Eikonal Rendering: Efficient Light Transport in Refractive Objects

Ivo Ihrke[1], Gernot Ziegler[1], Art Tevs[1], Christian Theobalt[1], Marcus Magnor[2], Hans-Peter Seidel[1]
1) Max-Planck-Institut für Informatik 2) Technical University Braunschweig

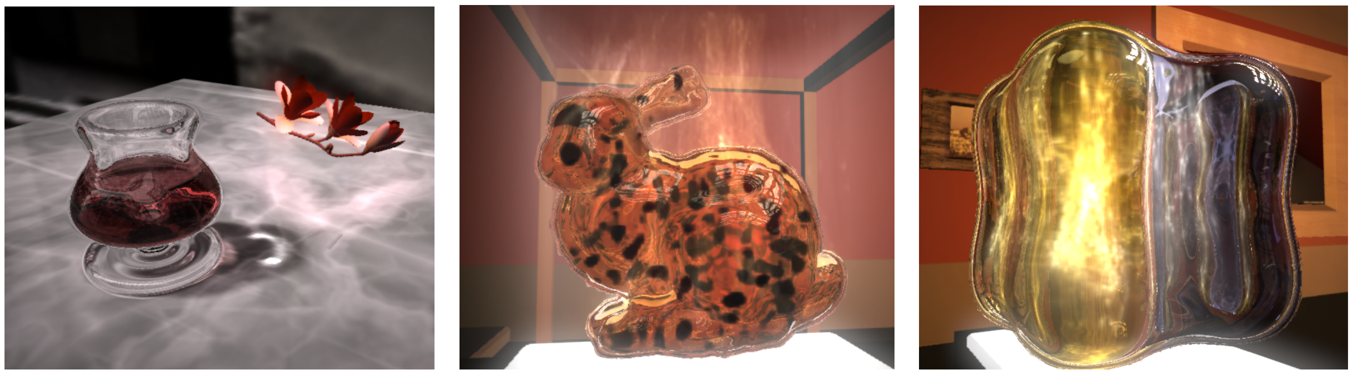
Figure 1: Real-time renderings of complex refractive objects – (left) glass with red wine casting a colorful caustic, 24.8 fps. (middle) Amber-like bunny with black embeddings showing anisotropic scattering and volume caustics in the surrounding smoke and its interior, 13.0 fps. (right) Rounded cube composed of three differently colored and differently refracting kinds of glass showing scattering effects and caustics in its interior, 6.4 fps.

Abstract

We present a new method for real-time rendering of sophisticated lighting effects in and around refractive objects. It enables us to realistically display refractive objects with complex material properties, such as arbitrarily varying refractive index, inhomogeneous attenuation, as well as spatially-varying anisotropic scattering and reflectance properties. User-controlled changes of lighting positions only require a few seconds of update time. Our method is based on a set of ordinary differential equations derived from the eikonal equation, the main postulate of geometric optics. This set of equations allows for fast casting of bent light rays with the complexity of a particle tracer. Based on this concept, we also propose an efficient light propagation technique using adaptive wavefront tracing. Efficient GPU implementations for our algorithmic concepts enable us to render a combination of visual effects that were previously not reproducible in real-time.

CR Categories: I.3.7 [Three-dimensional Graphics and Realism];

Keywords: refractive objects, real-time rendering, light transport, geometric optics

1 Introduction

Objects with complex optical properties, such as a crystal glass filled with wine, are fascinating to look at. This fascination emanates from the beauty of the lighting and color effects that are visible in, on and around these objects. The visual beauty has its physical origin in the interplay of the involved light/matter interaction processes that take place while light passes material boundaries, while it travels through the interior of an object, and even while it interacts with the object's surroundings. At material boundaries, light may be reflected and transmitted in a complex way. A spatially varying refractive index, possibly in conjunction with complex surface reflectance, can cause inhomogeneous focusing of light that becomes visible as beautiful surface and volume caustics. Some materials exhibit spatially varying or wavelength-dependent attenuation which leads to nice color-shifts. Finally, anisotropic scattering effects also greatly contribute to the overall look.

The contribution of this paper is a general framework that allows us to jointly reproduce the majority of the above effects in real-time on commodity graphics hardware. Our image formation model supports the rendering of complex light paths in scenes containing objects with arbitrarily varying refractive index, surface effects with arbitrary BRDFs, as well as view-dependent single-scattering effects with arbitrary scattering phase functions. Advanced effects, such as total reflection, are implicitly obtained at no additional cost. Furthermore, our renderer can reproduce refractive surface and volume caustics, and realistically render the appearance of translucent objects in scattering participating media, such as smoke.

In the following, we first introduce a general, physically motivated image formation model based on a volumetric scene representation that accounts for all these effects, Sect. 3. Subsequently, we describe simplifications to this model that efficiently map to the GPU. For rapid simulation of the light transport, we employ a simple set of ordinary differential equations that is derived from the eikonal equation, the main postulate of geometric optics [Born and Wolf 1999]. Our method enables evaluating complex light paths, both towards the viewer and from the light source, using the same elegant mathematical framework. The distribution of irradiance in the scene due to each light source is quickly pre-computed using a wavefront-based light propagation scheme, Sect. 4. Finally, we propose new concepts and dynamic data structures to efficiently evaluate our image formation model on off-the-shelf graphics hardware, Sect. 5. In Sect. 6, we show results with our prototype GPU renderer displaying a variety of the above effects around refractive objects in real-time.

2 Related Work

Several approaches were published in the literature that can approximate refraction effects in real-time on the GPU [Wyman 2005], on a special signal processor [Ohbuchi 2003], or in a CPU-based real-time ray-tracer [Wald et al. 2002]. Hakura and Snyder [2001] propose a hybrid ray-tracing based approach that produces appealing results but does not run in real-time. Most of these algorithms achieve good results by evaluating Snell's law at material boundaries. Rendering inhomogenous refractive index distributions has been mainly considered in the literature on modeling atmospheric effects. Berger et al. [1990] ray-trace mirages by repeated application of Snell's law in an off-line renderer. Musgrave [1990] includes total reflection which was ignored in the previous paper to render the same phenomenon. Stam and Languénou [1996] propose the use of the ray equation of geometric optics to render heat shimmering. Lately, Gutierrez et al. [2006] have also applied the ray equation to render mirages and other atmospheric effects. Zhao et al. [2007] simulate and render heat shimmering and mirages on the GPU at interactive frame rates. An interesting approach for displaying gemstones that handles refraction and polarization effects was presented by Guy and Soler [2004]. Although our method cannot handle polarization, it can easily cater for many other effects not treated by the above approaches, such as scattering, dispersion or volume caustics in participating media.

