Caustics, Light Shafts, God Rays

Anna Frühstück*
Vienna University of Technology
0626930

Stefanie Prast†
Vienna University of Technology
0727540

Abstract

Lighting effects, such as caustics and light shafts are an important component of the rendering of global illumination images. The correct depiction of the interaction of light with different surfaces is crucial to the realism of any rendered scene. Dealing with the complexity of global illumination has long been among the biggest challenges in computer graphics, a problem that is even more prominent when it comes to rendering interactive environments. Particularly the simulation of caustics is a difficult task since they can only be rendered satisfactorily through techniques which trace the light from the illuminants.
Several different techniques to speed up the process of rendering realistic global illumination effects have been developed. Among those are path tracing, ray tracing and photon mapping. Most state-of-the-art rendering techniques rely heavily on the computation power of the GPU. We wish to present a survey of current rendering techniques for approximating physically exact representations of caustics, light shafts and god rays.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Display algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Radiosity;

Keywords: GPU, rendering, real-time, caustics, god rays, light effects, refraction, reflection

1 Introduction

In computer graphics, the usage of well-simulated lighting effects plays a decisive role in the degree of realism of the global illumination of a rendered scene. For the accomplishment of satisfactory results, several different approaches have been proposed. Since the current state of hardware technology does not yet allow for the tracing of each particle of light in a scene in a reasonable amount of rendering time, the different techniques try to achieve excellent results through simplification and approximation while keeping the computation cost as low as possible. The necessity of minimizing the rendering time is even more significant in interactive environments, where the simulation of realistic lighting effects has to be computed in real-time.
Ever since powerful rendering devices such as GPUs have been developed and can be utilized in the graphics pipeline, significant gains in rendering speed have been achieved, evermore facilitating more complex operations performed in real-time, constantly drawing nearer to photorealism.
The purposes for renderings are manifold and lie, among others, in architecture, video games, simulators or special effects for movies. Since the research in rendering attempts to achieve as realistic images as possible, many sorts of physical behavior and natural phenomena formed by the interaction of light with objects are taken into account in a rendering pipeline. These effects include shadows, transparency, depth of field, motion blur and participating media.
The interest of this report is focused on three of these light effects (caustics, light shafts and god rays) and the manifold attempts to achieve realistic simulation of these effects in three-dimensional scenes.
This paper wishes to give an overview of the different rendering techniques used to simulate the following effects.

Figure 1: (image courtesy of Rob Ireton)
One of the the most prominent occurrence of caustics are the patterns that form on the floor of shallow water when sunlight is refracted by the moving water surface. The fascination of humans with this physical phenomenon has led to many attempts of simulating underwater caustics in computer graphics.

2 Light effects

When light interacts with matter, different sorts of optical phenomena may be observed. To some of them, we are so accustomed we hardly notice them in daily life. Others are less frequently to be seen and are considered natural spectacles. In the quest of achieving the highest degree of realism in a rendered scene, it is indispensable to take these effects into account when computing the scene's global illumination. However, when it comes to rendering, due to computational constraints, the nature of some of the lighting effects poses a challenge.
In the following, we introduce the lighting effects that will be discussed in this paper.

2.1 Caustics

Caustics are patterns of light, such as those formed by sunlight on the floor of a swimming pool (See figure 1), that originate from a focusing of light through reflection or refraction by a specular surface. They occur whenever rays of light get reflected or refracted by some optical medium and converge at a single point on a diffuse surface, thus forming areas of above-average brightness.

*e-Mail: anna.fruehstueck@student.tuwien.ac.at
†e-Mail: stefanie.prast@gmx.at

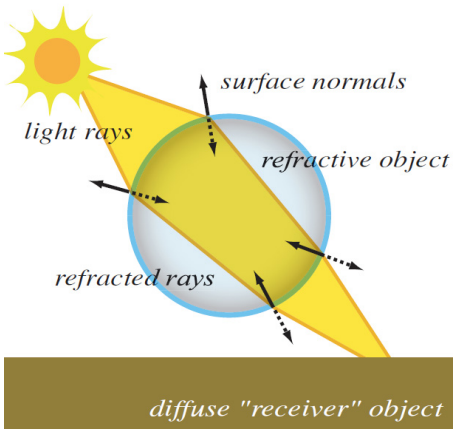

Figure 2: (image courtesy of Shah et al.)
This illustration demonstrates how several light rays, refracted through a curved object, converge at the same point on a non-specular surface, thus generating caustic patterns.

The origin of the terminus 'caustics' is derived from the Latin 'causticus' (which itself borrows from the Greek 'kaustikos'), signifying 'corrosive, burning'.

Another common occurrence of caustics can be perceived when bright light shines on a glass object, e.g. a wine glass. While the glass will cast a shadow, there is also a pattern of bright light formed where the light rays, refracted through the concave body of the glass, concentrate in specific regions.
Distinction is made between catacaustics and diacaustics.
Catacaustics develop through directional reflection from a curved specular surface. Such an effect can be observed at the base of a cup of coffee when light shines onto the inside of the cup diagonally.
The more spectacular form of caustics are diacaustics, produced by refraction through a transparent medium. The effects in the swimming pool and next to a glass object described above are examples for diacaustics.
In 3D computer graphics, the simulation of caustics has long been one of the most difficult tasks because they can only be well approximated by tracing the light forward from its origin.

2.2 Light shafts

The term light shafts describes an optical phenomenon that is perceived as beams of light in a shadowed environment, radiating from the light source. The reason the penetrating light can be distinct as beams from the darker ambiance is that small particles in the air like dust, mist or smoke reflect and scatter the light. Figure 3 shows an example of very strong light shafts caused by the sun breaking in through some windows.
Any small particles can provide the participating media required to make the light's path, otherwise not perceptible, visible, for instance, the smaller, the more continuous is the perception of the beam.
Typical situations where light shafts can be perceived include the observation of flashlights or light houses, but also natural formations where light incides through an opening into a dusk environment.
On some purposes, light shafts are not only an interesting sideeffect, but are generated purposefully for the reason of the dramatic effect the light shafts brings about: Laser beams and searchlights

Figure 3: (image courtesy of flickr user *somebody_*)
The effect of perceiving light inciding into a darkened environment as shafts of bright light is caused by minuscule particles suspended in the media through which the light is passing that scatter the light, thus visualizing not only the general brightening of the setting caused by the light but also the path it describes through the scene. Since occurrences of these light shafts give a very dramatic look to the scenario, many applications of rendering try to incorporate this effect, thus inspiring some academic and industrial research interest in the area.

are exemplary for this objective, on occasions, their effect is even enhanced by the use of fog machines to provide for the participating media.

2.3 God rays

God rays, or crepuscular rays, is a more specific expression for light beams mostly used for the light shafts that occur when the sun is partly covered by some shadow-inducing object (mostly clouds, but also trees, buildings, or any other object). The sunlight breaks through gaps in the obstacle and produces the light rays through reflection on particles in the atmosphere. The seemingly diverging rays are virtually parallel, but perspective makes them look like they were ascending. (See figure 4 for an illustration of the phenomenon)
Crepuscular rays are also frequently to be observed in underwater scenes when looking up toward the water surface and the sun above. The principle, of course, is the same; in this case, the water particles serve as scatterer.

3 Fundamentals

This chapter serves as a quick introduction to several fundamental physical principles that are indispensable to every algorithm that attempts to simulate caustics or god rays.

3.1 Geometrical optics

3.1.1 The rendering equation

The rendering equation, proposed by James Kajiya [Kajiya 1986] in 1986, is based on the physics of light and describes the law of

Figure 4: (image courtesy of flickr user *san_sci*)
As the term 'God rays' evidently suggests, the phenomenon of light beam occurrences in natural environments has fascinated man and was inspiration to a whole number of imaginative names and stories with the same effect: Jacob's Ladder (originating from the biblical story), Ropes of Maui (descending from and old Maori tale) and many more.

conservation of energy during the flow of light through the rendered scene. As it theoretically provides perfect results, various rendering techniques try to approximate the rendering equation and it has thus served as a fundamental basis for all algorithms simulating global illumination.

The following representation of the equation describes the amount of light that is reflected from the surface point x in direction of the vector $\vec{w}$:

$$L_o(x,\vec{w}) = L_e(x,\vec{w}) + \int_\Omega f_r(x,\vec{w}' \to \vec{w}) L_i(x,\vec{w}')(\vec{w}' \cdot \vec{n})\,\mathrm{d}\vec{w}'$$

In words:
For the position x and direction $\vec{w}$, the radiated light L_o is equal to the sum of the self-emitted light L_e (in case the point at x is an emitter) plus the reflected light.
This reflective part is an integral over the whole of all inward directions within a hemisphere; each integrand consists of the incoming light L_i from the direction $\vec{w}'$ multiplied by the extinction of light due to the incident angle $(\vec{w}' \cdot \vec{n})$ as well as the surface reflection f_r with incidence $\vec{w}'$ and emergent angle $\vec{w}$.
The surface reflection component f_r is a bidirectional reflectance distribution function, which will be explained in detail in section 3.1.4.

3.1.2 Refraction

In optics, refraction occurs when a light wave's direction is changed due to an alteration of its speed. Upon passing from one medium with a given refractive index η_1 into another with the index η_2 where the propagation velocity is different, the course of the light ray is altered according to the proportion of the two indices.
Occurrences of this phenomenon can easily be observed when light enters and leaves some sort of transparent medium, e.g. glass or water.
The angle of incidence θ lies between the light ray as it approaches the surface r_i and the surface normal $\vec{n}$, the angle of refraction θ_t is measured between the inverse normal $\vec{n}_t$ and the emergent refracted

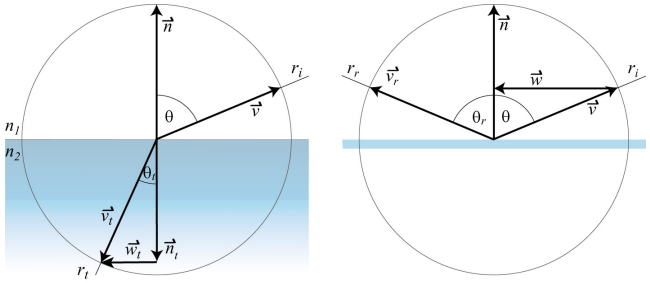

Figure 5: *illustrations to the principle of refraction (left) and reflection (right)*

ray. These parameters are illustrated in the left part of figure 3.1.2.

Snell's law states that the ratio of the sines of the angles θ and θ_t equals the ratio of the wave speeds v_1 and v_2 in the respective media as well as the ratio of the opposite refractive indices.

$$\frac{\sin\theta}{\sin\theta_t} = \frac{\eta_1}{\eta_2} \qquad \frac{\sin\theta}{\sin\theta_t} = \frac{v_1}{v_2}$$

The refracted angle θ_t may be calculated using the pythagorean trigonometric identity $\sin^2\theta_t + \cos^2\theta_t = 1$.
To substitute $\sin^2\theta_t$, a transformation of Snell's equation is used.

$$\begin{aligned}
\cos^2\theta_t &= 1 - \sin^2\theta_t \\
&= 1 - \left(\frac{\eta_1}{\eta_2}\right)^2 \sin^2\theta \\
&= 1 - \left(\left(\frac{\eta_1}{\eta_2}\right)^2 (1 - \cos^2\theta)\right)
\end{aligned}$$

3.1.3 Reflection

The effect in which the light wave's direction at striking the surface of a medium with a different refractive index is altered such that the course of the ray will return into the medium where the wave originated, is termed reflection.
For the reflection off a specular surface, e.g. a mirror, the law of reflection states that the angle θ of incidence between the incoming light ray r_i and the normal vector $\vec{n}$ to the interface equals the angle of reflection θ_r between the $\vec{n}$ and the outgoing light ray r_r. (See figure 3.1.2 on the right)

3.1.4 The bidirectional reflectance distribution function

The bidirectional reflectance distribution function (BRDF) is a function that defines the reflecting behavior of a surface from arbitrary incidences. For any ray striking the surface at a specified angle of incidence, the BRDF specifies the ratio between the radiance reflected in direction $\vec{w}$ and the irradiance incident from direction $\vec{w}'$.

$$f_r(x,\vec{w}' \to \vec{w}) \equiv \frac{L_r(x,\vec{w})}{E_i(x,\vec{w}')} \equiv \frac{L_r(x,\vec{w})}{L_i(x,\vec{w}')\cos(\theta_i)d\vec{w}'}$$

where E_i denotes the irradiance, i.e., the amount of flux hitting the surface, and L_r is the radiance, or differential flux density emitted from the surface.
The BRDF is an integral part of the rendering equation, representing the surface reflection component f_r.

3.1.5 The Fresnel equations

As light, traveling through a medium with refractive index η_1, hits another medium with refractive index η_2, reflection may occur as well as refraction. The Fresnel equations give a quantitative definition of the reflection and refraction of light off an interface. Thus, given the incident angle θ and the refractive indices η_1 and η_2, the intensity of the reflected (reflectance R) as well as the transmitted light rays (transmittance T) can be determined.
Distinction is made between s-polarized light and p-polarized light, the two of which differ in the proportion of the polarization of the waves to the plane of incidence, which is spanned by the surface normal and the propagation vector of the incoming ray; the polarization of p-polarized waves runs parallel to the plane of incidence, whereas s-polarized waves are perpendicular to the plane of incidence. For p-polarized light, the reflectance R is given by

$$R_p = \left(\frac{n_1 \cos\theta_t - n_2 \cos\theta}{n_1 \cos\theta_t + n_2 \cos\theta} \right)^2 = \left[\frac{\tan(\theta_t - \theta)}{\tan(\theta_t + \theta)} \right]^2$$

If the incident light is s-polarized, the reflectance R is given by

$$R_s = \left(\frac{n_1 \cos\theta - n_2 \cos\theta_t}{n_1 \cos\theta + n_2 \cos\theta_t} \right)^2 = \left[\frac{\sin(\theta_t - \theta)}{\sin(\theta_t + \theta)} \right]^2$$

Due to the law of conservation of energy, the transmission coefficient is calculated by

$$T_p = 1 - R_p \qquad T_s = 1 - R_s$$

If the polarization of the incident light consists of an equal share of s- and p-polarizations, the coefficients are calculated by summing the coefficients for p-polarization and s-polarization and dividing the result by two.

