Interactive Rendering of Caustics using Dimension Reduction for Manifold Next-Event Estimation

Ana Granizo-Hidalgo, Nicolas Holzschuch

Interactive Rendering of Caustics using Dimension Reduction for Manifold Next-Event Estimation

ANA GRANIZO-HIDALGO, Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LJK, France
NICOLAS HOLZSCHUCH, Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LJK, France

Fig. 1. Refractive specular surfaces, like the water surface in this pool, create caustics. Classical path-tracing (left) cannot find a path connecting the bottom of the pool and the point light source, and fails at rendering these caustics. Manifold exploration methods (right) work but are not compatible with interactive rendering. Our method (center) runs interactively, by reducing the space where it searches for solutions to a curve.

Specular surfaces, like water surfaces, create caustics by focusing the light being refracted or reflected. These caustics are very important for scene realism, but also challenging to render: to compute them, we need to find the exact path connecting two points through a specular reflective or refractive surface. This requires finding the roots of a complicated function on the surface. Manifold-Exploration methods find these roots using the Newton-Raphson method, but this involves computing path derivatives at each step, which can be challenging. We show that these roots lie on a curve on the surface, which reduces the dimensionality of the search. This dimension reduction greatly improves the search, allowing for interactive rendering of caustics. It also makes implementation easier, as we do not need to compute path derivatives.

CCS Concepts: • **Computing methodologies** → **Ray tracing**.

Additional Key Words and Phrases: Interactive ray-tracing, Caustics, Next-Event Estimation

ACM Reference Format:
Ana Granizo-Hidalgo and Nicolas Holzschuch. 2024. Interactive Rendering of Caustics using Dimension Reduction for Manifold Next-Event Estimation. In . ACM, New York, NY, USA, Article 1, 16 pages. https://doi.org/10.1145/3651297

1 INTRODUCTION

When we look at a body of water, we see bright animated lines at the bottom. These bright lines are called *caustics*, and are caused by the light being refracted by the water surface. As the water

I3D '24, May 08–10, 2024, Philadelphia, PA

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/10.1145/3651297

surface is curved, the refraction focuses the refracted light at certain points. These caustics are essential in rendering the appearance of specular materials, especially water.

Computing these caustics is challenging, precisely because of the refraction at the specular surface. During rendering, we need to connect points on both sides of the surface, but there are no easy ways to find a refracted path connecting two points. The classical path-tracing methods, using Next-Event Estimation, fail, resulting in a dark area under the surface. Alternative methods such as Manifold Next-Event Estimation [Hanika et al. 2015; Walter et al. 2009] work, but they require computing path derivatives. Their key idea is to translate the search for a refracted path into the search for the zeros of an objective function, which is done using Newton-Raphson's method.

Newton-Raphson's method is efficient, but it requires computing the derivatives of the objective function; this is potentially challenging, especially on the GPU. In this paper, we present an alternative method where we start by reducing the *dimensionality* of the search, using a *coplanarity constraint*: the incoming ray, the refracted ray and the normal to the surface must be coplanar. This condition defines a one-dimensional curve on the surface, and we restrict our search for solutions to this curve. With the dimension reduction, we find refracted paths efficiently and consistently.

In summary, we present a new algorithm to compute caustics using Manifold Next-Event Estimation. Our algorithm uses the coplanarity constraint to reduce the dimension of the space where we search for solutions, resulting in faster and more consistent computations. Our algorithm is fast enough to allow rendering underwater caustics in real-time, with an animated water surface.

2 RELATED WORK

Mitchell and Hanranhan [1992] computed reflective caustics on curved surfaces using the surface derivatives. They identify reflected paths as paths with extremal length, based on Fermat's principle, and use Newton-Raphson's method to find these paths.