Refraction rendering is related to the problem of rendering realistic caustics. Popular off-line approaches for high-quality caustic rendering are backward ray-tracing [Arvo 1986], and photon mapping [Jensen et al. 2001] which can also generate volume caustics [Jensen and Christensen 1998]. Either of them stores photon energies in spatial storage data structures and gathers their contributions during image formation. Gutierrez et al. [2005] extend volumetric photon mapping to non-linear light paths using the ray equation of geometric optics. They simulate refractive effects in the atmosphere and in underwater scenes. In addition to the effects treated in this work, they also render multiple inelastic scattering events in an off-line renderer. Real-time ray-tracing systems [Parker et al. 1999; Carr et al. 2002; Wald et al. 2002] enable the rendering of refraction and photon mapping at discrete interfaces at interactive frame rates [Wyman et al. 2004], but typically a cluster of PCs is needed [Günther et al. 2004] to handle the computational load.

Recently, researchers ported these algorithms to graphics hardware to achieve real-time performance. Wand and Strasser [2003] compute reflective caustics by approximating surfaces with uniformly sampled light sources. Wyman and Davis [2006] propose an interactive image space technique for approximate caustic rendering on the GPU that is related to photon mapping. They also suggest a light shaft structure similar to the illumination volumes of Nishita and Nakamae [1994] that approximates the intensity distribution of the flux of light in a beam in space. A similar concept is employed by Ernst et al. [2005] to generate surface and volume caustics.

In contrast to the above techniques, we employ a more general model of light transport that propagates wavefronts along arbitrary trajectories with arbitrary complex refraction characteristics. Surface and volume caustics can be generated by computing the irradiance distribution everywhere in a sampled 3D volume. We also obtain local light directions for every point in space, enabling us to render anisotropic lighting effects. Our image formation pipeline is based on the theory of geometric optics which enables us, in combination with a powerful image formation model, to faithfully handle a large variety of additional effects, such as dispersion, emission, scattering, BRDFs and spatially varying attenuation within a common framework.

The scattering of light in a volumetric scene description was introduced to computer graphics by Blinn [1982]. Kajiya and von Herzen [1984] derive a general formulation of scattering in terms of volume densities. They present general equations for single and multiple scattering. We use their single scattering equation in our image formation model. Lighting interaction between surfaces and volumes is treated by Rushmeier and Torrance [1987] in a radiosity style algorithm. Stam [1995] explores the limit of multiple scattering and presents a diffusion approximation to this limit. Recently, real-time single scattering implementations have been presented. Magnor et al. [2005] use a GPU ray-casting implementation to render reflection nebulae - this approach is most similar to our scattering, emission and absorption implementation but uses straight viewing and light rays. Mertens et al. [2003] render single subsurface-scattering and a dipole approximation to multiple scattering in real-time using the model by Jensen et al. [2001]. Although we do not approximate multiple scattering, we render single anisotropic scattering along complex non-linear light paths.

The fundamental concepts of our light propagation scheme are derived from the eikonal and transport equations, the main postulate of geometric optics [Born and Wolf 1999]. The curved eye rays are computed as in [Stam and Languénou 1996; Gutierrez et al. 2005; Gutierrez et al. 2006] based on the ray equation of geometric optics. This is similar to non-linear ray tracing [Gröller 1995; Weiskopf et al. 2004] that has been used to simulate gravitational lenses. For the pre-computation of the irradiance distribution in a volume we employ adaptive wavefront tracing. Wavefront-based irradiance estimation techniques have been used infrequently in computer graphics. Mitchell and Hanrahan [1992] compute Fermat paths analytically and evaluate the irradiance at a surface based on wavefront curvature which is tracked along the computed paths. Collins [1994] traces rays from the light source and evaluates the wavefront behavior by examining the distribution of ray hits across diffuse surfaces. Brière and Poulin [2000] suggest to use beam tracing to render surface and volume caustics in an offline approach. Irradiance estimation is based on the intensity law of geometric optics.

To summarize, we present a new fast and versatile framework derived from the eikonal equation that can jointly reproduce many lighting effects around complex refractive objects for which, up to now, individual specialized algorithms were required to obtain real-time frame rates.

3 Image Formation Model

3.1 General Image Formation

We are concerned with the realistic and efficient rendering of transparent objects with varying materials. To this end, we assume that the complex material distribution is stored in a 3D volume. Our general model of image formation accounts for emission, absorption, reflection and scattering. A mathematical formulation for a particular, potentially curved, ray that passes through the volume is given by

$$L(c) = \int_c L_c(\mathbf{x},\mathbf{v})\alpha(t,c)dt + L_{bg}\alpha(t_\infty,c) \ , \qquad (1)$$

where L_c denotes radiance on the ray c that is scattered, emitted or reflected into the direction of the eye. L_{bg} is the background radiance and $\alpha(t,c)$ the absorption of light at position t along the ray. L_c is composed of different components contributing to the radiance on a given ray. Function L_c depends on the position in space $\mathbf{x} = c(t)$ and the local ray direction $\mathbf{v} = \frac{dc}{dt}$. In general it is wavelength-dependent and can be computed using different parameters for each wavelength λ. We can express L_c in terms of these variables:

$$L_c(\mathbf{x},\mathbf{v}) = \hat{\omega} L_s(\mathbf{x},\mathbf{v}) + \delta(\mathbf{x})\rho L_r(\mathbf{x},\mathbf{v}) + L_e(\mathbf{x},\mathbf{v}) \ . \quad (2)$$

Here L_s denotes radiance due to inscatter, $\hat{\omega} = \frac{\sigma_s}{\sigma_a+\sigma_s}$ is the albedo of the participating medium, L_r the radiance reflected in the eye direction, and L_e the locally emitted radiance. The Dirac delta function $\delta(\mathbf{x})$ serves as a boundary indicator, i.e. it integrates to one over a boundary between two different objects and is zero elsewhere. This accounts for the fact that reflections occur on boundaries between different materials. ρ is the Fresnel reflection factor for unpolarized light [Born and Wolf 1999]. The Fresnel transmission factor τ enters the absorption equation (5) through factor $T(t)$, as we will describe later.

L_s, L_r and L_e all depend on the position in space $\mathbf{x}$ and on the local ray direction $\mathbf{v}$ and can be evaluated locally given volumetric descriptions of their distributions. The last point is important. The locality of L_c, given appropriate pre-computations, allows us to parallelize the computations in an efficient way.