4 Caustics

When first attempts were made to integrate caustics simulation into rendering systems, caustics only used to be rendered offline employing either photon mapping or backward ray-tracing techniques; the generation of realistic looking results necessitates the tracing of a very large quantity of either rays or light particles, a process which could not be executed in real-time with the limited computational power of the early 1990s.

James Arvo [Arvo 1986] developed the backward ray-tracing (or light ray tracing) method and observed early that effective algorithms for caustic generation always consist of at least two passes: the first pass identifying the locations of the final hits of the light rays or photons, the second pass rendering the result into an image as seen from the camera position, taking into account the number of photon hits for each non-specular surface point.

By the late 1980s, different approaches to render global illumination and therefore lighting effects, had over the years been developed. At that time, the most realistic technique to light a scene was a combination of ray tracing and radiosity [Wallace et al. 1989]. Despite various attempts to optimize them, both methods had an excessively high computation time and were memory consuming; additionally, they also produced inaccurate and deficient results.

A big breakthrough came with the development of photon mapping by Henrik Wann Jensen in 1995. A year later he published a more advanced approach [Jensen 1996] with two distinct photon maps; a technique that generates caustics implicitly, whereas previous methods had always demanded intricate computation for caustics calculation.
The photon mapping approach is basically an extension of ray tracing-based techniques, which provides means for calculating a scene's global illumination more efficiently, thus permitting to render greater realism at the same or less expense.

4.1 Backward Ray-Tracing

The motivation for Arvo's backward ray-tracing method [Arvo 1986] was to generate an indirect illumination algorithm that can also deal with half-shades.
The first of two passes implements a path tracing algorithm, during which rays are emitted from the light sources into the scene. Whenever one of the rays hits a diffuse surface, it's radiance is reduced according to the reflection and the energy difference is saved in the so-called illumination map at the location of the light ray's collision with the scene geometry.
The second pass carries out conventional ray tracing, with the illumination maps adding to the calculation of the surfaces' illumination.
Thus, one pass is calculated from each light source and another from the camera. The illumination map, which is applied to compute the value at the surface to which both passes' informations contribute, consists of a two-dimensional raster of data points (scalar values for white light or RGB triplets for colored light). It is connected to the object's surface by a function $T(u,v) \leftarrow (x,y,z)$. This function's inverse provides the necessary informations about the locations of the light rays' intersections with the scene geometry as well as the energy that they contain.
When a ray hits a surface that is connected to an illumination map, the respective coordinates u and v are calculated and a part of the ray's energy is stored in the illumination map at the corresponding location. The contribution is divided among the four adjacent data points through bilinear separation and added to their current values (closer points receive a higher quota).
At this point, the illumination map represents solely the accumulated energy of each surface point, but not the direction the light ray came from. The energy that is stored in the illumination map in the course of this process has to be converted to the adequate intensity by dividing the energy $E(u,v)$ at a specific data point by the corresponding area.
Up to this point, the backward ray tracing technique is independent of the view point.
The procedure is extremely costly because the illumination map has to have a very high resolution, also, a large number of rays in the path tracing pass is required to achieve good results.
Backward ray tracing is a method that can deal with caustics, however it only qualifies for scenes with punctiform light sources, extensive illuminants can only be approximated.

4.2 Path tracing

The path tracing approach is a generalization of ray tracing and therefore follows the path of viewing rays from the viewpoint through the scene, taking into account the interaction of the viewing rays with the scene geometry. Through the nature of the path tracing technique, light effects such as caustics, ambient occlusion or shadows are inherently generated in the course of the rendering process and do therefore not have to be added manually.
Images rendered with path tracing feature a superior quality to

most other rendering techniques, at the cost of greater demands on computation power. The accuracy that path traced images provide makes the results this approach generates a reference for the evaluation of other rendering techniques. However, the rendering requires a large number of view rays in order to achieve results without noise.
The path tracing approach was first described by James Kajyiya [Kajiya 1986] in 1986 in his presentation of the rendering equation, which quickly became a benchmark in computer graphics. Opposed to backward ray tracing, path tracing does not aim to follow the paths of the light, instead it attempts to trace the paths from the camera into the scene, pursuing them as they bounce off surfaces and to find out whether the view ray intersects a light source. Due to the fact that the probability of such an intersection is rather low for a random ray emitted into the scene, most traced paths will not contribute to the final image, causing high computational expense.
The main difference between path tracing and conventional ray tracing is that in the ray tracing procedure, on detecting an intersection between a ray and a non-specular surface, the lights are sampled directly. Contrary to this, in path tracing, as the ray hits the surface, the ray's new direction is calculated randomly. It is chosen among all possible direction within the hemisphere over the surface at the incident point, using BRDFs. The thus generated ray is then traced further around the scene, interacting with other surfaces and potentially reaching a light source.
Several attempts have been made to delimitate the inefficiency of this algorithm that lies in the number of rays that are traced even though they never reach a light source.

4.2.1 Bidirectional Path Tracing

Among these, particularly bidirectional path tracing is notable. This technique attempts to combine the respective advantages of both path tracing and backward ray tracing, first pursuing the light rays through the scene, then applying the above described path tracing algorithm, but this time checking whether the light paths strike one of the previously calculated light rays until either a hit is detected or the predefined number of permitted light bounces has been exceeded. This technique causes the algorithm to converge much faster than any single-directional tracing method can hope to achieve.

4.2.2 Metropolis Light Transport

Eric Veach and Leonidas J. Guibas [Veach and Guibas 1997] proposed a modification to the basic bidirectional path tracing method called Metropolis Light Transport (MLT).
The Metropolis Light Transport technique employs the bidirectional path tracing algorithm at first to determine the paths from the camera to the light source but then modifies the resulting paths, applying the Metropolis-Hastings algorithm (a variant of the Monte Carlo Method).
Said algorithm that is applied for the calculation of the final distribution of light among the scene geometry can, once a path from the view point to the light has been detected, explore and create potential nearby light paths which may otherwise be difficult and costly to determine wherein lies the advantage compared to basic bidirectional path tracing.
This method may, but does not necessarily result in a faster calculation of the final light distribution.

4.3 Photon mapping

A photon is defined as a single light particle that transports a certain radiant flux, a large number of which is emitted into the scene from each light source. Geometrically, this approach is equivalent to ray tracing, emanating from the light source.
Photon mapping [Jensen 1996] generates global illumination in two passes. At the heart of the photon mapping algorithm lies the construction of the photon maps which store the location of the final photon hits on non-specular surfaces.

4.3.1 Emission Of The Photons

The first pass concerns the construction of said photon maps. For this purpose, a light source emits a very large number of photons into the scene. The ultimate image quality is heavily dependent on the quantity of photons, which has to be traded off against the photon number's direct impact on the algorithm's running time. The photons represent light particles (in Jensen's publication labeled "energy packages"), each featuring its respective radiance and direction. The radiance of the photons depends on the type of the light source. Since different types of light sources emit different amounts of energy, it is important that the physical law of power conservation should be considered, and the respective energy of each light source has to be divided between the amount of its individual photons to get a realistically looking illumination of the scene. The outgoing direction of the photons depends on the volume, the surface and the geometry of the light source. These photons are pursued through the scene employing a technique similar to path tracing, called photon tracing. As stated in chapter 3, in physics, light which hits a surface can be absorbed, reflected, refracted, scattered or a combination of the previous. Since it would be virtually impossible to compute physically correct light behavior, it has to be simplified.
When a photon hits a surface, a Monte Carlo method called 'Russian roulette' is used to decide whether it will be reflected or absorbed. Calculating the reflected photon's new direction, the bidirectional reflectance distribution function is employed (See chapter 3.1.4). Reflected photons are traced further until they are absorbed. Every reflection and absorption is stored in a photon map.

4.3.2 Construction Of The Photon Maps

Two different kinds of photon maps are applied in the photon mapping algorithm: the global photon map and the caustic photon map. For the creation of a caustic map, photons are only aimed at objects that can cause caustics, i.e., transparent and specular objects. Reflected photons are stored in this map whenever they hit a diffuse surface. Thus, the caustic map contains all photons which underwent specular reflection or refraction.
As for the global photon map, photons are emitted into the whole scene and are stored when they hit a diffuse surface, photons that reflect from a specular surface are not traced any further. This map is responsible for the indirect illumination of nonspecular surfaces.
The degree of precision of the caustic map is important to generate accurate caustics. The radiance of the scene is approximated through the global map. Since the global map is not directly visualized, less photons are needed for the estimation of the lighting than for the caustic map.
The application of two photon maps greatly improves the speed and accuracy of the results, and it reduces the memory consumption of the technique.
To manage the huge mass of gathered information, the key to an acceptable computing time lies in the choice of the right data

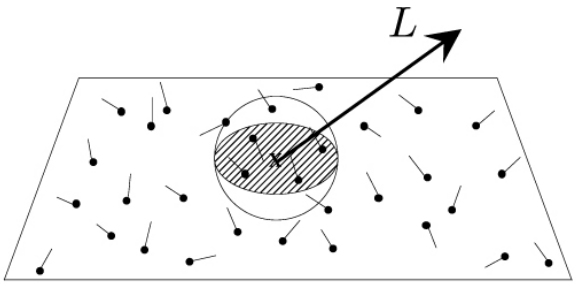

Figure 6: *For the estimation of the surface point's local irradiance a hemisphere that is positioned over the surface point x is expanded until it holds n photons. The surface area that the hemisphere covers approximates the average of photon reflections.*

structure. The best structure to store a photon map is a balanced k-dimensional tree, which enables fast neighborhood searches, is compact and efficient. One photon can be stored in only 20 bytes.

4.3.3 Rendering Of The Scene

Out of these two photon maps we create a lighted scene, by running through the second pass, the rendering pass:
For each pixel of the projection plane a form of Monte Carlo ray tracing follows the path from the viewer through the plane right into the scene. At the first intersection point of a ray and a surface the radiance can be estimated by the rendering equation (See chapter 3.1.1). By using the information stored in the photon maps, the results of a BRDF and the incoming light, we are able to speed up the process.

The rendering equation's integral, here denoted L_r, has to be split up into four components regarding the different light sources. $L_{i,l}$ is the light from caustics, $L_{i,c}$ the light from indirect soft illumination, and $L_{i,d}$ the diffuse reflected light. f was split into a diffuse $f_{r,d}$ and a specular part, $f_{r,s}$. All parameters have been disregarded due to clarity:

$$\begin{aligned} L_r &= \int_\Omega f_r L_{i,l}(\vec{w}' \cdot \vec{n})\,\mathrm{d}\vec{w}' \\ &+ \int_\Omega f_{r,s}(L_{i,c} + L_{i,d}(\vec{w}' \cdot \vec{n})\,\mathrm{d}\vec{w}' \\ &+ \int_\Omega f_{r,d} L_{i,c}(\vec{w}' \cdot \vec{n})\,\mathrm{d}\vec{w}' \\ &+ \int_\Omega f_{r,d} L_{,d}(\vec{w}' \cdot \vec{n})\,\mathrm{d}\vec{w}' \end{aligned} \tag{1}$$

where $f_r = f_{r,s} + f_{r,d}$ and $L_i = L_{i,l} + L_{i,c} + L_{i,d}$

The first integral represents direct illumination, the second specular reflection, the third is responsible for the generation of caustics and the last produces soft indirect illumination. All these components are approximated through the photon maps. The result of the equation is the light radiating from a surface.

$$L_r(x,\vec{w}) \approx \sum_{p=1}^{n} f_r(x,\vec{w}',\vec{w}) \frac{\Delta\Phi_p(x,\vec{w}'_p)}{\pi r^2} \tag{2}$$

Equation 2 estimates the radiance at a point x on a surface. It includes the photon map's information into the rendering equation.

Figure 7: (image courtesy of Henrik Wann Jensen)
An image rendered in 1996 with the then new photon mapping technique by Henrik Wann Jensen.
Rendering time for a resolution of 1280x960 was 56 minutes.

For that purpose imagine a hemisphere (c.f. image 6) over x expanding to the radius r until it holds n photons p. $\Delta\Phi_p$ is the energy radiated from each photon p in direction $\vec{w}'_p$. f_r is the BRDF.

On some occasions the result may be a bit blurry, a weighted confilter can be used to regain accuracy. The weight represents the distance between the intersection point and a neighborhood photon. An example result of Henrik Wann Jensens photon mapping is shown in picture 7.

Photon mapping has all advantages of bidirectional ray tracing; it is able to to simulate all possible light paths and even render caustics. No aliasing problems arise as would for instance through texture mapping or working with graphic primitives. Monte Carlo methods cause high-frequency grain noise, but this can prevented through the con filter.
As long as the surfaces are locally flat, the equation to estimate radiance delivers good results.

5 Real-time and interactive caustic rendering

Since it is sufficient for most interactive applications to render the scene with considerable simplifications to the global illumination model rather than attempting to achieve physical accuracy through costly GPU-based ray tracing, several methods for rendering plausible results through approximation have been developed.
As stated earlier in this chapter, most approaches for the efficient rendering of caustics operate in two passes; In the first pass, particles or rays are emitted from the light source into the scene, in the second pass the occurrences of the particles in the scene are counted for each position as seen from the eye. When attempting to render in real-time, most approaches opt for passing the majority of the calculation load of one or preferably both of the passes to the GPU in order to keep the CPU available for other tasks.