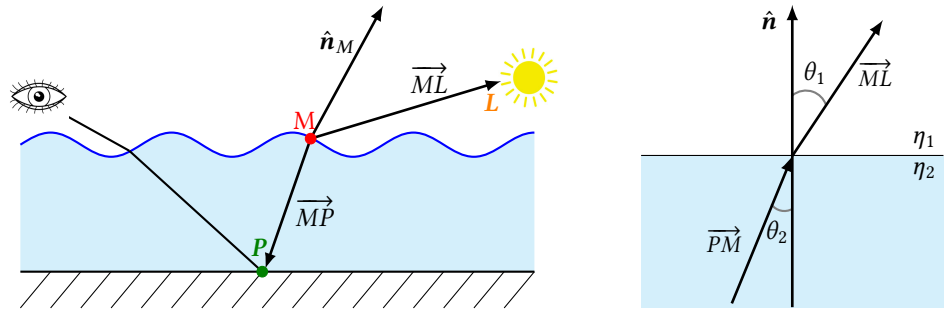
Jensen [1996] introduced Photon Mapping, a two-step generic method to compute global illumination, including caustics: in a first step, photons are scattered through the scene. In the second step, they are gathered to reconstruct illumination effects. Photon mapping is very versatile: it can render almost all illumination effects. The quality of high-frequency effects such as caustics depends on the number of photons allocated.

Walter et al. [2009] compute refracted paths inside surfaces defined by triangles with interpolated normals. They identify refracted paths as paths through points where the half-vector is equal to the opposite of the normal, and search for these points using Newton-Raphson's method.

Jakob and Marschner [2012] extend Metropolis Light Transport [Veach and Guibas 1997] by restricting the space of perturbations to a manifold, computed using the constraints introduced by specular surfaces. It is very efficient to find specular paths into the scene, but not so efficient when we need to connect the last point of a path to the light source.

Practical path guiding [Müller et al. 2017] gradually learns the incoming radiance distribution using an adaptive spatio-directional hybrid data structure, resulting in stable performance in scenes with difficult geometry, including caustics. This performance stability is a strong advantage of the method, making it compatible with production environments.

Classical path-tracing usually relies on Next-Event Estimation: computing direct illumination by connecting the current point to the light source using a shadow ray. Next-Event Estimation dramatically improves convergence speed, but it is not available when there is a refractive surface on the way. Hanika et al. [2015] combine Next-Event Estimation and Manifold exploration into *Manifold Next-Event Estimation*: they use manifold exploration to find the refracted path through the surface, connecting the light source with the last point on the path.



(a) A specular path through a refractive interface. (b) Snell's law.

Fig. 2. Our problem: we consider a point P, at the end of a path traced from the camera. We search for a refracted path through a point M on the surface connecting this point and the light source (left). Paths through the refractive surface must obey Snell's law (right): $\eta_1 \sin\theta_1 = \eta_2 \sin\theta_2$.

Zeltner et al. [2020] improve this technique, using a different objective function and systematic search for multiple solutions. Loubet et al. [2020] used slope-space integrals for fast computation of specular next-event estimation.

Our work uses the same objective function as [Zeltner et al. 2020], but we do not compute derivatives in our search for solutions, relying instead on geometric constraints to reduce the search space.

Manifold Next-Event Estimation methods [Hanika et al. 2015; Zeltner et al. 2020] compute a single specular path and work well for specular or quasi-specular surfaces with point light sources. Li et al. [Li et al. 2022] compute caustics for glossy materials using path guiding around the path computed for a specular surface.

All these techniques target offline rendering. Yang and Ouyang [2021] render high-quality water caustics in real-time, by first tracing photons through the water mesh, then converting the hit points into a procedural caustics mesh, to be composited in the rasterization pass.

3 OUR ALGORITHM

3.1 Problem position

We place ourselves in the same settings as previous works [Hanika et al. 2015; Walter et al. 2009; Zeltner et al. 2020]: we have a scene with a medium enclosed by a refractive surface, such as water. As depicted in Figure 2a, we have traced a path from the camera into the scene until a point $\boldsymbol{P}$ inside this medium. We are looking for a path that connects this point $\boldsymbol{P}$ to the light source $\boldsymbol{L}$ through a point M at the refractive surface, taking into account the varying normals at the surface. This path must obey the laws of refraction at the interface:

- $\overrightarrow{PM}$, $\overrightarrow{ML}$ and the normal at M, $\hat{\boldsymbol{n}}_M$ must be coplanar.
- The angles between these vectors must obey Snell's law (see Figure 2b): $\eta_1 \sin\theta_1 = \eta_2 \sin\theta_2$, where η_1 is the refractive index outside of the medium, and η_2 is the refractive index inside the medium.