We formulate inscatter in terms of the scattering phase function p. It may vary in space and depends further on the local ray direction $\mathbf{v}$ and the local differential irradiance dE_ω from direction ω.

$$L_s(\mathbf{x},\mathbf{v}) = \int_\Omega p(\mathbf{x},\mathbf{v},\omega) dE_\omega \ . \quad (3)$$

The light contributions due to inscatter are integrated over the sphere of incoming directions to yield L_s. Similarly we write

$$L_r(\mathbf{x},\mathbf{v}) = \int_{\Omega_+} f_r(\mathbf{x},\mathbf{v},\omega) \cos\theta dE_\omega \ , \quad (4)$$

where f_r describes a BRDF and $\cos\theta$ is the cosine of the angle between the surface normal and the incident light direction ω. The normal of the surface can either be provided as an additional function or be derived from the refractive index field n. L_r thus gives us the radiance contribution due to reflection on a boundary between two different materials. Please keep in mind that this term is only valid on the boundary of objects and its contribution is triggered by $\delta(\mathbf{x})$.

L_e is just a function $L_e(\mathbf{x},\mathbf{v})$ in its most general form. In conjunction with the light source definitions, it can be used to model multiple scattering effects or self-emission due to fluorescence or phosphorescence.

Finally, we have a closer look at the absorption function α in Eq. (1). If arbitrary absorption distributions are considered, it depends on the distance along the ray and the ray's shape, and thus it evaluates to

$$\alpha(t,c) = T(t) e^{-\int_0^t \sigma_t(c(s)) ds} \ , \quad (5)$$

i.e. the absorption function describes the exponential attenuation of radiance at position $\mathbf{x} = c(t)$ due to a spatially varying attenuation function $\sigma_t = \sigma_a + \sigma_s$, where σ_a is the absorption coefficient and σ_s the scattering coefficient describing the amount of radiance lost due to out-scatter. $T(t)$ is the product of all Fresnel transmission factors τ encountered along the ray up to position t.

3.2 Simplified Image Formation

In its general form, our image formation model is too complex to be evaluated in real-time. Therefore, we make two simplifying assumptions:

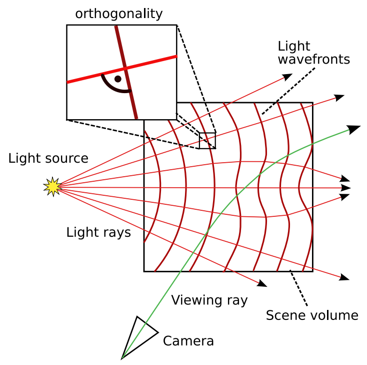

Figure 2: 2D illustration of our complex image formation scenario – due to inhomogeneous material distribution, light rays and viewing rays are bent on their way through the scene volume. Light rays always travel orthogonally to the light wavefronts, i.e. the isosurfaces of constant travel time.

1. The light in the scene originates from a discrete number of light sources, and
2. for each point in the scene, there is only a discrete number of incoming light rays from each of the light sources.

These restrictions allow us to develop an efficient rendering algorithm for a fairly complex image formation model, since we can convert the integrals of Eqs. (3) and (4) into discrete sums over all incoming light directions:

$$L_s(\mathbf{x},\mathbf{v}) = \sum_j p(\mathbf{x},\mathbf{v},\mathbf{l}_j) \Delta E_{\omega j} \quad (6)$$

$$L_r(\mathbf{x},\mathbf{v}) = \sum_j f_r(\mathbf{x},\mathbf{v},\mathbf{l}_j) \cos\theta \Delta E_{\omega j} \ . \quad (7)$$

Thus, if we can pre-compute the incoming light directions and differential irradiance values, we can evaluate Eq. (1) with local operations only. In the following section, we derive the mathematical recipes for viewing ray traversal and irradiance computation.

4 Light Transport

In this section, we develop the equations for the transport of light in the scene. The propagation of viewing rays is described in Sect. 4.1 and light transport is discussed in Sect. 4.2. Viewing rays and light rays, Fig. 2, behave very similarly and the governing equations are derived from the same basic equation, the *ray equation of geometric optics* [Born and Wolf 1999]. However, we use different parameterizations to account for specifics in the two processes. Please note that for light rays, we have to take the irradiance fall-off into account whereas viewing rays carry radiance.

4.1 Viewing Ray Propagation

The ray equation of geometric optics has been previously used in computer graphics e.g. by Stam and Languénou [1996] and Gutierrez et al. [2005]. The equation describes the motion of a light 'particle' in a field n of inhomogeneous refractive indices:

$$\frac{d}{ds}\left(n \frac{d\mathbf{x}}{ds}\right) = \nabla n \ . \quad (8)$$

It is derived from the eikonal equation and the motion of a massless particle along the gradient of the eikonal solution. ds denotes

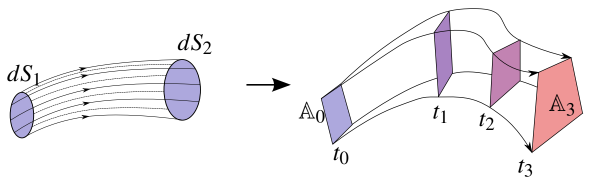

Figure 3: The intensity law of geometric optics (left) and its discretized version (right) in the form of a *stream tube*. The product of area and differential irradiance is constant along a tube of rays.

an infinitesimal step in the direction tangential to the curved ray. Eq. (8) can be re-written as a system of first order ordinary differential equations

$$\frac{d\mathbf{x}}{ds} = \frac{\mathbf{v}}{n} \tag{9}$$

$$\frac{d\mathbf{v}}{ds} = \bigtriangledown n \tag{10}$$

which can be discretized using a simple Euler forward scheme

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \frac{\Delta s}{n}\mathbf{v}_i \tag{11}$$

$$\mathbf{v}_{i+1} = \mathbf{v}_i + \Delta s \bigtriangledown n \tag{12}$$

or some higher order integration method like the Runge-Kutta family [Press et al. 1992]. The equations (9) and (10) have the property that the spatial step size is equal for all ray trajectories, see the Appendix for a proof. This proves advantageous for rendering, Sect. 5, where the number of iterations for each particle trace should be approximately equal to ensure optimal performance. Conveniently, ray bending and total reflection are naturally supported by the ray equation of geometric optics.

4.2 Modeling Light Sources

We model a light source with a three-dimensional vector field of local light directions $\mathbf{l}(\mathbf{x})$ and a scalar field of differential irradiance values $\Delta E_\omega(\mathbf{x})$ (c.f. Sect. 3.2). These fields can be computed in several ways. A popular choice among computer graphics researchers is photon mapping [Jensen 2001] of which GPU implementations are available [Purcell et al. 2003]. In the computational physics and numerical analysis literature a huge range of methods have been proposed to solve this problem. Choices range from purely Eulerian formulations using the eikonal and transport equations [Buske and Kästner 2004], phase space methods [Osher et al. 2002] and hybrid Lagrangian-Eulerian approaches [Benamou 1996] to adaptive wavefront tracing [Enquist and Runborg 2003]. All methods except for the purely Eulerian approach deal with the inherent multi-valuedness of the solution of the underlying equations.