The calculation of the position and intensity of caustics is on principle a question of visibility, that is, the number of times a point can be reached from the light source. A basic algorithm for determining the visibility of a surface point from the light source is

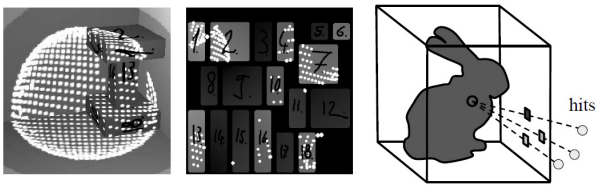

Figure 8: (image courtesy of Szirmay-Kalos et al.)
A distinction can be made among several possibilities of storing the locations of the photon hits. Among these are image space methods (illustrated on the left) in which the photon locations are stored according to their pixel coordinates, whereas the middle image illustrates texture space techniques that store the photon hits' texel coordinates. Cube map texel directions are used in the approaches that employ cube map space methods (on the right).

shadow mapping as proposed by Lance Williams in 1988 [Williams 1988], however, this technique produces only a binary result for the visibility and can thus not be employed directly for caustic generation. Several approaches ([Wyman 2007], [Shah et al. 2007]) are predicated on the principle of shadow mapping and have augmented the shadow mapping technique to not only determining whether a point is visible, but instead counting the numbers of rays reaching the point, thus being able to render caustics and shadows in a single step.

5.1 Calculating the Photon Buffer

When the photon buffers that are used for the computation of the light distribution in the scene in the second rendering step are calculated, it must be taken into consideration that the final hit of a photon on the scene geometry should not only affect that very surface point, but should also be distributed to a certain region also the surface neighborhood of the point of impact. This distribution of photon energy may cause light leaks and even though this artifact cannot be completely suppressed, the unwelcome occurrences can be mitigated if the individual spreading among the neighboring particles also depends upon the visibility of the particle from the caustic generating object.
Many approaches have been proposed for both the determination of the photon hits as well as the calculation of how the photon power will be distributed among the affected surface points. We will categorize these techniques by the selection of the representation of the photon hits' locations. Therefore, we will first list the techniques according to their choice of representation (see image 8) and then give an overview over the existing approaches to optimize the calculation of the photon's influence on its neighboring regions.

5.1.1 3D Grid

The most obvious solution to the photon storage problem is to save the photon hits regardless of the surface they intersected with in a 3-dimensional grid. Since this method is prone to produce light leaks, storing the surface normal alongside with each photon hit can greatly help to reduce artifacts; with this enhancement, for the determination of the local irradiance, only the hits that have similar normal vectors as the respective surface point are taken into account. The stored surface normal permits also the negligence of the photons that arrive from the back face of the surface.
This storage form has been employed by Purcell et al. [Purcell et al. 2003] who developed algorithms for the determination of the nearest photons in the photon grid, thus allowing the estimation of the irradiance corresponding to the neighboring photon hits at any surface point of the scene geometry.

5.1.2 Screen- or Image-Space

In image-space techniques, the specific location of any point of the scene geometry can be indicated by the two-dimensional coordinates of the point when projected onto the view plane as seen from the camera, and the depth from the image plane, hence the term "'screen space"'. Several techniques use this principle to store the photon hit locations. A huge advantage of image-space approaches is that the photons can be splatted onto the surface of the non-specular caustic recipient directly without further calculations. The downside is, however, that the results of image-space techniques are susceptible to light leaks and the BRDF for the photon hit's location is hard to determine. A further limitation of these approaches is that caustic appearances can only be rendered in such cases when not only the caustics, but also the caustic generating object in the scene is visible from the viewpoint.

Augmenting Shadow Mapping

The image-space caustics algorithm proposed by Wyman [Wyman 2007] is based on photon mapping [Jensen 2001], which natually renders in two passes, the first pass determining the position of photons emanating from the light source, and the second pass joining adjacent photons from the first pass to calculate the point's actual color.
As with all algorithms, a decisive factor for the speed is the choice of photon storage method. Most photon mapping approaches opt for balanced kd-trees, the downside of which is that their generation is at best achieved at a runtime of $O(NlogN)$. To bypass this inconvenience, a third rendering pass is added. This pass runs through all points visible in the shadow map and generates a caustic map which stores the number of photons reaching the respective point in the shadow map.
For the determination of the photon's location, the scene is rendered from the light's origin using an approximate refraction algorithm that supports up to two refractions along the light ray's path through the geometry. The algorithm computes most of the necessary calculation steps on the GPU.
Wyman's [Wyman 2007] trade-off between the limited amount of rendering time and a maximum of realism employs a simplification of permitting a maximum of two refractions of the light rays in the scene. In spite of this limitation, the algorithm achieves satisfactory results for real-time applications.
This approach follows the observation that the crucial point in calculating light passing through two refractive interfaces is the calculation of the location of the second refraction: the first ray-object intersection can be calculated through standard rasterization and a pixel shader can be applied to find the direction of the refraction.

Determining The Distance

The main challenge lies in the determination of the distance between the first and the second refraction point. The direction of the refracted ray is, of course, established by the ratio of the densities of the environment and the refractor. The distance $\tilde{d}_1$ is, according to this circumstance, interpolated between the distances to the second refractive surface in two extreme cases of the proportion of the

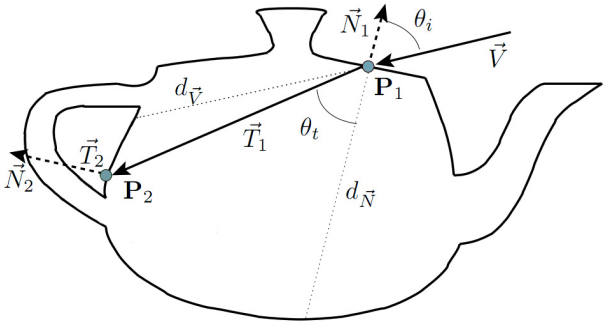

Figure 9: (image courtesy of Chris Wyman)
The first ray-object intersection point P_1 and the surface normal N_1 at point P_1 can be determined by the rasterizer, the refraction that occurs at this point is implemented employing pixel shaders which compute the transmitted vector T_1 according to Snell's law.
The second intersection point P_2 between the refracted ray and the geometry is calculated using the approximated distance $P_1 + d_1T_1$, the refraction itself can again be computed employing pixel shaders to determine the second transmitted vector T_2. Finally, the twice refracted ray $P_2 + d_2T_2$ is intersected with the background geometry to identify P_3.

refractive indices, the first of which being the distance $d_{\vec{V}}$ in direction of the refracted ray in case the indices of refraction are equal, the second distance $d_{\vec{N}}$ following the negative surface normal $-N_1$ which is the limit of the refracted direction for high refractor and low environment density. (See figure 9) This method for the distance calculation is evidently not physically correct since the topology of the second surface is completely disregarded, however, the value is easily calculated and the results are sufficiently plausible.
Arising from the inaccuracy of the distance value $\tilde{d}_1$, it is unlikely that the thus computed point P_2 actually lies on the surface of the second refractor. This necessitates a precalculation of possible back-facing surface-normals; the appropriate surface-normal is then selected for point P_2 through projection.
The second refraction can then, in turn, be computed through pixel shaders. For the intersection of the then twice refracted ray with the background geometry, which also is a challenging task, several existing techniques have been employed to achieve adequate results. The output location P_3 of the refracted ray is stored in the photon buffer. If the algorithm should support both reflection and refraction corresponding to the Fresnel effect, two separate photon buffers must be constructed.
In the course of this rendering pass, additional information might be obtained in separate buffers to produce more complex effects (e.g. colored glass or caustics on non-diffuse surfaces).
In the second rendering pass, the photons that hit the surface indirectly, that is, after one or more reflection or refraction, are stored in the caustics map.
Every photon in the buffer storing the photon locations is treated as a point primitive. This poses the problem that the photon's energy is accumulated in a single point rather than spreading some of its energy over its adjacencies. This problem becomes even more prominent when a smaller number of photons is used because more energy is aggregated in each individual photon.

5.1.3 Texture Space

The basic approach to texture space methods is that a photon hit's resulting reflected radiance is composed from the local BRDF multiplied by the photon's intensity. The photon hit's depiction should hence feature both the surface point and its BRDF; these may be identified by the pair of texture coordinates of the photon's intersection with the scene geometry. Following these considerations, a single entry in the photon hit storage structure consists of the two texture coordinates of the referenced photon hit and the luminous intensity carried by the photon, which is calculated from the illuminant's intensity and the entrance angle. Light leaks are an issue with this approach as well, since the adjoining points in texture space might comprise points that are actually occluded from the viewing direction.

5.1.4 Ray Space

A method that will theoretically provide per-pixel accuracy and can render without light-leaks is to store the light ray after the last permitted reflection or refraction rather than the surface intersection point. However, for the final projection of the caustics, the hit of the ray with the scene geometry has to be determined which is not trivial and requires the trace of many photons unless some filtering is applied. For simplification, some approaches therefore disregard visibility issues.

Warped Volumes For 3D Caustics

A recent approach proposed by Manfred Ernst et al. [Ernst et al. 2005] that uses a ray space storage approach considers caustics as the intersection planes of warped volumes with receiver surfaces. The nature of this approach also supports the rendering of volumetric caustics.
The basic steps of this technique will be described in the following passages.
In contrast to previous algorithms using caustic volumes, which assumed that caustic volumes could be described through prisms, Ernst's approach chooses not to adapt this simplification. Instead, each side of the caustic volumes utilized in this method is represented by the specular triangle's vertices and their corresponding refractive or reflective vectors. In the right part of figure 10, the corner points of the volume's specular triangle are depicted by the vertices v_0 and v_1, their respective vectors are r_0 and r_1.
Each side surface of the caustic volume that is defined by two points from the specular triangle and two points of the caustic triangle forms a bilinear patch which is a quadrilateral formed by four vertices in three dimensional space. The linear interpolation of the vertices' positions defines the surface of the patch, whereby bilinear interpolation is a generalization of linear interpolation.
For the calculation of a point p inside the caustic volume, a caustic triangle that contains the point p must be determined. On principal, it is irrelevant which plane is chosen for the caustic triangle as long as the triangle contains the point p. However, if an arbitrary plane is chosen and intersected with the bilinear patch, the resulting cut surface is a quadratic curve with curved edges. To prevent this, the normal of the plane for the caustic triangle is set equal to the normal of the caustic triangle, thus ensuring that the caustic triangle containing point p will have straight edges.

Point-In-Volume-Testing

The most performance-intensive part of the warped volumes method is the testing whether each pixel inside the caustic volume's bounding volume lies inside the caustic volume.

In the basic method of determining whether a point p is contained by a volume V, the coordinates of p are transformed to a point p' that lies in the coordinate system of volume V. As described

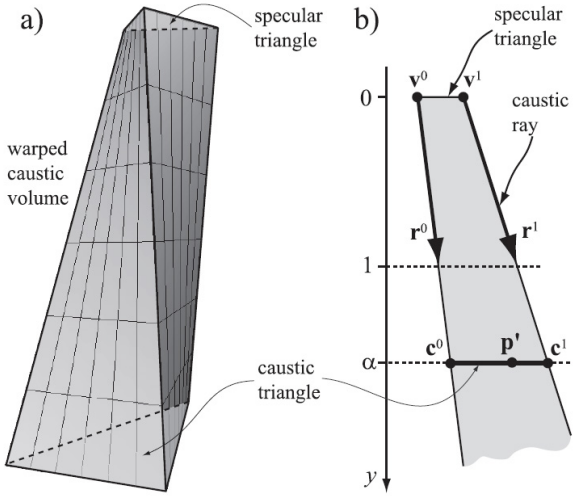

Figure 10: (image courtesy of Ernst et al.)
Figure a) displays an example of the caustic volumes employed in this method.
The caustic volumes are stored in a 2-dimensional coordinate system as depicted in figure b)

above, an intersection plane containing the transformed point p' is constructed in the volume's coordinate system, resulting in the calculation of the intersection points c_0, c_1 and c_2 of the intersection triangle of the plane and the caustic volume. Checking whether p' is inside the intersection triangle is trivial and can be done employing any point-in-triangle test that provides for sufficient performance.

5.1.5 Cube Map Space

When rays leaving the caustics generating object are traced, a distance map is employed whose coordinate system can be directly adapted to represent the photon hits' locations. In this approach, a photon hit is therefore specified by the direction and the distance from the referenced texel in the cube map. This reference is used as the projection center when a neighborhood for filtering is specified by the surface points that are projected onto adjacent texels who also feature similar distances according to the distance map.
Light leaks are not an issue in this approach which is related to shadow testing.

5.1.6 Shadow Map Space

In shadow map space, each photon hit location is specified by the direction of the ray emanating from the light source that hit the surface point and the calculated depth, the illuminant lying in the origin of the coordinate system. Since rendering from the light source is needed in many rendering techniques anyway, this method of photon hit identification can easily be integrated in to the rendering pipeline. However, light leaks may cause artifacts in this storage approach as well and as in the image-space approaches, it is not possible to simulate caustics that originate outside of the illuminant's frustum.

An algorithm by Musawir A. Shah et al. [Shah et al. 2007] that stores photon hits in shadow map space and that resembles shadow mapping closely in general will be described in the next passages.

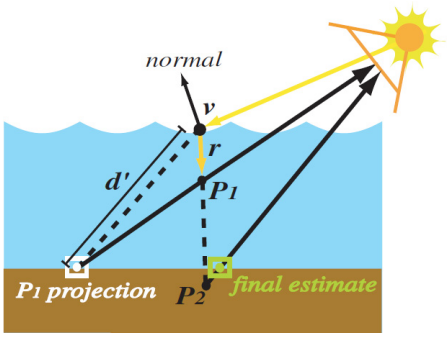

Figure 11: (image courtesy of Shah et al.)
The intersection is estimated through the initial point of on the ray, v, a normalized vector $\vec{r}$ pointing in the direction of the refraction and the previously generated texture 'positionTexture'

in a first step, the receiver geometry is rendered into a 3-dimensional texture labeled 'position texture' is generated. The position texture stores the 3D-world coordinates of each pixel and is required in the next step for the performance of ray-intersection-estimations.
Subsequently, the 3D-world coordinates and the surface normals are rendered for each pixel as seen from the light source. Thus, textures are generated which represent the refractive object during the further progress of the algorithm. The texture is basically a mapping of each vertex in a grid of vertices to a pixel on the texture.
The crucial part of the algorithm lies in the assembly of the caustic map. The vertex grid texture is rendered from the light's position in a vertex shader, the distance of the resulting vertices from the light has to be estimated and the vertices are placed along the light direction according to the distance information. To allow multiple vertices at a single location of the receiving surface (thus, generating caustics), each vertex contributes its respective intensity through additive alpha blending.
For the accomplishment of the non-trivial task of calculating the intersection point between light ray and scene geometry, Shah et al. propose an iterative technique that renders at less cost than conventional intersection calculation that cannot perform in real-time.
Any position along the refracted light ray can be defined as

$$P = v + d * \vec{r} \tag{3}$$

d being the distance from the current vertex v and $\vec{r}$ a normalized vector pointing in the direction of the refracted ray. (These parameters are illustrated in figure 11)
Now according to this consideration, an initial value is assigned to the distance d, thus computing a position P_1. In the first iteration, the initial value will be set to 1, for all other iterations, the distance between v and the last estimated intersection point will be used as new estimation for the value of d.

$$P_1 = v + 1 * \vec{r} \tag{4}$$

When P_1 is projected onto the light's view space, the previously computed position texture map is used to lookup the position of the point $P_1 projection$ on the light's view space. The distance d' between the thus computed point $P_1 projection$ and the vertex v is appointed as an estimation for the calculation of another point P_2 which is subsequently projected into the light's view space as well, finally calculating the estimated intersection point.

