 10.1109/TVCG.2011.211.

[19] C. Wyman, "An approximate image-space approach for interactive refraction," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 1050–1053, 2005. DOI: 10.1145/1073204.1073310.

[20] C. Wyman and S. T. Davis, "Interactive image-space techniques for approximating caustics," in *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 2006, pp. 153–160. DOI: 10.1145/1111411.1111439.

[21] M. M. Oliveira and M. Brauwers, "Real-time refraction through deformable objects," in *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 2007, pp. 89–96. DOI: 10.1145/1230100.1230116.

[22] S. T. Davis and C. Wyman, "Interactive refractions with total internal reflection," in *Proceedings of Graphics Interface*, 2007, pp. 185–190. DOI: 10.1145/1268517.1268548.

[23] B. Walter, S. R. Marschner, H. Li, and K. E. Torrance, "Microfacet models for refraction through rough surfaces," in *Proceedings of the Eurographics Symposium on Rendering*, 2007, pp. 195–206. DOI: 10.2312/EGWR/EGSR07/195-206.

[24] C. de Rousiers, A. Bousseau, K. Subr, N. Holzschuch, and R. Ramamoorthi, "Real-time rough refraction," in *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 2011, pp. 111–118. DOI: 10.1145/1944745.1944764.

[25] M. Berger, T. Trout, and N. Levit, "Ray tracing mirages," *IEEE Computer Graphics and Applications*, vol. 10, no. 3, pp. 36–41, 1990. DOI: 10.1109/38.55151.

[26] E. Gröller, "Nonlinear ray tracing: Visualizing strange worlds," *The Visual Computer*, vol. 11, no. 5, pp. 263–274, 1995. DOI: 10.1007/BF01901044.

[27] D. Weiskopf, T. Schafhitzel, and T. Ertl, "GPU-based nonlinear ray tracing," *Computer Graphics Forum*, vol. 23, no. 3, pp. 625–633, 2004. DOI: 10.1111/j.1467-8659.2004.00794.x.

[28] J. Stam and E. Languénou, "Ray tracing in non-constant media," in *Rendering Techniques '96. Eurographics*, 1996, pp. 225–234. DOI: 10.1007/978-3-7091-7484-5_23.

[29] I. Ihrke, G. Ziegler, A. Tevs, C. Theobalt, M. Magnor, and H.-P. Seidel, "Eikonal rendering: Efficient light transport in refractive objects," *ACM Transactions on Graphics*, vol. 26, no. 3, 59:1–59:9, 2007. DOI: 10.1145/1276377.1276451.

[30] X. Sun, K. Zhou, E. Stollnitz, J. Shi, and B. Guo, "Interactive relighting of dynamic refractive objects," *ACM Transactions on Graphics*, vol. 27, no. 3, 35:1–35:9, 2008. DOI: 10.1145/1399504.1360634.

[31] C. Cao, Z. Ren, B. Guo, and K. Zhou, "Interactive rendering of non-constant, refractive media using the ray equations of gradient-index optics," *Computer Graphics Forum*, vol. 29, no. 4, pp. 1375–1382, 2010. DOI: 10.1111/j.1467-8659.2010.01733.x.

[32] M. Ament, C. Bergmann, and D. Weiskopf, "Refractive radiative transfer equation," *ACM Transactions on Graphics*, vol. 33, no. 2, 17:1–17:22, 2014. DOI: 10.1145/2557605.

[33] S. Li and K. Mueller, "Spline-based gradient filters for high-quality refraction computations in discrete datasets," in *Proceedings of EuroVis*, 2005, pp. 215–222. DOI: 10.2312/VisSym/EuroVis05/215-222.

[34] S. Li and K. Mueller, "Accelerated, high-quality refraction computations for volume graphics," in *Proceedings of IEEE VGTC Workshop*, 2005, pp. 73–81. DOI: 10.1109/VG.2005.194100.

[35] D. Rodgman and M. Chen, "Refraction in volume graphics," *Graphical Models*, vol. 68, no. 5, pp. 432–450, 2006. DOI: 10.1016/j.gmod.2006.07.003.