We use adaptive wavefront tracing [Enquist and Runborg 2003; Collins 1997] for the computation of the local light directions and differential irradiance values because it offers the best trade-off between computation time and accuracy of the solution. A wavefront is an iso-surface of constant travel time of light originating from a light source, see Fig. 2. In accordance with Fermat's Principle light rays travel always normal to these wavefronts. The wavefront is discretized by a set of connected particles. These are propagated through the inhomogeneous refractive index field. In case the wavefront becomes under-resolved new particles are inserted to preserve a minimum sampling rate, Fig. 4. The local light directions are represented by the traveling directions of the particles and the differential irradiance values can be computed from the areas of wavefront patches, see Sect. 4.2.2. The pre-computation of the three-dimensional light distribution takes the following subsequent steps:

I wavefront propagation,

II irradiance computation,

III wavefront refinement,

IV voxelization of the local light directions and differential irradiance values.

This process is repeated until the wavefront leaves the volume of interest. The individual steps are detailed in the following.

4.2.1 Wavefront Propagation

We discretize the wavefront into a set of inter-connected particles which are propagated independently. This way, the wavefront is subdivided into so-called wavefront patches whose corners are defined by light particles, Fig. 4 (right). The connectivity information is needed for the differential irradiance computation. The propagation of the particles is performed according to Eq. (8) similar to eye ray propagation, Sect. 4.1. We reparameterize it to yield equitemporal discretization steps:

$$n\frac{d}{dt}\left(n^2\frac{d\mathbf{x}}{dt}\right) = \bigtriangledown n \; . \tag{13}$$

A proof of this property is given in the Appendix. The reparameterization is necessary to enable a simple formulation of the differential irradiance computation described in Sect. 4.2.2. It ensures that all particles stay on a common wavefront over time which is necessary to apply the simple intensity law of geometric optics instead of wavefront curvature tracking schemes as in [Mitchell and Hanrahan 1992; Collins 1994]. Similar to Eqs. (9) and (10) we can write Eq. (13) as a system of first order ordinary differential equations

$$\frac{d\mathbf{x}}{dt} = \frac{\mathbf{v}}{n^2} \tag{14}$$

$$\frac{d\mathbf{v}}{dt} = \frac{\bigtriangledown n}{n} \; . \tag{15}$$

This formulation enables a fast GPU implementation of the wavefront propagation scheme as a particle tracer. Once the wavefront can be tracked over time we can compute the differential irradiance at every point in space from the area of the wavefront patches that connect the particles.

4.2.2 Irradiance Computation

The irradiance computation is based on *the intensity law of geometric optics* [Born and Wolf 1999], see Fig. 3 (left). The law states that in an infinitesimal tube of rays the energy stays constant:

$$dE_{\omega 1} dS_1 = dE_{\omega 2} dS_2 \; . \tag{16}$$

We use a discretized version of the intensity law to update the energy contribution of wavefront patches during propagation. The motion of each patch through the scene describes a so-called stream-tube, Fig. 3 (right). Eq. (16) then reads

$$\Delta E_\omega(t) = \frac{\Delta E_\omega(0)\mathbb{A}(0)}{\mathbb{A}(t)} \; . \tag{17}$$

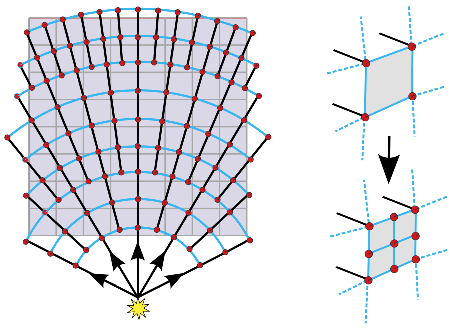
Figure 4: Adaptive wavefront refinement – (left) 2D illustration: the wavefront is represented by particles (red dots) that are connected to form a wavefront (blue lines). While advancing through the voxel volume (shown in gray) the wavefront is tessellated such that its patches span less than a voxel. – (right) 3D illustration of the tessellation for one wavefront patch.

Here $\mathbb{A}(t)$ denotes the area of a wavefront patch at time t and $\Delta E_\omega(t)$ the discretized differential irradiance associated with it. Since we are modeling absorption in our image formation model this effect has to be included in the irradiance computation as well. Therefore, the final discretized differential irradiance for a wavefront patch is given by

$$\Delta E_\omega(t) = \frac{\Delta E_\omega(0)\mathbb{A}(0)}{\mathbb{A}(t)} e^{-\int_0^t \frac{\sigma_t(c(\hat{t}))}{n} d\hat{t}} . \quad (18)$$

4.2.3 Wavefront Refinement and Voxelization

In order to obtain a continuous volumetric representation of the light distribution the wavefront patches have to be voxelized. However, due to divergent corners the patches can in general become arbitrarily large while they are propagated. If a patch area slides through a voxel without touching it with one of its corners, it effectively ignores the influence of this voxel's refraction value. The wavefront will thus be *undersampled*. To alleviate this, we adaptively split the wavefront patches once they grow larger than one voxel, see Fig. 4. Since at the same time, graphics hardware is not able to rasterize arbitrarily sized quads into 3D volumes, we use the adaptive sampling and equate wavefront patches with their midpoints, storing the differential irradiance and directional information as a single voxel sample. Undersampling of the wavefront is thus solved in conjunction with implementing GPU-based voxelization.

5 Implementation Issues

After the theoretical foundation has been set, we now have a closer look at how to map the outlined concepts onto the GPU. Fig. 5 illustrates the work-flow of our renderer. In the following, we detail its most important components, the employed data format, Sect. 5.1, the light simulator, Sect. 5.2 and the view renderer, Sect. 5.3.

5.1 Input Data Format

Input scenes are stored as a set of 3D volume textures. In a first set of volumes, the spatially varying refractive index field, as well as its gradient field are stored. The objects in our test scenes were created as solids of revolution, implicit surfaces, or triangle meshes that we rasterized into the 3D volume. Refractive index distributions can be derived directly from the implicit functions of the objects or they can be defined interactively. Prior to gradient computation, we smooth the volumes. We use a uniform Gaussian filter kernel with a standard deviation of typically $0.5 - 1.0$ voxels, resulting in object boundaries that extend over $2 - 3$ voxels. The problem of blurry boundaries can be alleviated by employing a suitable proxy geometry to start the ray casting. While improving the sharpness of the boundaries and resulting in higher frame rates, participating media surrounding the object can no longer be rendered.