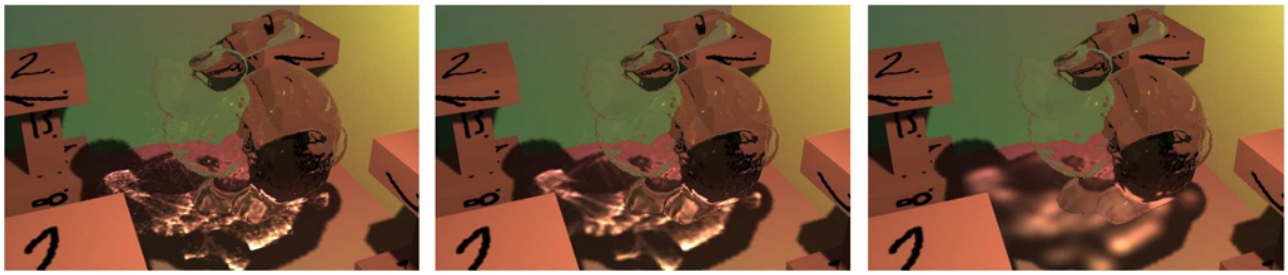

Figure 12: (image courtesy of Szirmay-Kalos et al.)
A bad choice of splatting filter size will usually result in the rendering of unsatisfactory caustic depictions. Whereas the image on the left was generated using a too small splatting filter and the right one with a too large filter, the filter for the middle image had to be adjusted manually to optimize the results.

5.2 Photon Hit Filtering

Regardless of the storage structure of the constructed photon maps, the photon hits are now represented as a discrete structure, which is insufficient for the final rendering, thus filtering is required to convert this discrete texture into a continuous pattern which can then be applied onto the receiver surfaces.
In the original photon mapping technique [Jensen 1996], this filtering was performed during the last rendering pass when rendering from the view point. The filtering method of the photon mapping algorithm was explained in image 6.
However, the filtering can be significantly simplified if it is not implemented in camera space but instead in the coordinate system where the photons are stored. This observation suggests the introduction of a filtering pass that is executed between photon tracing and final gathering, which generates a texture, the caustic intensity map. During the third rendering pass, the caustic intensity map is then projected onto the affected surfaces.

5.2.1 Photon Splatting

Passing through the previously generated photon map, the photons that have been reflected or refracted are selected and rendered into the caustic map using additive alpha-blending to count the number of photons per pixel.
Especially at a smaller photon count, it is advantageous for the image quality to treat each photon as a splat and accordingly distribute the photon's energy over multiple texels, for instance with Gaussian weights. The weight of the Gaussians must sum to 1 in order to preserve the energy. The smaller the splats are, (and thus, the sharper the caustics) the more splats must be calculated to obtain a high-resolution caustic-map with a minimum of noise. It depends on the application whether it is sufficient to generate blurry caustics with a smaller number of large splats or very crisp caustics at a higher resolution of the caustic map.
Essentially, photon splatting is a filtering pass during which the filter radius is invariable, but more photons are averaged in high-density regions. As figure 12 illustrates notably, the choice of a splat size is not trivial since a choice of one uniform size does not consider the irregular distribution of the photons on the surface inherent to the caustic pattern. Furthermore, the orientation of the receiver surface is not incorporated in the photon splatting algorithm. This may cause unrealistic results in extreme cases.
During the execution of photon splatting on the GPU, a quadriliterals is employed for each photon in the photon map. The rectangle is rasterized and processed by the fragment shader, subsequently a filter texture is assigned to the rectangle and with the use of additive alpha-blending, the contribution of all splats to the energy of one single surface point is calculated.
The splatting procedure can be performed in screen space, but also in any other of the above described photon storage spaces.

5.2.2 Caustic Quads Or Triangles

The caustic triangles algorithm follows the idea of beam tracing [Watt 1990] which is based on the proposition that adjacent rays emanating from an illuminant form a beam. Every single beam is defined by its corner rays and usually has a triangular profile, thus a beam-surface-intersection can also be approximated to a triangle. This idea is pursued in this technique where three neighboring photon hits are assumed to form a caustic triangle, which are additively blended together. A filter should be applied before projection to avoid sharp triangle boundaries.
An advantage of this approach over photon splatting methods is that unlike photon splatting, this technique always provides continuous textures.

In caustic volume approaches like in the method by Ernst et al. [Ernst et al. 2005], it is absolutely necessary for obtaining satisfactory results that the caustic intensities inside the caustic triangles are interpolated so as to prevent visible block artifacts and guarantee smooth caustic presentation. A comparison of a scene with and without triangular interpolation can be found in figure 13.

5.3 Projecting Caustics And Composing The Scene

Projection of the caustic pattern onto the receiver surfaces may an issue before the rendering of the scene depending on the choice of photon storage in the last rendering step.
If the photons are stored in screen space, obviously no projection is needed.
The projection of the caustic patterns from texture space is performed at one go in a normal texturing step.
In shadow map space or cube map space, an algorithm similar to shadow mapping can be applied for the projection; during the gathering pass each point is transformed to the cube map coordinate system. A comparison is made between the distance value stored in each of the cube map texels and the distance of the currently processed point. For similar values, it is assumed that the point is visible from the caustics generating object. This method helps minimizing light leaks artifacts on the cost of one extra texture lookup in the shader. For ray space techniques, the individual photon-surface-hits are projected onto the receiving surface through an auxiliary

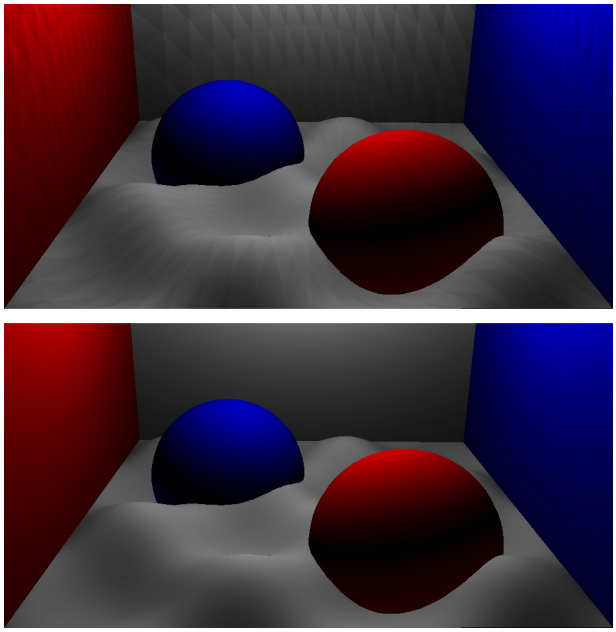

Figure 13: (image courtesy of Ernst et al.)
This is a characteristic example of how the interpolation of the caustic intensities inside the caustic triangles improves the resulting image. The top image and the bottom image display the same scene, but the caustic effects of the top image have not been interpolated, which evidently results in a discernible and undesirable triangular pattern.

function in image-space. For each photon, a line is drawn in screen-space, after which it is examined whether the current pixel is similar to the point depicted by the line. If this is the case, an intersection can be located by rendering the ray into a separate row of a texture, where the first hit is subsequently calculated.

Finally, the scene is rendered using one of the existent rendering techniques, adding the caustic contribution to the results of the diffuse surfaces.
Wyman's approach only updates the caustic map whenever either the light or the scene geometry changes. Since both the first and third pass of this approach use the reflection and refraction algorithm, this method must be fast enough to run at least twice per frame to maintain a fluent handling of the interactive application. By choosing the resolution for photon buffer, caustic map and final screen size, a trade-off between cost and quality can be individually adjusted.

5.4 Simplified Caustic Generating Algorithms

Several algorithms that try to achieve visual attractiveness or simulate caustic effects without regard to the physical correctness of the caustic calculation have been proposed over the course of years of caustics research. Perhaps most notably the simulation of underwater caustics has received particular attention due to the large visual enhancement and increased degree of realism that the integration of caustic simulation in underwater scenes brings about.
In 1996, Jos Stam [Stam 1996] precomputed random caustic textures and projected them onto the receiving surfaces in the scene. This technique renders caustics very quickly because little computation is involved and physical correctness is disregarded. However, this method is only sufficient in cases where the caustics cannot be perceived very clearly and the scene geometry is simple since both water surface and receiver geometry do not influence the caustic pattern.
The same limitation applies to Trendall and Stewart's caustic algorithm [Trendall and Stewart 2000] which computes refractive caustics through integration of the caustic intensities numerically on a flat receiver surface. Like Stam's algorithm, this technique is not applicable to arbitrary receiver surface and moreover cannot handle shadows.
The interactive caustics rendering technique by Wand and Strasser [Wand and Straßer 2003] samples points on the caustic-forming object explicitly. The caustic intensity contribution of each of the points on the caustic-generating object are taken into account without testing for visibility. This intensity of these sample points is composited by the entries in a cube map which embody the visible light sources in the reflection direction.
This method supports both reflective and refractive caustics. The explicit sampling poses a disadvantage of this algorithm: the scalability is restricted since the runtime is directly proportional to the sample points count.

6 God Rays

For the accomplishment of physically correct rendering results of god rays, the process of the scattering of light would theoretically have to be reproduced; while travelling through the scene, on the impact on a particle, a light ray gets scattered, thus creating more rays, which in turn get scattered on their next impact, and so further. This recursion would be, with current computational capabilities, impossible to compute which is why, once again, good approximations of the complex physical process will bring more effective results.

The air is always full of small particles; in nature there are even special weather phenomenons which accumulate the air with solids or fluids, such as morning fog and sandstorms. When light hits these particles, the path of the light, normally not perceptible, is rendered visible: god rays occur. Differently from the previously discussed caustics depiction which produce a two-dimensional pattern on a surface, the simulation of god rays always has a 3-dimensional character.
Caustics could actually be regarded as the final intersections of multiple (god) rays with a surface, a consideration which is taken in account by some god ray rendering techniques that produce caustics and god rays simultaneously. From this point of view, god rays can be looked upon as a sort of three-dimensional caustics.

6.1 Volume Photon Mapping

With the paper *Efficient simulation of light transport in scences with participating media using photon maps* [Jensen and Christensen 1998] photon mapping was extended into 3-dimensional space. This is necessary to trace light traveling through participating media. Volume photon mapping is able to handle lighting effects that other methods have to neglect: Anisotropic scattering, multiple scattering, color bleeding, god rays and light radiating through nonhomogeneous media are easily obtained.

Bidirectional Monte Carlo ray tracing with photon maps is used to generate the scene's illumination, and it is the base to create illumination in participating media.

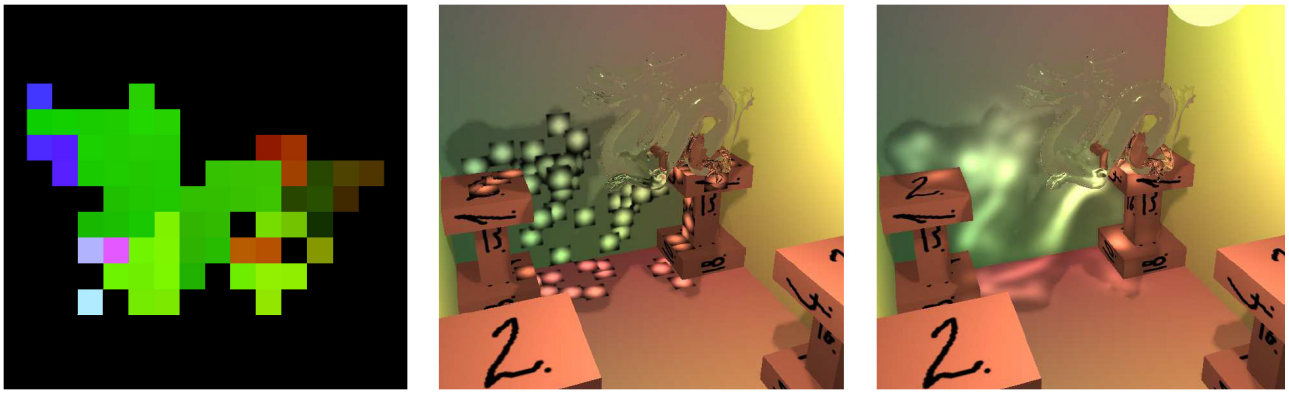

Figure 14: (image courtesy of Szirmay-Kalos et al.)
The left figure depicts a low-res photon map of a 3-dimensional scene, on the middle, the same scene is seen already featuring the photon hits, but yet without blending. On the right side, the final scene with blending, displaying beautiful caustic effects.