[36] K. Engel, M. Hadwiger, J. Kniss, C. Rezk-Salama, and D. Weiskopf, *Real-time volume graphics*. Wellesley: A K Peters, Ltd, 2006.

[37] M. Ament, "Computational visualization of scalar fields," PhD thesis, Faculty of Computer Science, Electrical Engineering and Information Technology, University of Stuttgart, 2014.

[38] T. Porter and T. Duff, "Compositing digital images," *SIGGRAPH Computer graphics and interactive techniques*, vol. 18, no. 3, pp. 253–259, 1984. DOI: 10.1145/964965.808606.

[39] J. P. Schulze, M. Kraus, U. Lang, and T. Ertl, "Integrating pre-integration into the shear-warp algorithm," in *Proceedings of the IEEE TVCG workshop on Volume graphics*, 2003, pp. 109–118. DOI: 10.1145/827051.827068.

[40] K. Engel, M. Kraus, and T. Ertl, "High-quality pre-integrated volume rendering using hardware-accelerated pixel shading," in *Proceedings of the workshop on Graphics hardware*, 2001, pp. 9–16. DOI: 10.1145/383507.383515.

[41] W. Li, K. Mueller, and A. Kaufman, "Empty space skipping and occlusion clipping for texture-based volume rendering," in *Proceedings of IEEE Visualization*, 2003, pp. 317–324. DOI: 10.1109/VISUAL.2003.1250388.

[42] R. J. Oddy and P. J. Willis, "A physically based colour model," *Computer Graphics Forum*, vol. 10, no. 2, pp. 121–127, 1991. DOI: 10.1111/1467-8659.1020121.

[43] B. Jobard, G. Erlebacher, and M. Y. Hussaini, "Lagrangian-eulerian advection of noise and dye textures for unsteady flow visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 8, no. 3, pp. 211–222, 2002. DOI: 10.1109/TVCG.2002.1021575.

[44] M. Born and E. Wolf, *Principles of optics: Electromagnetic theory of propagation, interference and diffraction of light*. Cambridge: Cambridge University Press, 1999.

[45] R. L. Cook and K. E. Torrance, "A reflectance model for computer graphics," *ACM Transactions on Graphics*, vol. 1, no. 1, pp. 7–24, 1982. DOI: 10.1145/357290.357293.

[46] B. Karis, "Real shading in unreal engine 4," 2013, [Online]. Available: http://blog.selfshadow.com/publications/s2013-shading-course/karis/s2013_pbs_epic_notes_v2.pdf.

[47] T. S. Trowbridge and K. P. Reitz, "Average irregularity representation of a rough surface for ray reflection," *Journal of the Optical Society of America*, vol. 65, no. 5, pp. 531–536, 1975. DOI: 10.1364/JOSA.65.000531.

[48] C. Schlick, "An inexpensive brdf model for physically-based rendering," *Computer graphics forum*, vol. 13, no. 3, pp. 233–246, 1994. DOI: 10.1111/1467-8659.1330233.

[49] B. Burley, "Physically-based shading at disney," 2012. [Online]. Available: http://blog.selfshadow.com/publications/s2012-shading-course/burley/s2012_pbs_disney_brdf_notes_v3.pdf.

[50] A. E. Bashirov, E. M. Kurpınar, and A. Özyapıcı, "Multiplicative calculus and its applications," *Journal of Mathematical Analysis and Applications*, vol. 337, no. 1, pp. 36–48, 2008. DOI: 10.1016/j.jmaa.2007.03.081.

[51] L. Florack and H. van Assen, "Multiplicative calculus in biomedical image analysis," *Journal of Mathematical Imaging and Vision*, vol. 42, no. 1, pp. 64–75, 2012. DOI: 10.1007/s10851-011-0275-1.

[52] E. Sundén, S. Kottravel, and T. Ropinski, "Multimodal volume illumination," *Computers and Graphics*, vol. 50, pp. 47–60, 2015. DOI: 10.1016/j.cag.2015.05.004.