Other 3D textures contain the spatially varying RGB attenuation function, the material boundary indicator, as well as BRDF parameters or emission descriptions. The boundary indicator is a discrete version of the Dirac delta in Eq. (2). We compute it by voxelizing a mesh or use the gradient strength of the refractive index gradients. For some of our test objects, we simulated spatially varying attenuation in the interior by applying a noise function or by painting it into the 2D input for the surface of revolution. For approximating anisotropic scattering effects, we employ the scattering-phase function model by Henyey and Greenstein [1941]. Its parameters are also stored in volumetric textures.

5.2 Light Simulator

Our implementation follows the adaptive wavefront propagation described in Sect. 4. However, since we aim for an efficient simulation also on pre-Shader Model 4.0 hardware, we introduce additional concepts.

Basically, after initialization at the light source, the wavefront is represented as a *particle system*. The difference to a standard particle system is that the particles are bound into packets of four and thus span a *wavefront patch*, Fig. 4 (right). This allows us to simulate the stream tube concept on graphics hardware. All the patches are stored in textures, which hold the four corners' positions, their propagation directions and a RGB energy value, see Sect. 4.2.2.

During initialization, we use the 2D-parameterization of the patch list texture to either produce a planar wavefront (directional light source) or a spherical wavefront (point light source). The initialization ensures that the wavefront is large enough to cover the simulation volume. Other light source types (as multi-directional light) can be implemented, as the wavefront patches are independent and thus can be stacked on top of each other. The propagation of the wavefronts through the scene and the logging into the output 3D volume is performed in three subsequent steps described in the following.

5.2.1 Patch List Update

For every time step, we update the patches' corner positions and directions according to Eqs. (14) and (15). We further update the patches' held RGB energies according to Eq. (18).

5.2.2 Patch List Voxelization

After each update step, we need to protocol the wavefront patches into the 3D output volumes for irradiance and direction. On graphics hardware, this is accomplished using point primitives and the concept of Flat 3D textures introduced by Harris et al. [2003]. We limit ourselves to storing only one incoming light direction, corresponding to the highest energy ray passing a particular voxel. This is justified by a statistical analysis. For the wine glass model, Fig. 8 (right), only 5.6% of the voxels receive light from more than one direction. For these 5.6% of voxels the highest energy ray contributes a mean of 81.6% of the total energy arriving at these voxels. Similar numbers hold for the other models. Before we commit a patch to the 3D volume, we check if it is allowed to overwrite

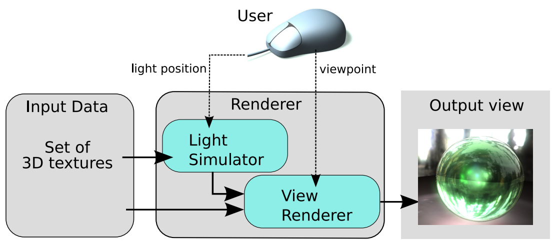

Figure 5: Work-flow of our rendering system.

the one already stored there (if any), based on the highest energy criterion.

5.2.3 Patch List Tessellation Analysis

After each simulation step, the patch list has to be reorganized due to various reasons:

Divergence tessellation: Since the wavefront diverges at places of varying refraction, it must be tessellated to stay below a minimum sampling bound, as outlined in Sect. 4.2. We also have to tessellate the wavefront patches larger than one voxel due to GPU voxelization restrictions. Our simple tessellation currently divides a patch into four smaller ones if its corners span more than one voxel in any direction, Fig. 4.

Patch termination: If a patch holds too little energy, we apply an *energy threshold* to eliminate the patch, assuming it will not contract again and thus nevermore yield a noteworthy energy contribution. Termination typically happens after too much tessellation or loss of energy due to repeated attenuation. We also eliminate patches which leave the simulation volume, since we assume that they will not re-enter it. The physical model of ray optics breaks down at wavefront singularities [Born and Wolf 1999], resulting in infinite energy at catastrophic points, giving rise to non-physical caustics. We detect these areas by examining the patch orientation with respect to its propagation direction. In case the orientation changes, a singular point has been crossed by the wavefront patch and we discard it from further simulation.

Effectively, this means that a patch can have three patch list states: Eliminate (0), Retain (1) or Tessellate (4). The numbers in brackets define the space that each input patch occupies in the output patch list generated for the next simulation step. We conduct the tessellation analysis after the patches' corner directions have been updated. We thus need to reorganize the patch list, which faces us with the non-trivial problem of data compaction and expansion on graphics hardware. Data compaction (i.e. patch elimination) has been solved in the GPU algorithm presented by Ziegler et al. [2006]. The algorithm uses a mipmap-like data structure, the HistoPyramid, to construct a list of retained data entries (here: patches) without involving the CPU. We extend the algorithm to handle patch tessellation (*data expansion*). It utilizes the fact that the original algorithm allocates multiple output positions to an input entry, if this entry is marked with a value larger than one during the HistoPyramid building process. This results in a number of equal copies of the input entry. Instead of receiving four equal patch copies, we then introduce specific behavior in the output generator to tessellate the input into four smaller patches. Doing this, we effectively implement a simple, but very fast adaptive wavefront tessellation. Our algorithm runs without geometry shaders, which are only available on Shader Model 4.0 graphics hardware.

After the new patch list has been generated, it is forwarded to the

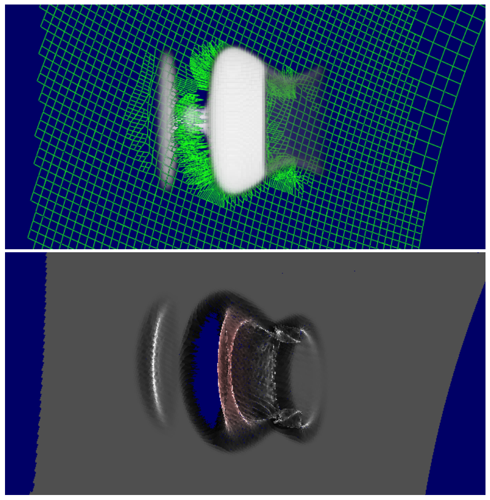

Figure 6: (top) The refractive index volume of the glass is approached by a spherical wavefront from the right. The adaptive tessellation of the wavefront is also visible. – (bottom) When it passes through the object, caustic patterns appear in its irradiance distribution.

patch list update to advance the simulation. This repeats until no patches remain in the simulation volume. In Fig. 6, we show a wavefront propagating through a wine glass. The computed irradiance values are used as colors, a preview on the yielded caustic patterns in and around the object.

5.3 View Renderer

Given the output from the light simulator, we can render arbitrary user views of a complex refractive object. The view renderer implements a fast ray-caster for arbitrarily bent viewing rays based on the update equations (11) and (12). Please note that no explicit ray-surface intersections are required. The radiance along viewing rays is computed according to Eq. (1), using the simplified image formation model and the scene parameters stored in the input textures.