To approximate the light's path through participating media, the underlying physics have to be examined:

6.1.1 Light Transport In Participating Media

The following equation 5 describes the change in radiance at the point x in the direction $\vec{w}$. $L(x,\vec{w})$ is composed of emission, in-scattering, absorption and out-scattering of light. Some rearranging results in the following formula:

$$\begin{aligned} L(x,\vec{w}) &= \int_{x_0}^{x} \tau(x',x)\alpha(x')L_e(x',\vec{w})dx' \\ &+ \int_{x_0}^{x} \tau(x',x)\sigma(x') \int_{\Omega} f(x',\vec{w}',\vec{w})L(x',\vec{w})dw'dx' \\ &+ \tau(x_0,x)L(x_0,\vec{w}) \end{aligned} \tag{5}$$

with L_e being the emitted radiance.

$$\tau(x',x) = e^{-\int_{x'}^{x} \kappa(\xi)d\xi} \tag{6}$$

τ is the light transfer along x' to x.

$$\kappa(x) = \alpha(x) + \sigma(x) \tag{7}$$

$\kappa(x)$ is composed of the absorption coefficient α and the scattering coefficient σ. If these coefficients are constant throughout the medium, the medium is called homogeneous or uniform.

The light transport in the participating media equation describes a general case, several simplifications can be accomplished for homogeneous media or isotropic scattering.

6.1.2 The Volume Photon Map

In addition to the caustic map and the global map, a volume map is built. This map serves as storage of the photons which interact with the participating media, and it is only used to create illumination in the participating media. The density of photons in space can serve as a measurement of the intensity of the illumination. Furthermore, we save the incoming direction of each photon for the computation of isotropic scattering.

In this approach, the direct illumination is computed through ray tracing although volume photon mapping might achieve the same results. The reason is that ray tracing causes lesser computational cost in exchange for speed. Hence the volume photon map is only used for indirect illumination, i.e. only photons reflected of a surface or photons that are scattered at least once, are stored.

When light travels through a medium it can either collide with a particle or pass straight through the medium without collision. The probability of collision on position x is described by the cumulative probability density function:

$$F(x) = 1 - \tau(x_s,x) = 1 - e^{-\int_{x_s}^{x} \kappa(\xi)d\xi} \tag{8}$$

The position where the light ray enters the medium is x_s, $\tau(x_s,x)$ is calculated with ray marching.

If the light hits a particle it can be either absorbed or scattered. The Russian roulette method chooses between either possibilities, the probability of scattering is defined by $\sigma(x)/\kappa(x)$.
A modification of the phase function eventually computes the new direction of the light ray.

6.1.3 Estimating The Radiance

The volume map's information is necessary to compute the in-scattered radiance $L_i(x,\vec{w})$ at point x in direction $\vec{w}$.

Just like for the estimation of the photon mapping's radiance 4.3 we assume a hemisphere over the point x, see figure 15. Out of all photons gathered in this sphere the in-scattered radiance is computed with the following formula.

$$L_i(x,\vec{w}) = L_{i,d}(x,\vec{w}) + \frac{\sigma(x)}{\kappa(x)} L_{i,i}(x,\vec{w}) \tag{9}$$

where $L_{i,d}$ denotes the direct illumination, and $L_{i,i}$ the indirect illumination.

6.1.4 Rendering The Scene

To render the scene the same method as shown at 4.3 is used with a little modification. For a proper adaption those light rays have to be taken into consideration that went straight through the participating

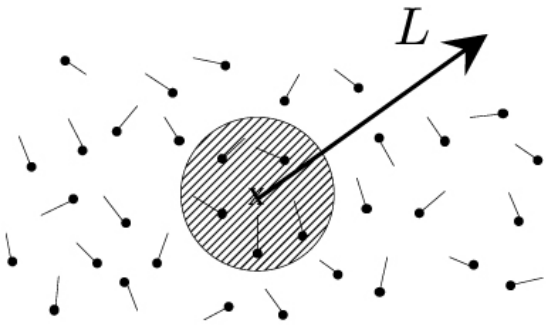

Figure 15:
Photon hemisphere in a volume:
Over a point x a hemisphere is presumed. The irradiance at point x is calculated out of all photons in this hemisphere.

media while calculating the global illumination.
The radiance is calculated iteratively along a ray with a ray marching implementation. In each step, the radiance of the ray's previous section diminishes. This is accomplished with the approximation stated in equation 12.

$$L(x_k, \vec{\omega}) = \alpha(x_k) L_e(x_k, \vec{\omega}) \Delta x_k \tag{10}$$
$$+\sigma(x_k) L_i(x_k, \vec{\omega}) \Delta x_k \Delta \tag{11}$$
$$+e^{-\kappa(x_k)\Delta x_k} L(x_k - 1, \vec{\omega}) \tag{12}$$

where $\Delta x_k = |x_k - x_k - 1|$ is the step size.

Examples of images rendered with volume photon mapping are presented in figure 16 and 17.

Figure 16:
This cloud is modeled as a non-homogeneous, anisotropic medium. In the upper image, the cloud is rendered with direct illumination and single scattering, below with global illumination and multiple scattering.

6.2 Line Space Gathering

The paper *Line Space Gathering for Single Scattering in Large Scenes* [Sun et al. 2010] refrained from the idea of lighting simulation through photons. It examines the light ray as a whole instead. The light rays near the viewing ray are gathered and their illumination is summed up.
The simplifications that are conduct in the presented technique are single scattering and the assumption of homogeneous media. It is, however, possible to display reflective and refractive objects, and different light interactions such as occlusion and multiple specular bounces. The only other method which is able to calculate these light effects is volume photon mapping, a technique with considerably higher computational and memory costs. The algorithm introduced in [Sun et al. 2010] cuts those costs significantly. For a direct comparison between photon mapping and line space gathering see figure 18.

Figure 17:
To render the god rays in this underwater scene as many as three million photons were used.

6.2.1 Radiance Estimation

The radiance calculation is based on equation 5 from 6.1, where the change of radiance at a point x in direction $\vec{w}$ is computed. But instead of gathering all the in-scattering light of photons within a square (see equation 9) of sample points along $\vec{w}$, the radiance is approximated through beam gathering, equation 13. $L(x, x_0)$ is the sum of the illuminations L_r, of the lighting rays which pass through

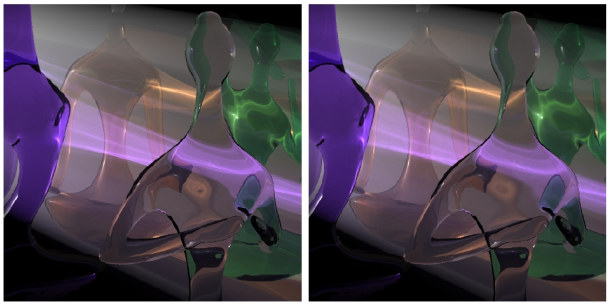

Figure 18:
The scene was rendered twice on the same computer (dual Intel Xeon X5470 3.33 GHz quad-core CPUs and a NVIDIA GeForce GTX 280 graphics card). The left image used line space gathering, it was calculated in 26 min on the CPU, 3.7 min on the GPU and took 404MB (CPU), ¡800MB(GPU) memory. The right image was done with volume photon mapping in 729 min (CPU) and at 7.1 GB (CPU) memory costs. The image had a resolution of 512 x 512 with 2x2 supersampling.

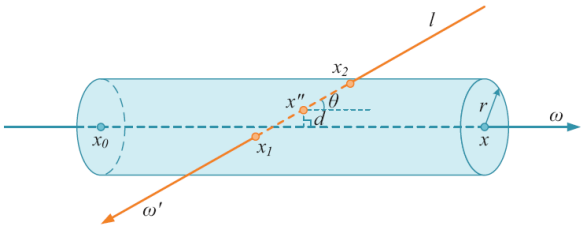

Figure 19:
Passing of a lighting ray through a space near a viewing ray: The lighting ray l passes through a cylinder with the radius r that is assumed around the viewing ray. x'' is the nearest point on l toward the viewing ray, passing at a distance of d.

a space along the viewing ray, defined as a cylinder with radius r, see figure 19.

$$L(x,x_0) \approx \sum_{l=1}^{n} L_r(x,x_0,l) \tag{13}$$

L_r is estimated by assuming the flux of the ray to be constant and the media to be homogeneous.

$$L_r(x,x_0,l) \approx \omega(x'',x,\vec{\omega}',\vec{\omega})\phi_l(x'',\vec{\omega}')\frac{(r^2-d^2)^{1/2}}{sin(\theta)} \tag{14}$$

$$\omega(x'',x,\vec{\omega}',\vec{\omega}) = \frac{\tau(x',x)\sigma(x')f(x',\vec{\omega}',\vec{\omega})}{\pi r^2 \|x-x_0\|} \tag{15}$$

The vertices x and x_0 describe a line segment on the viewing ray. x'' is a point on the lighting ray l with minimal distance d to this line segment. θ is the angle between these two rays. $\phi_l(x'',\vec{w}')$ describes the flux at x'' flowing in direction $\vec{w}'$ along a lighting ray l. For a reference see figure 19.

As illustrated in figure 20, equation 13 enables direct illumination transfer between lighting rays and viewing rays completely without the representation of light particles or media particles.

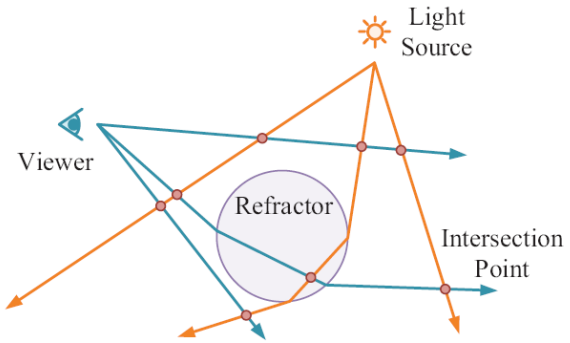

Figure 20:
First lighting and viewing rays are generated. For the generation of the final image the radiance of lighting rays intersecting or nearing a viewing ray are summed up, resulting in the final radiance value of the corresponding pixel.

6.2.2 Gathering Lines

The main challenge of this method is the gathering of lines that nearly intersect. Since there is no efficient spatial hierarchy in 3D space to do this job, the line gathering will be performed in a 6-dimensional parametric space of Plücker coordinates and coefficients.

The procedure will be described in the following passage:
Initially, the scene geometry's spatial hierarchy is constructed as a kd-tree. Subsequently, the lighting rays are computed, a step which includes their refractions, reflections and their radiance.

The next step is responsible for transforming the lighting rays l to Plücker coordinates $\pi(l)$ and the viewing rays l' to Plücker coefficients $\varpi(l')$. In this 6D space every line from 3D space is represented as a point $\pi(l)$ or a hyperplane $\varpi(l')$ which goes through the origin of the 6D space.

Finding the nearest light rays to a viewing ray is equivalent to finding the nearest points to the corresponding hyperplane. Equation 16 defines which points are adjacent, in this case d is the lower bound of the distance between two lines.

$$d \geq \frac{6V_{(a,b,a',b')}}{c^2} = \frac{|\pi(l) \bullet \varpi(l')|}{c^2}, \tag{16}$$

where c is a fixed distance between the lines (a,b) and (a',b') and accordingly l and l'. The correlation between the two equal parts of the equation is that a $\frac{1}{6}$ of the determinant's magnitude of $\pi(l) \bullet \varpi(l')$ is the volume of the tetrahedron hoisted by the two line's vertices, see figure 21.

Out of the points $\pi(l)$ and the distance information a 6-dimensional space hierarchy is constructed as a perfectly balanced octary tree. Finding the nearest lighting rays along a viewing ray then takes only a search through the tree. To get the final radiance along the viewing ray the radiance of these nearing lighting ray segments are summed up.

This method is not fixed to a certain type of light path or volumes of fixed resolution. It can be implemented on the GPU and is therefore much more efficient than volume photon mapping. Various ray tracing techniques can be combined with line space gathering. This technique is able to generate a lot of advanced lighting effects

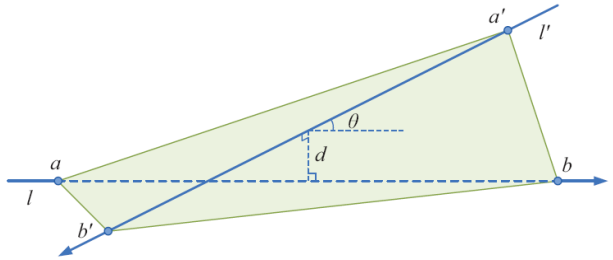

Figure 21:
The 3-dimensional coordinates of the viewing ray l and the lighting ray l' construct tetrahedron. d is the distance between l and l'. This distance is the measurement for the proximity of these two lines.

among others caustics and god rays. Observing the rapid GPU computation developments, [Sun et al. 2010] expects their technique to be capable of achieving real-time implementations in the future.

6.3 Creation Of Atmospheric Lighting Effects

This method [Hura and Hall 2006] was designed specifically for the rendering of god rays caused by clouds. It aims for realistic and highly aesthetic results. Therefore it introduces an algorithm with pseudo-physical faithfulness to simulate a quite authentic light-cloud interaction.

6.3.1 The Cloud Model

For this approach clouds are constructed by a hybrid model of particle systems and metaballs (blobs). Particles are interpreted as points in 3D space with different properties such as size density and color. Since real clouds consist of tiny droplets of water this would be the physically correct simulation of clouds. Metaballs are volumes with deformed boundaries; each ball has a center, a radius and a density. Through intersection with other metaballs, their surfaces deform accordingly to their properties, simulating a natural behavior of clouds.
These two models are combined by using the Metaballs as macrostructure for the particles. This allows us to comfortably model light shafts caused by gaps in the clouds as well as light scattering in participating media.