In theory, we can handle arbitrary BRDF models, including parametric or tabulated representations. However, since our glass objects come close to perfect reflectors and a good approximation of the first reflection is already visually pleasing, we use simple dynamic environment mapping. The Fresnel effect and the anisotropic scattering phase function are computed on-the-fly in the fragment shader. Through spatially varying as well as color-channel-dependent attenuation, beautifully colored objects can be reproduced. Optionally, emission can be added, and dispersion effects can be simulated if the input data contain a separate refractive index field for each RGB channel. After the viewing ray has finished volume traversal, we use its exit direction to conduct a lookup into a dynamic environment map to approximate the background radiance. All lighting computations are performed in high dynamic range and an adaptive tone-mapping based on [Krawczyk et al. 2005] is applied prior to display.

6 Results and Discussion

We rendered result sequences with five different objects in several surroundings, thereby visualizing different combinations of effects. The results are shown in Figs. 1 and 8 as well as in the accompanying video. Our first object is a glass block with an embedded colored SIGGRAPH logo. It demonstrates the reproduction of spa-

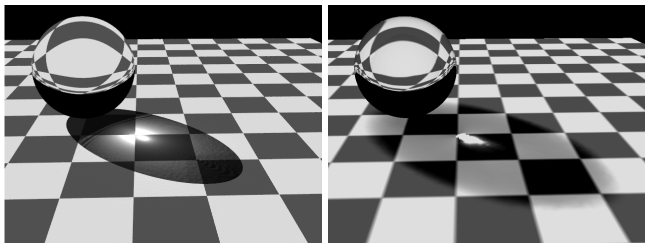
Figure 7: Comparison between a ray-traced image rendered with the Persistence of Vision raytracer (left) and our algorithm (right). The differences in the refraction and shadow size as well as the slightly displaced caustic pattern are due to the smoothing of the refractive index field.

tially varying refraction and attenuation behavior, in particular close to the logo symbol and the text, Fig. 8 (left). On the boundary of the object, total reflection can be observed. Another interesting object is the solid rounded cube which is composed of glass layers with different attenuation characteristics, as well as varying refraction indices, Fig. 1 (right). It also shows anisotropic scattering in its interior visible as sparkles. Focusing of light in the glass leads to volume caustics in its interior. Similar effects can be seen on the glass sphere rendered into a captured real-world environment, Fig. 8 (middle). In addition to all other lighting effects, it has a slight emissive component. We also show a glass filled with red wine, Fig. 1 (left). The glass is illuminated with a directional light source that casts colored caustics onto the table. It also shows interesting and complex refraction effects, as well as appealing surface reflections, Fig. 8 (right). We can also render objects in scattering participating media. Fig. 1 (middle) depicts the glass bunny in a showing case filled with anisotropically scattering smoke. We tuned the attenuation parameters to lend the impression that it is made of amber with black embeddings. It also anisotropically scatters light in its interior.

In the video we first show a light moving behind a SIGGRAPH logo. This is implemented by rendering a particle system into the dynamic environment map. No lighting simulation was performed for this part. The wine glass scene shows the temporal behavior of our wavefront-based irradiance computation scheme. The irradiance distributions are pre-computed. The pre-computation took around 90 minutes for 600 frames. Note that no temporal smoothing has been applied to the irradiance distributions. In the museum scene we simultaneously render 5 refractive objects, and also dynamically update the environment maps. By this means, refractive objects can be seen through other refractive objects, see Fig. 8 (left).

To compare our algorithm against ground truth we rendered a simple test scene, Fig. 7, consisting of a plane and a refractive sphere illuminated by a directional light source. For the purpose of this rendering we replaced the environment map lookup by a ray-plane intersection in the fragment shader. More complex nearby geometry can be rendered accurately using the approach of Hu and Qin [2007]. The difference between the reference solution and our renderer is an artifact of the volumetric discretization and the smoothing of the refractive index fields prior to gradient computation.

Our test data are stored in 128^3 voxel volumes. On an AMD Dual Core Athlon with 2.6 GHz equipped with an NVidia GeForce 8800 GTX and 768 MB of video RAM, we obtain a sustained rendering frame rate of around 25 fps at a resolution of 800×600 pixels if one object is displayed and if the light source remains in a fixed position, Fig. 1 (left) and Fig. 8 (middle)[1]. Mind that the frame rate decreases when zooming in very closely, since more rays need to be cast from the viewpoint into the volume. Also note that the screenshots in Fig. 1 (middle) and (right) and Fig. 8 (left) and (right) are taken from a scene containing 5 refractive objects for which dynamic environment maps have to be rendered. These frame rates are thus not representative for rendering a single refractive object. After moving a light source to a new position, the lighting simulation has to be re-run once. This typically takes around 5 to 10 seconds.

For our particular application, the ODE-based ray propagation and adaptive wavefront tracing formulation have a couple of intriguing advantages. The voxel representation allows for fast non-linear ray casting. Expensive ray/geometry intersections, like in [Purcell et al. 2003], would lead to performance bottlenecks on complex curved light paths. Adaptive wavefront tracing also enables us to simulate non-linear light transport with a fast particle tracer while simultaneously avoiding undersampling problems. Our update times after light position changes are comparable to other state-of-the art GPU methods reproducing fewer effects, e.g. only caustics in isotropic media [Ernst et al. 2005]. We see an advantage over alternative methods like photon-mapping [Jensen 2001] because we only insert particles when needed, i.e. when the wavefront is undersampled. We obtain densely sampled irradiance and directional information throughout 3D space, such that we can cater for anisotropic visual effects at any point in the scene. Also, no special reconstruction kernels are required. Furthermore, we obtain a physically plausible[2] light distribution with significantly reduced sampling problems. An advantage over PDE approaches is the fast simulation and the ability to pick a particular solution in case of multi-valuedness of the light distribution. For a particular point in space, we choose the ray carrying the highest energy. With additional memory consumption and higher algorithmic complexity multi-valued solutions could be computed as well, e.g. using multiple rendering targets.

Despite these advantages for refractive object rendering, on general scenes our algorithm does not match the power of related approaches like photon mapping, which can efficiently produce full global illumination solutions. The required level of discretization makes our method only suitable for a simulation of spatially confined refractive objects. These objects may appear as part of larger scenes by computing standard irradiance fall-offs for mesh based objects and lighting surfaces falling into our simulation volume with the pre-computed volumetric irradiance values. Due to the volumetric representation, the scene's size is mainly limited by the available video memory. Octree representations can help to further reduce memory consumption. Besides, with future generations of graphics boards, memory limits will become less of an issue. Furthermore, we are dependent on decent gradient fields to yield visually pleasing results. To this end, we pre-smooth the refractive index volumes prior to gradient evaluation. Here, one needs to take care to not over-smooth which would lead to halo-effects around material boundaries. A sufficiently high voxelization level is needed for extreme close-up renderings. Otherwise, discretization artifacts in the lighting effects may occur.