6.3.2 Illumination Approximation

To calculate the lighting, a variation of the Light Transport Equation is used (equation 17).

$$\frac{\mathrm{d}L(x,\omega)}{\mathrm{d}s} = K_s(x)\int_{4\pi} P(x,\omega,\omega')L(x,w')\mathrm{d}\omega' - K(x)L(x,\omega) \quad (17)$$

There a several new terms in this equation: $K_s(x)$ is the scattering cross section per unit volume. $P(x,\omega,\omega')$ is a phase function describing the probability density of light coming from ω and scattering into direction ω'. To solve the phase function the Henyey-Greenstine method is used 18. $K(x)$ is the extinction coefficient, it is the sum of absorption $K(a)$ and scattering coefficient $K(s)$, describing light attenuation per volume unit.

$$P_{HG}(\phi) = \frac{1}{4\pi}\frac{1-g^2}{(1+g^2-2g\cos\phi)^{\frac{3}{2}}} \quad (18)$$

The LTE is normalized to get a value between 0 and 1. 0 indicates no light and 1 maximal lighting. In [Hura and Hall 2006] four different approaches to solve LTE were discussed: Monte Carlo, discrete ordinations, spherical harmonics, and the finite element.
The conclusion drawn by Hura et al. was that the modified Monte Carlo delivers best results in regard to physical correctness. Monte Carlo shoots a large number of photons from the light source into the scene in random directions, calculating a random sample of the integral domain of the LTE. The Monte Carlo method can cause aliasing if too little samples are used; Henrik Wann Jensen [Jensen and Christensen 1998] showed a way to prevent aliasing.

This method imposes standards on aesthetic and realism, it has high computing costs, but since its an offline algorithm, this factor is not as relevant as in real-time applications.

7 God Rays In Real-Time

Same as in caustics simulation, a minimization of computation cost is imperative to the success of an algorithm.

For the accomplishment of real-time rendering, an optimal physical simulation of the scattering effect immanent to god ray creation is not possible. The physical lighting model has to be simplified and satisfactory and visually appealing results are achieved through simplification, filtering and interpolation.

7.1 A Post-Processing Pipeline To Render Sunlight With Volumetric Light Rays

To render light beams that are caused by a direct glance at the partly occluded sun, Pawel Rohleder and Maciej Jamrozik [Rohleder and Jamrozik 2008] propose a post-processing approach on the GPU. The fundamental principle of their approach is to bloom and blur parts of the scene around the position of the sun to simulate overexposure.

This method requires three render targets and seven post-processing steps to render the final image using simple post-processing operations.
The main render target holds the image of the rendered scene in full resolution. The two other render targets (*Temp*0, *Temp*1) contain the same imagine a smaller size with one-sixteenth resolution of the original image.
The post-processing pipeline is shown in figure 22.

1. In the first step the scene is rendered into the main render target.
2. The main render target is down-sampled into *Temp*0.
3. *Temp*0 is horizontally blurred into *Temp*1 as shown in Microsofts HDRLighting sample, in which Bloom-Effect, Star-Effect and tone-mapping are added.
4. *Temp*1 is Vertically blurred into *Temp*0 again like Microsofts HDRLighting sample.
5. Steps five and six are the main steps of the method.
   In step five a Radial Glow Mask is calculated (see figure 23). The purpose of this mask is the simulation of pixel glowing, at which pixel farther away from the sun exhibit little to no glowing. The mask is applied to the blurred image of *Temp*0 by positioning a greyscale gradient texture on the blurred image at the sun's position. The texture's and *Temp*0's pixels are then multiplied and saved in *Temp*1.

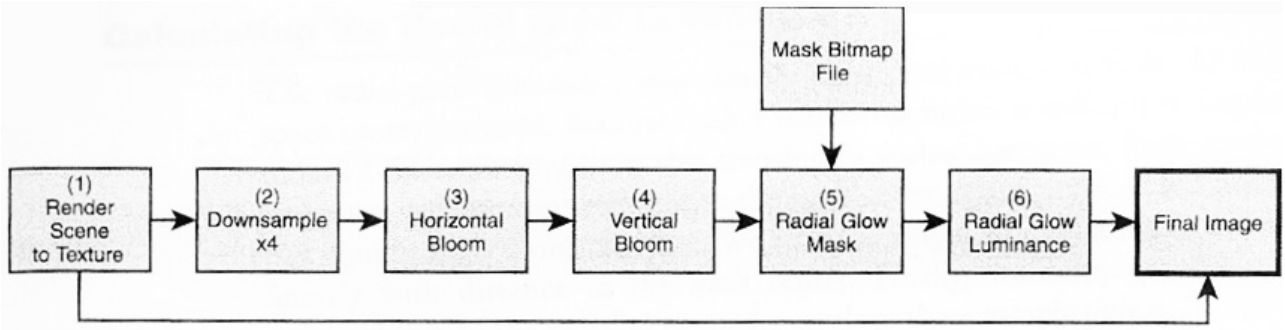

Figure 22:*This image shows the seven steps of the post-processing pipeline introduced at 7.1.*

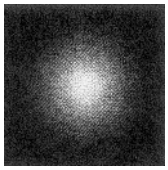

Figure 23:
This is an example grey scale texture used to compute the radial glow mask.

By scaling the texture's size or intensity different appearances of the final effect can be achieved.
The Radial Glow Mask is implemented with a vertex and pixel shader. The source code can be obtained at Appendix A of [Rohleder and Jamrozik 2008].

6. Subsequently we calculate the Radial Glow Illumination from *Temp*1 and save it in *Temp*0. This is done by means of a gather operation. For each pixel, a line connecting it to the sun's screen-space position is computed. Along this line n samples are placed. These samples are weighted according to the sun's distance. The highest value is at the sun's center, it decreases with increasing distance from the sun. The output of this operation is the weighted sum of the radial glow mask's texels at the sample points. This way the pixel intensities are blurred from the sun's center outward.
The source code of the vertex and pixel shader to calculate the Radial Glow Illumination are shown in Appendix B of [Rohleder and Jamrozik 2008].

7. To get the final image *Temp*0 has to be added to the main-render target.

The complexity of the scene has no relevance for computational cost since this method is post-processing. It is also easily integrated in an existing post-processing pipeline. The technique is only able to simulate god rays caused by a direct glance into the sun, i.e. if the illuminant does not lie beyond the image borders.

7.2 Shadows Inside God Rays

[Tóth and Umenhoffer 2009] introduces a method to render god rays in real-time by using shadow maps to detect the occlusion in the light beams and interleaved sampling to reduce the computation cost. The algorithm is limited to single scattering in non-emissive homogeneous media but able to manage solid dynamic occluder objects (figure 24) and dynamic light sources. It is based on classic

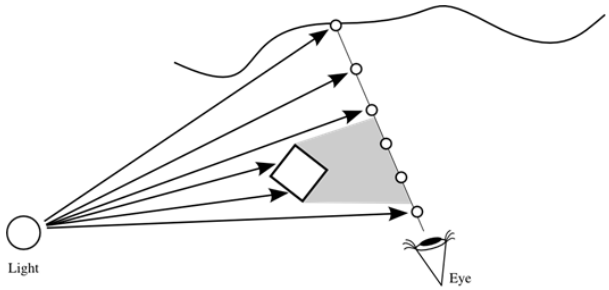

Figure 24:
An object blocks the lighting rays path and causes a shadow inside the god ray.

ray marching with some extensions and implemented in a single fragment shader as a post-process operation on the GPU.

7.2.1 Radiance Estimation

The lighting is calculated with the radiative transport equation, which is based on the render equation(3.1.1) and introduced by [Kajiya 1986]. It computes the change in radiance$L(\vec{x}_s, \vec{\omega})$ along a ray $\vec{x}(s) = \vec{x}_0 + \vec{\omega}s$ connecting a point at a surface and the eye, in which s stands for the ray parameter.
After simplifying the equation by assuming single scattering, it is approximated through a finite Riemann summation, see 19.

$$L(\vec{x}(s), \vec{\omega}) \approx L(\vec{x}(l_n), \vec{\omega})e^{-\tau s} + \sum_{n=0}^{N} L_i(x(\vec{l_n}), o\vec{m}ega)e^{-\tau(s-l_n)}\Delta l \tag{19}$$

$\Delta l = s/N$ is the step size, which is proportional to the length of the ray and inverse proportional to the number of sample points N. τ is the density describing the probability of collision.

The in-scattering light L_i is described in 20.

$$L_i(\vec{x}, \vec{\omega}) = \tau\alpha\frac{\phi}{4\pi d^2}v(\vec{x})e^{\tau d}P(\vec{\omega}_l, \vec{\omega}) \tag{20}$$

ϕ is the power of the light source. α describes the albedo equaling the probability of scattering, $P(\vec{\omega}_l, \vec{\omega})$ is the Phase function, d is the distance between the point and the light source. $v(\vec{x})$ indicates the visibility of the sample point from the light source.

The principles of these calculations are illustrated in figure 25

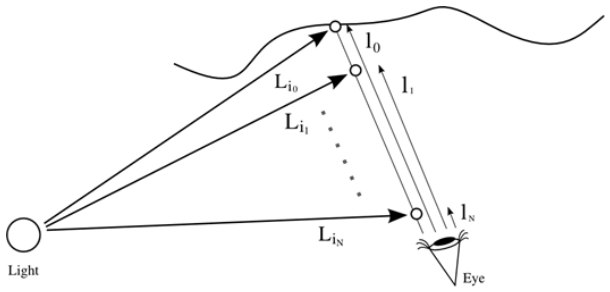

Figure 25:
A viewing ray l_n travels until it hits a surface. Along this ray the values of L_i are calculated iteratively with a certain step size.

The ray marching algorithm approximates equation 19 iteratively. The underlying principle is to determine the radiance at a visible surface point without taking into account the participating media and use it as a boundary condition for the volume radiance. The participating media causes a decrease of the radiance between the surface and the eye. To determine the final value we iterate N times along the ray and calculate the in-scattering light L_i and the absorption factor $e^{-\tau(s-l_n)}$ for each segment. The final illumination is composed by the sum of their products.

The number of steps N along a ray is proportional to the computing time. Too many steps cause slow rendering, however, too few samples cause image artifacts: A compromise must be found.

A good way to solve this problem is to use interleaved sampling for the ray marching algorithm. This is possible because the surface and the scattering are similar in the neighboring pixels.

7.2.2 GPU Implementation

If any occluders lie in the light beam's way, we take them into account by means of ray marching. The visibility factor can be calculated in every step along the ray, using regular shadow maps generated from the light source position.

The final radiance is stored in every pixel of the scattering image.

The following shader code calculates the radiance and the visibility for every line segment.

```
L = L0 * exp(-s* tau);
for{float l= s - dl; l<= 0; l-= dl){
x += viewDir * dl;
float v = shadowMC(shadowMap,x);
float d= length(x);
Lin = exp(- d * tau) * v * Pi/ 4 M_pi/d/d;
Li = Lin * tau *albedo * P(x, viewDir);
}
```

The visibility indicator v is computed through the shadow test function *shadowMC*. The ray marching algorithm can be applied as during post-processing.

8 Underwater Lighting Effects

The realistic rendering of water is an essential part to create realistic images of natural scenes. Due to water's transparent character, the water surface can be perceived from two directions, from above looking down and from below, looking up.

Figure 26: (image courtesy of Jason Slaughter)
This is an exceptionally beautiful example for underwater lighting effects.

In nature, water is never a homogeneous media. It is always mixed with small particles or organisms, causing light to scatter while it travels through. As a result of the presence of these particles, the god ray phenomenon is ubiquitous in underwater scenes. Other important effects which add to a rendered scene's realism are underwater caustics and shadows. In natural environments catacaustics occur when observing the water from above and diacaustics below the water surface.

Underwater szenarios are not only fascinating and visually attractive (see figure 26) but also gain importance in real-time 3-dimensional applications. Real-time underwater lighting effects are particularly difficult to achive because they depend on the light refraction at the animated water surface. This adds another step in computation, making real-time yet harder to accomplish.

In the following several approaches to generate underwater god rays will be discussed.

8.1 Illumination Volumes on GPU

Iwasaki et al. [Iwasaki et al. 2002] introduced a viewpoint dependent technique to render underwater optical effects offline on graphics hardware. The method is able to generate caustics, god rays and shadows of objects within the water volume. Caustics are obtained by a stencil buffer, god rays make use of the hardware's color blending functions and the shadow map technique is used to produce the shadows.
Since caustics and god rays are the result of convergence and divergence of refracted light at the water surface, it is important which wave model was used to generate the surface. In this case a statistical wave model builds the surface.

8.1.1 Previous Work

The basic idea for this method is picked up at previous work by Nishita and Nakamae [Nishita and Nakamae 1994]. In their approach, the water surface is considered a mesh of triangles. Through each triangle a light beam refracts and strikes the ocean floor. These refracted light beams hoist a parallelepiped with the

height of the water's depth, called illumination volume (see figure 27).
Nishita and Nakamae [Nishita and Nakamae 1994] calculate the intensities of the illumination volumes and store them in a accumulation buffer. Since this buffer was not implemented on the GPU, the method was slow, the generation of underwater images took several minutes. By using the GPU [Iwasaki et al. 2002] the process could be speeded up significantly and was able to render in several seconds.

8.1.2 The Method

In this model the light shining on the water surface consists of direct sunlight and skylight. The skylight is considered to be ambient light.

To calculate the light intensity $I_v(\lambda)$ at the viewpoint I_v below the water surface, reaching from a point at the water surface Q, the following equation is used:

$$I_v(\lambda) = I_Q(\lambda)e^{-\alpha_\lambda dist} + \int_0^{dist} I_P(\lambda)e^{-\alpha_\lambda l}\,\mathrm{d}l \qquad (21)$$

$I_v(\lambda)$ depends on the wavelength λ of the light, the distance *dist* between view point I_v and the the surface point Q, a light attenuation coefficient $-\alpha_\lambda$, and the intensity of scattered light I_p at a point between Q and I_v. The integral term describes the intensity of the light scattering along the viewing ray.
This integral term is approximated through the illumination volumes. Since light fades exponentially with the water's depth, the volumes are divided horizontally into subsections (c.f. figure 27). For each of these sections the scattered light at a point I_P^S in the subvolume is calculated with equation 22. A smooth change between sections can be obtained by creating a sufficiently large number of thin subsections near the water surface, where the light intensity is still high. The GPU can take care of blending sections together.

$$I_P^S(\lambda) = (I_{sun}(\lambda)T\frac{a_S}{a_C}P(\lambda,\phi)e^{-\alpha_\lambda d} + I_a)\rho \qquad (22)$$

Here I_{sun} describes the intensity of the sunlight at the waters surface, I_a is the ambient light. T stands for the transmission coefficient from the fresnel equation, d is the length of the light's underwater path, $P(\lambda,\phi)$ is the phase function, ρ is the water density. $fraca_Sa_C$ describes the ratio between the original water surface triangle and the current subvolume triangle.