7 Conclusions

We presented a fast and versatile method to render a variety of sophisticated lighting effects in and around refractive objects in real-time. It is based on a sophisticated model of light transport in volumetric scene representations that accounts for a variety of effects,

[1]see captions for exact fps in individual scenes

[2]within the limits of geometrical optics, see [Born and Wolf 1999] for details

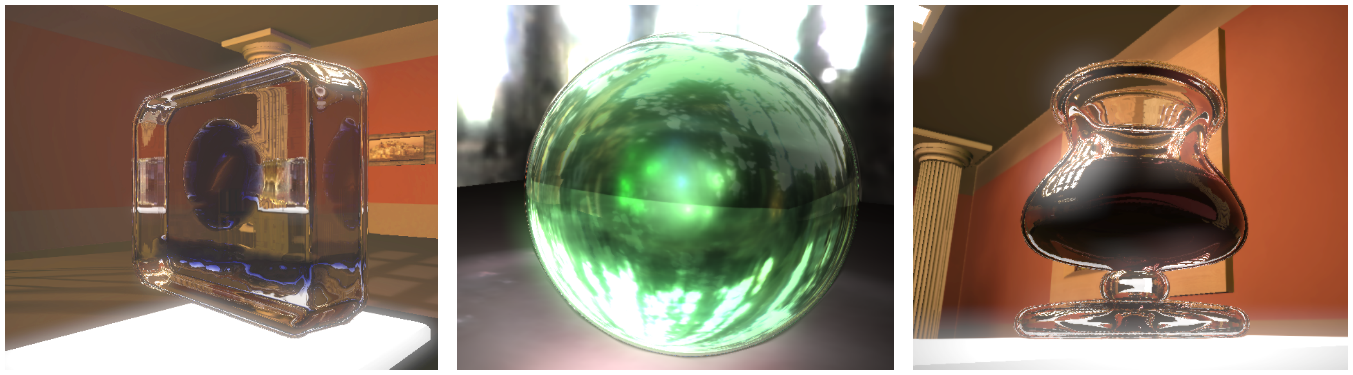

Figure 8: (left) Glass block with embedded SIGGRAPH logo of different refraction and attenuation, 15.5 fps, (5 objects in scene). (middle) Colored sphere rendered into an HDR environment map showing slight emission in addition to all other effects, 26.2 fps. (right) Complex refraction patterns in the glass, 13.7 fps, (5 objects in scene). – Also note the surface reflections and the total reflections within, as well as the rounded cube being visible through the glass block.

including refraction, reflection, anisotropic scattering, emission and attenuation. It employs a fast particle tracing method derived from the eikonal equation that enables us to efficiently simulate non-linear viewing rays and complex propagating light wavefronts on graphics hardware. To implement adaptive wavefront tracing on the GPU, we have developed new data structures to perform geometry tessellation that even runs on pre-Shader Model 4.0 architectures.

Acknowledgements

We would like to thank Anders Sundstedt and Michael Schantin for help in scene modeling and GusGus for giving us permission to use their music in the video. This project was supported by the Max-Planck-Center for Visual Computing and Communication MPI Informatik/Stanford.

Appendix

We derive a constant spatial and a constant temporal step size parameterization of the ray equation of geometric optics. Eq. (8) is derived by combining the eikonal equation

$$|\bigtriangledown S| = n \tag{19}$$

and the equation of a particle moving normal to the wavefronts $S = const$.

$$\frac{d\mathbf{x}}{ds} = \frac{\bigtriangledown S}{|\bigtriangledown S|}. \tag{20}$$

S is a solution of the eikonal equation and iso-surfaces of this function are called wavefronts. They are surfaces of constant travel time from the light source. The derivation of Eq. (8) can be found in [Born and Wolf 1999].

Parameterization with constant spatial step size

Using Eq. (20) we immediately have

$$|\frac{d\mathbf{x}}{ds}|^2 = \frac{d\mathbf{x}}{ds} \cdot \frac{d\mathbf{x}}{ds} = 1. \tag{21}$$

Inserting Eq. (19) into Eq. (20) yields

$$n\frac{d\mathbf{x}}{ds} = \bigtriangledown S. \tag{22}$$

Setting $\mathbf{v} = \bigtriangledown S$ we obtain a parameterization with constant spatial step size ds, Eqs. (9) and (10).

Parameterization with constant temporal step size

We are looking for a parameterization where

$$\frac{dS}{dt} = \bigtriangledown S \cdot \frac{d\mathbf{x}}{dt} = 1, \tag{23}$$

i.e. the infinitesimal change of the eikonal function S with respect to the parameter t is constant. Inserting Eq. (22) into Eq. (23) yields

$$\frac{1}{n} = \frac{d\mathbf{x}}{ds} \cdot \frac{d\mathbf{x}}{dt} = \frac{d\mathbf{x}}{ds} \cdot \frac{d\mathbf{x}}{ds}\frac{ds}{dt} = \frac{ds}{dt}, \tag{24}$$

where the last identity is due to Eq. (21). Using this result we perform a change of parameters using the chain rule and obtain Eqs. (14) and (15) from Eqs. (9) and (10).

References

ARVO, J. R. 1986. Backward Ray Tracing. In *ACM SIGGRAPH '86 Course Notes - Developments in Ray Tracing*, vol. 12.

BENAMOU, J.-D. 1996. Big ray tracing: Multivalued travel time field computation using viscosity solutions of the eikonal equation. *Journal of Computational Physics 128*, 2, 463–474.

BERGER, M., TROUT, T., AND LEVIT, N. 1990. Ray tracing mirages. *IEEE CGAA 10*, 3, 36–41.

BLINN, J. 1982. Light reflection functions for simulation of clouds and dusty surfaces. In *Proc. of SIGGRAPH'82*, 21–29.

BORN, M., AND WOLF, E. 1999. *Principles of Optics, seventh edition*. Cambridge University Press.

BRIÈRE, N., AND POULIN, P. 2000. Adaptive Representation of Specular Light Flux. In *Proc. of Graphics Interface*, 127–136.

BUSKE, S., AND KÄSTNER, U. 2004. Efficient and Accurate Computation of Seismic Traveltimes and Amplitudes . *Geophysical Prospecting 52*, 313–322.

CARR, N. A., HALL, J. D., AND HART, J. C. 2002. The ray engine. In *Proc. of Graphics Hardware*, 37–46.

COLLINS, S. 1994. Adaptive Splatting for Specular to Diffuse Light Transport. In *Proc. of EGWR*, 119–135.

COLLINS, S. 1997. *Wavefront Tracking for Global Illumination Solutions*. PhD thesis, Department of Computer Science, Trinity College Dublin.

ENQUIST, B., AND RUNBORG, O. 2003. Computational High Frequency Wave Propagation. *Acta Numerica 12*, 181–266.

ERNST, M., MOELLER, T. A., AND JENSEN, H. W. 2005. Interactive rendering of caustics using interpolated warped volumes. In *Proc. of GI*, 87–96.

GRÖLLER, E. 1995. Nonlinear ray tracing: visualizing strange worlds. *The Visual Computer 11*, 5, 263–274.

GÜNTHER, J., WALD, I., AND SLUSALLEK, P. 2004. Realtime caustics using distributed photon mapping. In *Proc. of EGSR*, 111–121.

GUTIERREZ, D., MUÑOZ, A., ANSON, O., AND SERON, F. J. 2005. Non-linear volume photon mapping. In *Proc. of EGSR*, 291–300.

GUTIERREZ, D., SERON, F. J., MUÑOZ, A., AND ANSON, O. 2006. Simulation of Atmospheric Phenomena. *Computers & Graphics 30*, 6, 994–1010.

GUY, S., AND SOLER, C. 2004. Graphics gems revisited: fast and physically-based rendering of gemstones. In *Proc. of SIGGRAPH'04*, 231–238.

HAKURA, Z. S., AND SNYDER, J. M. 2001. Realistic reflections and refractions on graphics hardware with hybrid rendering and layered environment maps. In *Proc. of EGSR*, 289–300.

HARRIS, M., BAXTER, W., SCHEUERMANN, T., AND LASTRA, A. 2003. Simulation of cloud dynamics on graphics hardware. In *Proc. of Graphics Hardware*, 92–101.

HENYEY, L. G., AND GREENSTEIN, J. L. 1941. Diffuse Radiation in the Galaxy. *Astrophysical Journal 93*, 70–83.

HU, W., AND QIN, K. 2007. Interactive Approximate Rendering of Reflections, Refractions, and Caustics. *IEEE TVCG 13*, 1, 46–57.

JENSEN, H. W., AND CHRISTENSEN, P. H. 1998. Efficient simulation of light transport in scences with participating media using photon maps. In *Proc. of SIGGRAPH'98*, ACM Press, 311–320.

JENSEN, H. W., MARSCHNER, S. R., LEVOY, M., AND HANRAHAN, P. 2001. A practical model for subsurface light transport. In *Proc. of SIGGRAPH'01*, ACM Press, 511–518.

JENSEN, H. W. 2001. *Realistic Image Synthesis Using Photon Mapping*. AK Peters.

KAJIYA, J., AND VON HERZEN, B. 1984. Ray tracing volume densities. In *Proc. of SIGGRAPH'84*, 165–174.

KRAWCZYK, G., MYSZKOWSKI, K., AND SEIDEL, H.-P. 2005. Perceptual effects in real-time tone mapping. In *Proc. of Spring Conference on Computer Graphics*, ACM, 195–202.

MAGNOR, M., HILDEBRAND, K., LINŢU, A., AND HANSON, A. 2005. Reflection Nebula Visualization. In *Proc.of IEEE Visualization*, 255–262.

MERTENS, T., KAUTZ, J., BEKAERT, P., SEIDEL, H.-P., AND REETH, F. V. 2003. Interactive rendering of translucent deformable objects. In *Proc. of EGRW'03*, 130–140.

MITCHELL, D., AND HANRAHAN, P. 1992. Illumination from curved reflectors. In *Proc. of SIGGRAPH '92*, 283–291.

MUSGRAVE, F. K. 1990. Ray tracing mirages. *IEEE CGAA 10*, 6, 10–12.

NISHITA, T., AND NAKAMAE, E. 1994. Method of displaying optical effects within water using accumulation buffer. In *Proc. of SIGGRAPH'94*, ACM Press, 373–379.

OHBUCHI, E. 2003. A real-time refraction renderer for volume objects using a polygon-rendering scheme. In *Proc. of CGI*, 190–195.

OSHER, S., CHENG, L.-T., KANG, M., SHIM, H., AND TSAI, Y.-H. 2002. Geometric Optics in a Phase-Space-Based Level Set and Eulerian Framework. *Journal of Computational Physics 179*, 2, 622–648.

PARKER, S., MARTIN, W., SLOAN, P., SHIRLEY, P., SMITS, B., AND HANSEN, C. 1999. Interactive ray tracing. In *Proc. of I3D*, ACM Press, 119–126.

PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. 1992. *Numerical Recipes in C*. Cambridge University Press.

PURCELL, T. J., DONNER, C., CAMMARANO, M., JENSEN, H. W., AND HANRAHAN, P. 2003. Photon mapping on programmable graphics hardware. In *Proc. of Graphics Hardware*, 41–50.

RUSHMEIER, H., AND TORRANCE, K. 1987. The zonal method for calculating light intensities in the presence of a participating medium. In *Proc. of SIGGRAPH'87*, 293–302.

STAM, J., AND LANGUÉNOU, E. 1996. Ray-tracing in nonconstant media. In *Proc. of EGSR*, 225–234.

STAM, J. 1995. Multiple Scattering as a Diffusion Process. In *Proc. of EGSR*, 41–50.

WALD, I., BENTHIN, C., SLUSALLEK, P., KOLLIG, T., AND KELLER, A. 2002. Interactive global illumination using fast ray tracing. In *Proc. of EGSR*, 15–24.

WAND, M., AND STRASSER, W. 2003. Real-time caustics. *Computer Graphics Forum (Eurographics 2003) 22*, 3, 611–620.

WEISKOPF, D., SCHAFHITZEL, T., AND ERTL, T. 2004. GPU-Based Nonlinear Ray Tracing. *Computer Graphics Forum (Eurographics 2004) 23*, 3, 625–633.

WYMAN, C., AND DAVIS, S. 2006. Interactive image-space techniques for approximating caustics. In *Proceedings of ACM I3D*, 153–160.

WYMAN, C., HANSEN, C., AND SHIRLEY, P. 2004. Interactive caustics using local precomputed irradiance. In *Proc. of Pacific Graphics*, 143–151.

WYMAN, C. 2005. An approximate image-space approach for interactive refraction. In *Proc. of SIGGRAPH'05*, 1050–1053.

ZHAO, Y., HAN, Y., FAN, Z., QIU, F., KUO, Y.-C., KAUFMAN, A. E., AND MUELLER, K. 2007. Visual Simulation of Heat Shimmering and Mirage. *IEEE TVCG 13*, 1, 179–189.

ZIEGLER, G., THEOBALT, C., AND SEIDEL, H.-P. 2006. On-the-fly point clouds through histogram pyramids. In *Proc. of VMV*, 137–144.