Light does not only diminish with depth, but also with range. The farther away the light travels from the view point, the more it decreases. Therefore, the subsections are additionally split into tetrahedra, see 28. Because the intensity now depends on the view point, the subsection vertices are weighted according to their distance from the view point, see equation 23. A viewing ray penetrates the tetrahedra on their respective thickest spot through the intersection points A and B. The light intensity along the ray is finally interpolated by $I_C = \frac{I_A+I_B}{2}\left|\vec{AB}\right|$.

$$I_P(\lambda) = I_P^S(\lambda)e^{-\alpha_\lambda dist} \qquad (23)$$

There is an example image rendered with this method in figure 29. Iwasaki et al. [Iwasaki et al. 2002] also propose a variation of this technique to generate under water caustics. It is the base of many real time underwater god ray rendering techniques.

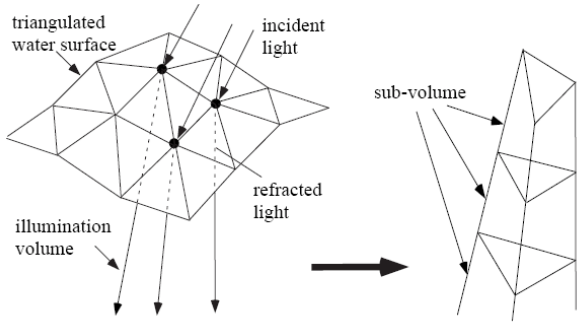

Figure 27:
creating illumination volumes:
The refracted light vectors and the water triangles compose illumination volumes.
Subsequently splitting illumination volumes:
An illumination volume is split horizontally into subvolumes

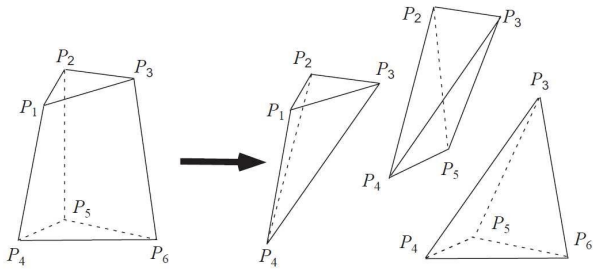

Figure 28:*Subvolumes are further split into tetrahedra.*

Figure 29:
This underwater scene of two dolphins shows that this method is capable of rendering underwater scenes with complex objects. It was generated on a desktop computer with a PentiumIII 1GHz CPU and a Geforce2ULTRA graphics card. The resolution of the image was 640x480, it was rendered within 5.1 seconds.

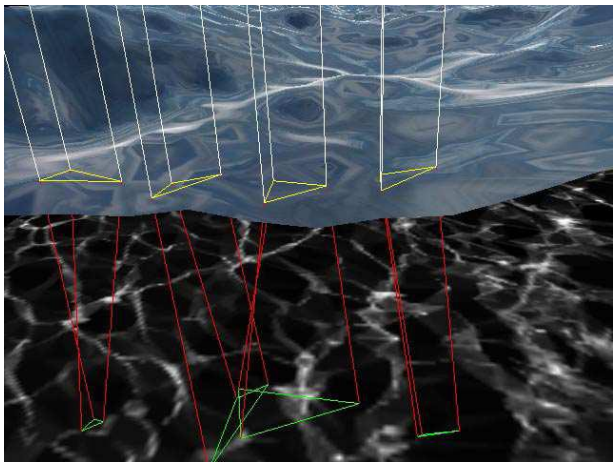

Figure 30:
At the intersected area of the projected water surface triangles, caustics occur.

8.2 Underwater God Rays Via Texture Maps

8.2.1 Caustics

In *Deep-Water Animation and Rendering* [Jensen and Goliáš 2001] an approach to render the phenomena of caustics in real-time is proposed by using the standard graphic primitives only. This algorithm also employs the concept of illumination volumes shown in section 8.1.

There are several constraints to this method: light rays can only be reflected or refracted once, no second order rays are considered. Furthermore, the ocean floor is defined to be in one uniform depth, without any possibility to add reefs or rifts - it is assumed flat. It will be the projection plane of our caustics.

As in section 8.1 the water surface is considered a mesh of triangles, through each of whom a light beam strikes the ocean floor. The beam irradiates vertically from the sky and its edges extend to the vertices of the water triangles. When it hits the water surface it is refracted using Snell's law (See chapter 3.1.2). The refracted beam is then projected onto the ocean floor. Since the water surface is not planar, the projected triangles are now partly intersected and distorted from refraction, as seen in figure 30.

The irradiating light intensity on the ocean floor can be estimated using the following equation:

$$I_c = N_s \cdot L(\frac{a_s}{a_c}) \tag{24}$$

where N is the normal vector of the water's surface triangle, L the vector along a light beam edge to the light, and the fraction describing the ratio of the water surface triangle's area to the area of its projection on the ocean floor.

The resulting intensities of the ocean floor triangles are now rasterized onto an texture, labeled caustic texture. White areas in the texture represent a high intensity, black areas low intensity. To reduce aliasing, the texture has to be filtered before further usage.
To apply the caustic texture onto objects underneath the surface, it is parallel projected from the surface in direction of the refracted light ray. For feigning light fainting with depth of the water the dot product of the triangle's surface normal and the light ray can be used to measure the intensity in which the texture is applied.

8.2.2 God Rays

The real-time caustics rendering method introduced by [Jensen and Goliáš 2001] and mentioned in section 8.2.1 can be extended to render god rays. The previously generated caustics texture will be used for the simulation of god rays in volume space.

The previously rendered caustic texture represents a single layer of a god ray structure at a specific depth, in this case the ocean floor. To obtain god rays throughout the whole body of water, caustic textures at different depths are needed. These god ray layers are then blended together with additive alpha-blending and rendered into the scene in a post-processing step.

To prevent aliasing, the amount of textures near the camera has to be higher. Since there are fewer textures in the distance, the god rays seemingly disappear into the distance. A second way to improve the quality of the rendered god rays, can be obtained by using the multitexturing abilities of the GPU to increase the the amount of samples.

Adding shadows of objects in the water via a shadow-buffer to the scene will result in more realism.

8.3 Underwater God Rays On GPU Through Pixel And Vertex Shaders

The algorithm proposed by Stefano Lanza [Lanza 2007] is a real-time rendering approach for underwater god rays. The method uses DirectX's HLSL-shaders to achieve this goal. All the work is split between two shaders, a vertex and a pixel shader.
To gain speed all second order physical lighting effects are ignored. This includes multiple scattering, the contribution of sky light, and the conservation of energy, when light shafts diverge or converge, instead a constant intensity is assumed.

In this method several ideas from algorithms outlined in previous sections are picked up. The light shafts take on a adapted form of parallelepipes from 8.1. Here the water surface is not subdivided into triangles but squares, see figure 31.

The pipes are represented by a indexed primitive consisting of two sets of vertices, describing the top and bottom square centering around a point $\hat{(}P)(0,0,0)$ with spacing 1, and a depth value. This primitive is translated and scaled by a vertex shader according to the viewers position E and his the depth. The first parallelpipe is placed at the center of the sun's reflection on the water surface according to the viewers position. T refracts the light ray, h_0 is the water altitude, I is the incident sunlight, and N the water normal.

$$\hat{P} = E + T(h_0 - E_y)/T_y, \tag{25}$$

$$T = refract(I, N, 1/1.333) \tag{26}$$

The spacing S to arrange the following god rays optimally is computed by equation 27, where K is the adjustment factor (circa 0.8), d the view's depth, fov the field of view and N the number of god ray vertices along the grid.

$$S = K\frac{2d\tan(fov/2)}{N} \tag{27}$$

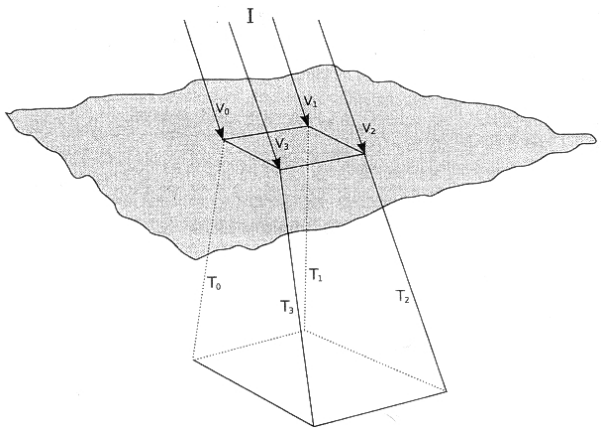

Figure 31:
Parallelepipe of a godray
I is the incident light, V_i are the vertices of the water surface square and T_i are the corresponding refracted vectors.

8.3.1 Vertex-Shader

The GPU handles the entire water animation, it is modeled as a sum of Gerstner waves as proposed by Hinsinger[Hinsinger et al. 2002]. A vertex shader calculates the water surface normal, for the HLSL-shader code see [Lanza 2007].

The vertex shader takes care of the water animation and refraction of light against the water surface. It calculates the new world space vertices of the refracted parallelpipes, uses the fresnel equation 3.1.5 to get light intensity after refraction and computes the water's light absorption with 28.

$$I_L(\lambda) = I_0(\lambda)e^{-c(\Lambda)L} \tag{28}$$

λ is the light's wavelength, $c(\lambda)$ the extinction coefficient and L is the length of the light's underwater path.
The resulting light intensities are then interpolated over the paralellepipe's polygons. By appointing these calculations to the vertex shader instead of the pixel shader we can reduce the fill rate, and therefore solve a great problem of god ray calculation. We also sacrify physical accuracy for better performance.

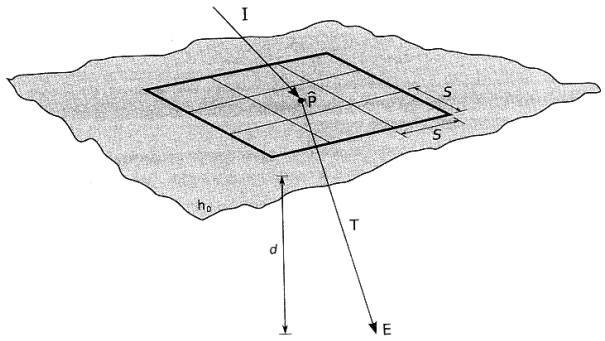

Figure 32:
Central position and spacing of a pipe-primitive
I is the incident light,T the corresponding refracted vector in the direction of the viewer E.$\hat{(P)}$ is the center point, S is the space to optimal place the primitives, calculated with equation 27.

8.3.2 Pixel-Shader

The pixel shader is responsible for the rendering part, it is divided into two steps.

1. All shadowed rays are computed and rendered into a temporary render target.
2. On each pixel of this render target the phase function is performed in image space.

To get the final image the result of step two is blended onto the rendered scene. For the pixel-shader code see [Lanza 2007].

8.4 God Rays and Caustics Through A Photon Grid

The process of photon mapping (section 4.3 and 6.1) provides physically accurate pictures at very high computational costs. [Papadopoulos and Papaioannou 2009] adapts ideas of that approach and approximates it to get it ready for real-time. The other base for [Papadopoulos and Papaioannou 2009] is the image-space ray scene intersection technique introduced in [Shah et al. 2007].

This new approach makes excessive use of render-to-texture techniques and programmable GPU shaders.
Caustics are simulated through photon point primitives of variable size while god rays are described by line primitives, which simulate scattering between water surface and a point at the ocean floor. The light intensities are calculated for each photon separately.

An overview over the whole algorithm is given in figure 33.

8.4.1 Preparations

As in photon mapping (c.f. section 4.3), the basic principle is to shoot photons into the scene, however a much smaller number of photon is used. Therefore we must ensure that each of the photons hits a visible surface. That is achieved by binding the light frustum to the camera frustum, see equation 30.

$$\vec{l} = p_{mid} - p_{light} \tag{29}$$

$$p_{mid} = p_{viewer} + \frac{z_{near} + z_{far}}{2} * \vec{l_{viewer}} \tag{30}$$

p_{mid} describes the center of the camera's frustum, calculated from the viewers position p_viewer, the camera's clipping distances z_{near} and z_{far}, and the viewer's line of sight $\vec{l_{viewer}}$. $\vec{(l)}$ is then the vector pointing from the center of the camera frustum to the light source p_{light}.

For later use the scene geometry is rendered to a off-screen buffer, and the z-buffer is saved for caustic and god ray generation.

8.4.2 Caustics

The emission of photons is modeled through a grid. That grid is further subdivided with the GPU's geometry shader to increase the trace able number of photon positions. To get a photons final position p_i on an underwater surface, we lay a ray $\vec{r_i}$ through a grid point. Next the intersection point between $\vec{r_i}$ and the water's surface point p_i is calculated. A geometry shader computes $\vec{r_i}$'s refraction, the result is the refracted ray $\vec{r_t}$.

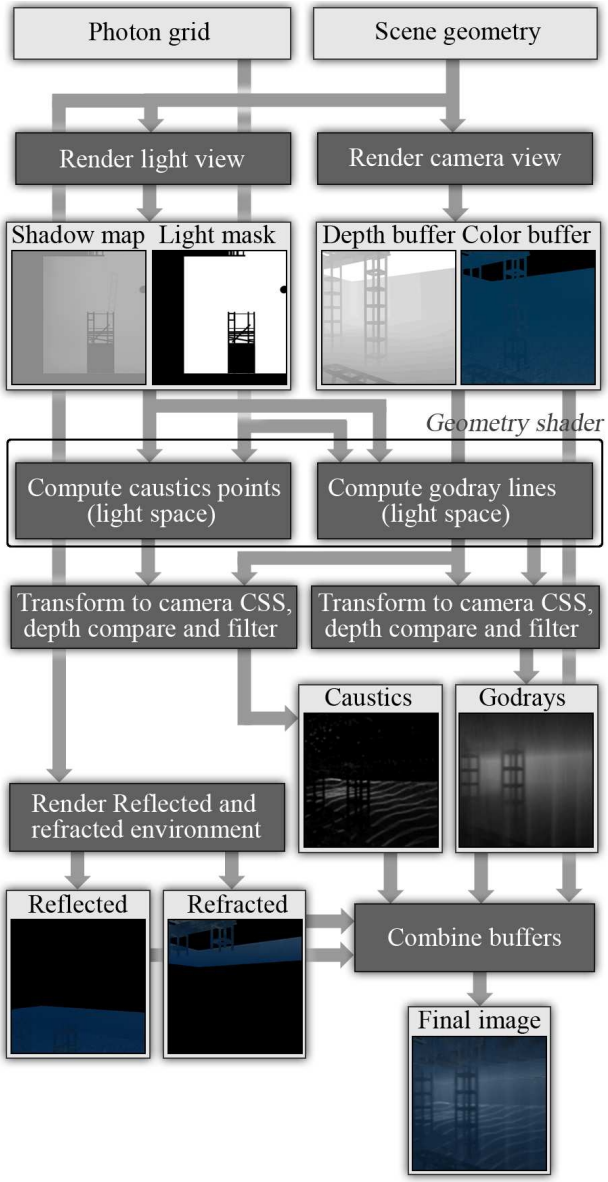

Figure 33: *Algorithm overview:*
First the shadow map and the scene geometry are rendered. Based on that, caustics and god rays are generated. To get the final image all buffers are combined to a single image.

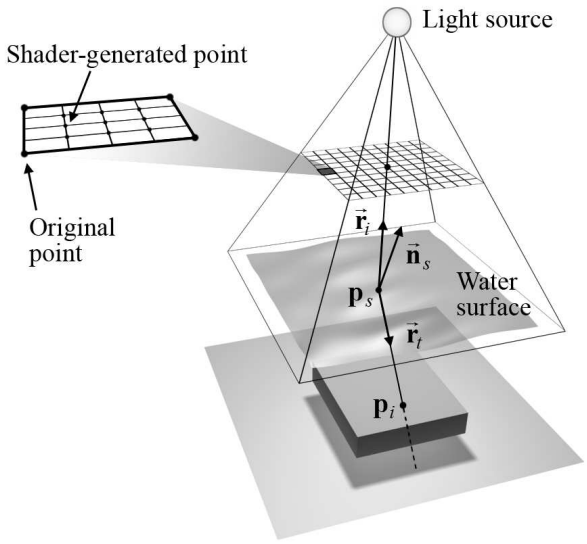

Figure 34:
The photon casting process:
A light source casts rays through a photon grid, which is subdivided by a geometry shader in even finer parts. A ray $\vec{r}_i$ hits the surface an a point p_s. There it is refracted as a vector $\vec{r}_t$ and finally hits the underwater surface point p_i where the photon primitive is placed.

The values of $\vec{r}_t$, p_s and the shadow map texture is delivered to the image space intersection algorithm [Shah et al. 2007]. Our adaption of that algorithm uses the Newton-Rhapson derived iterative method for approximation. The result of this calculations is the point p_i. This process is visualized in figure 34.

At p_i a photon point primitive of a specific size is emitted. The size is important because it is relevant for calculating the final caustic intensity. Photon point with constant size would cause the caustics to be overexposed by large point size or to cause noise by small point size. The solution is a variable point size depending on the distance to the viewer as shown in figure 35. The photon point primitive's final size is calculated with following equation 33. The light attenuation toward the view point's direction is also handled through the variable point size.

$$s_{final} = \frac{a+b}{d_{pointFromViewer}} \tag{31}$$

$$a = s_{max} - \frac{z_{far} * (s_{max} - s_{min})}{z_{far} - z_near} \tag{32}$$

$$b = \frac{z_{near} * z_{far} * (s_{max} - s_{min})}{z_{far} - z_near} \tag{33}$$

s_{final} is the final point size, while s_{max} is the maximal possible point size and s_{min} the minimal possible point size.
These point primitives are then transformed to camera space and rendered into a high-accuracy render target. They are also compared to the beforehand saved depth buffer.

The final light intensity of a photon primitive is calculated by 34. The illumination diminishment according to the water depth is considered through γ, which is the medium attenuation parameter, and $d_{fromSurface}$, which is the distance between the water surface p_s and

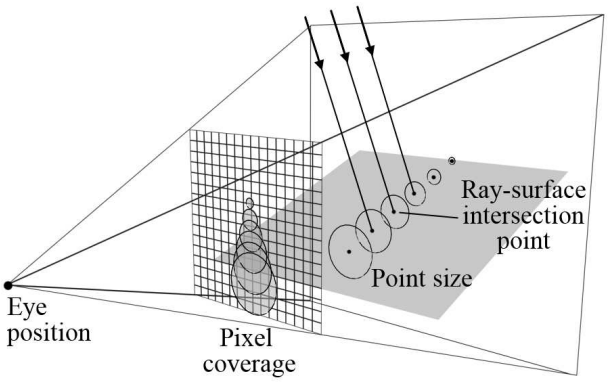

Figure 35:
The point size of the photon point primitive at an intersection point, shrinks with distance to the viewer. This describes the light attenuation in viewer direction. The photon point primitive's final size is calculated with equation 34.

the photons position p_i.

$$I_{final} = I_{photon} * e^{-\gamma * d_{fromSurface}} \tag{34}$$

8.4.3 God Rays

To generate god rays the same photon casting process which was used to create caustics, is employed, (c.f. figure 34). Yet instead of describing the light as a point primitive at a specific position on the sea floor, the light is modeled as a line primitive from the water surface point p_s to the point p_i.

Again the primitives are transformed, rasterized into a render-target and Z-tested.
The illumination for each line primitive is calculated with 35.

$$I_{final} = I_{photon} * Mie(\theta) * e^{-\gamma * d_{fromViewer}} * e^{-\gamma * d_{fromSurface}} \tag{35}$$

where $Mie(\theta)$ is the scattering phase function.

8.4.4 Further Treatment

The remaining steps are filtering and composing.
The filtering step prevents aliasing by applying a multitap low pass filter with a rotating sampling kernel on the rendered images waiting in the render-targets.
To create the final image, all images which were created in the different steps of the algorithm (color, reflection, refraction, god ray, caustic buffers and the rendered scene geometry) are combined to a single one.

This algorithm introduced by Papadopoulous[Papadopoulos and Papaioannou 2009] can be implemented for common graphics hardware, and can be easily integrated in modern 3D engines. It makes high frame rates in real-time possible.

9 Conclusion

Lighting effects such as caustics, god rays and light shafts are very important to render realistically appearing scenes. They add to the mood of a scene and to its impression on the viewer.

The best results are obtained by sticking close to the physical correct descriptions of light and it's behavior. Also the physical correct specification of the participating media adds to the authenticity of a scene.

We introduced several methods which can handle light characteristics, which are normally hard to obtain, such as anisotropic scattering, multiple scattering, color bleeding, god rays and light radiating through non-homogeneous media.
The price of this high accuracy are hight computing times and hight memory costs. That's why physically correct methods are only used for offline rendering.

Nowadays most 3D graphic applications, for instance games and visualizations, require real-time rendering.
This is always accomplished by simplifying and approximating the lighting model. By implementing the methods on the GPU more speed can be gained.

In the future the main task will be the realization of better approximations and the improvement of the computing capabilities of the GPU, to generate physical accurate lighting effects in real-time.

References

ARVO, J. 1986. Backward ray tracing. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '86, 259–263.

DMITRIEV, K., BRABEC, S., MYSZKOWSKI, K., AND SEIDEL, H.-P. 2002. Interactive Global Illumination Using Selective Photon Tracing. In *13th Eurographics Workshop on Rendering*, The Eurographics Association, P. Debevec and S. Gibson, Eds., 21–34.

DOBASHI, Y., YAMAMOTO, T., AND NISHITA, T. 2002. Interactive Rendering of Atmospheric Scattering Effects Using Graphics Hardware. In *Proceedings of Graphics Hardware 2002*, 99–108.

ENGELHARDT, T., AND DACHSBACHER, C. 2010. Epipolar sampling for shadows and crepuscular rays in participating media with single scattering. In *Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games 2010*, ACM Press, New York, New York, USA, 119.

ERNST, M., AKENINE-MÖLLER, T., AND JENSEN, H. W. 2005. Interactive rendering of caustics using interpolated warped volumes. In *In Proceedings of Graphics Interface*, 87–96.

FLECK, B., 2007. Real-Time Rendering of Water in Computer Graphics.

HINSINGER, D., NEYRET, F., AND CANI, M.-P. 2002. Interactive animation of ocean waves. In *ACM-SIGGRAPH/EG Symposium on Computer Animation (SCA)*.

HU, W., AND QIN, K. 2007. Interactive Approximate Rendering of Reflections, Refractions, and Caustics. *IEEE Transactions on Visualization and Computer Graphics 13*, 1, 46–57.

HURA, J., AND HALL, R. 2006. Design of a simulation of atmospheric sunbeams. *WSEAS Transactions on Computers 5*, 10, 2466–2471.

IWASAKI, K., DOBASHI, Y., AND NISHITA, T. 2002. An Efficient Method for Rendering Underwater Optical Effects Using Graphics Hardware. 701–711.

JENSEN, H. W., AND CHRISTENSEN, P. H. 1998. Efficient simulation of light transport in scences with participating media using photon maps. In *Proceedings of 25th annual conference on Computer graphics and interactive techniques - SIGGRAPH '98*, ACM Press, New York, New York, USA, 311–320.

JENSEN, L. S., AND GOLIÁŠ, R., 2001. Deep-Water Animation and Rendering.

JENSEN, H. W. 1996. Global illumination using photon maps. Springer-Verlag, 21–30.

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

KAJIYA, J. T. 1986. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '86, 143–150.

LANZA, S. 2007. Animation and Rendering of Underwater God Rays. In *Shader X5*, Charles River Media, W. Engel, Ed., 315–327.

LIKTOR, G., AND DACHSBACHER, C. 2010. Real-Time Volumetric Caustics with Projected Light Beams. In *Proceedings of 5th Hungarian Conference on Computer Graphics and Geometry*.

MITCHELL, J. L. 2004. Light Shaft Rendering. In *Shader X3*, W. Engel, Ed., 573–590.

MITCHELL, K. 2007. Volumetric Light Scattering as a Post-Process. In *GPU Gems 3*, Addison-Wesley Professional.

NISHITA, T., AND NAKAMAE, E. 1994. Method of Displaying Optical Effects within Water using Accumulation-Buffer. 373–380.

PAPADOPOULOS, C., AND PAPAIOANNOU, G. 2009. Realistic Real-time Underwater Caustics and Godrays. In *Proceedings of GraphiCon '09*, 89–95.

PURCELL, T. J., CRAIG, D., CAMMARANO, M., JENSEN, H. W., AND HANRAHAN, P. 2003. Photon mapping on programmable graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, Eurographics Association, 41–50.

ROHLEDER, P., AND JAMROZIK, M. 2008. Sunlight with Volumetric Light Rays. In *Shader X6*, Charles River Media, W. Engel, Ed., 325–331.

SHAH, M. A., KONTTINEN, J., AND PATTANAIK, S. 2007. Caustics Mapping: An Image-Space Technique for Real-Time Caustics. *IEEE Transactions on Visualization and Computer Graphics 13*, 2, 272–280.

SHENG, B., SUN, H., LIU, B., AND WU, E. 2009. GPU-based refraction and caustics rendering on depth textures. In *Proceedings of 8th International Conference on Virtual Reality Continuum and its Applications in Industry*, ACM Press, New York, 139–144.

STAM, J. 1996. Random caustics: natural textures and wave theory revisited. In *ACM SIGGRAPH 96 Visual Proceedings: The art and interdisciplinary programs of SIGGRAPH '96*, ACM, New York, NY, USA, SIGGRAPH '96.

STAMATE, V. 2008. Real-Time Photon Approximation on the GPU. In *Shader X6*, Charles River Media, W. Engel, Ed., 393–400.

SUN, X., ZHOU, K., LIN, S., AND GUO, B. 2010. Line Space Gathering for Single Scattering in Large Scenes. In *Proceedings of ACM Transactions on Graphics (SIGGRAPH 2010)*.

SZIRMAY-KALOS, L., UMENHOFFER, T., PATOW, G., SZÉCSI, L., AND SBERT, M. 2009. Specular effects on the gpu: State of the art. *Computer Graphics Forum 28*, 6, 1586–1617.

TÓTH, B., AND UMENHOFFER, T. 2009. Real-time Volumetric Lighting in Participating Media. *Eurographics 2009*.

TRENDALL, C., AND STEWART, A. J. 2000. General calculations using graphics hardware with applications to interactive caustics. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, Springer-Verlag, London, UK, 287–298.

VEACH, E., AND GUIBAS, L. J. 1997. Metropolis light transport. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '97, 65–76.

WALLACE, J. R., ELMQUIST, K. A., AND HAINES, E. A. 1989. A ray tracing algorithm for progressive radiosity. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '89, 315–324.

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

WATT, M. 1990. Light-water interaction using backward beam tracing. *SIGGRAPH Comput. Graph.*, 377–385.

WILLIAMS, L. 1988. *Casting curved shadows on curved surfaces*. Computer Science Press, Inc., New York, NY, USA, 23–27.

WYMAN, C., AND DACHSBACHER, C. 2006. Improving image-space caustics via variable-sized splatting. In *Tech.Rep. UICS-06-02*, University of Utah.

WYMAN, C., AND DAVIS, S. 2006. Interactive image-space techniques for approximating caustics. In *In Proceedings of ACM I3D*, ACM Press. 2, 153–160.

WYMAN, C. 2007. Interactive Refractions and Caustics Using Image-Space Techniques. In *Shader X5*, Charles River Media, W. Engel, Ed., 359–371.

WYMAN, C. 2008. Hierarchical caustic maps. In *Proceedings of the 2008 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, 163–171.