To find the point M such that the incoming ray (PM) is refracted into (ML) and reaches the light source, we write an objective function f such that $f(M) = 0$ when M is on a valid refracted path. Walter et al. [2009] used the half-vector; Zeltner et al. [2020] used the angle between the refracted ray computed from (PM) and the line joining M and L. We use the latter in our implementation, but our approach works with both.

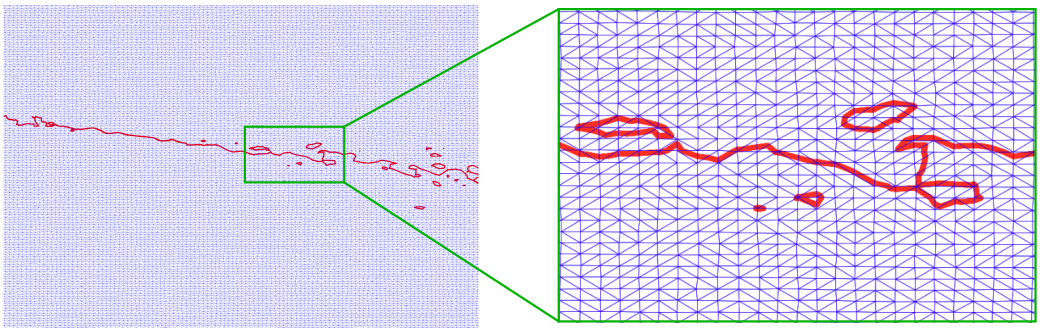


Fig. 3. Given a water surface made of triangles with interpolated normals, a point P inside the water and a light source L, the set of points M on the surface such that $\overrightarrow{PM}$, $\overrightarrow{LM}$ and $\hat{\boldsymbol{n}}_M$ are coplanar create the red curve.

3.2 Notation

In all this paper, we use $\hat{\boldsymbol{u}}$ to denote a normalized vector ($\|\hat{\boldsymbol{u}}\| = 1$), and $\boldsymbol{u}$ to denote a general vector.

3.3 Coplanarity constraint

Our key idea is to separate the constraints on M, and to treat the coplanarity constraint first: the three vectors $\overrightarrow{PM}$, $\overrightarrow{LM}$ and the normal at point M, $\hat{\boldsymbol{n}}_M$ must be coplanar. This creates a condition on M that reduces the dimensionality of the problem: instead of M being free to move over the entire two dimension surface of the refractive medium, it has to be on a specific one-dimension curve on this surface (see Figure 3). We restrict the search for solutions to this curve, allowing for a more efficient search. We enforce this constraint *before* normalizing the vectors, which also reduces the *degree* of the problem.

3.3.1 Flat surfaces. As an illustration, let us consider a plane $\mathcal{P}$ with a constant normal $\hat{\boldsymbol{n}}$: the coplanarity constraint means that the triple product of the three vectors $\overrightarrow{PM}$, $\overrightarrow{LM}$ and $\hat{\boldsymbol{n}}$ is null:

$$\hat{\boldsymbol{n}} \cdot \left(\overrightarrow{PM} \times \overrightarrow{LM} \right) = 0 \tag{1}$$

This means that M must be on a straight line on the plane. This straight line connects the orthogonal projections of points $\boldsymbol{P}$ and $\boldsymbol{L}$ on the plane, $\boldsymbol{P}_p$ and $\boldsymbol{L}_p$:

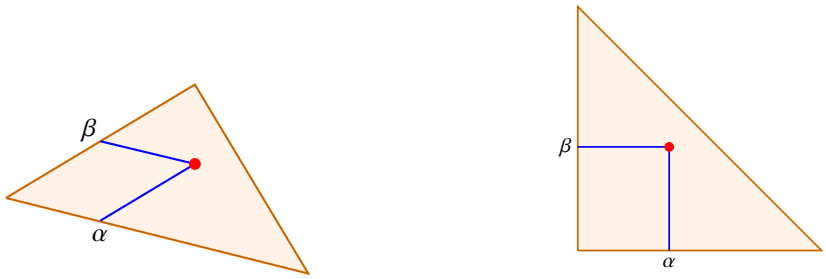
$$M \in (\boldsymbol{P}_p \boldsymbol{L}_p) \tag{2}$$

3.3.2 Triangle with interpolated normals. The most common graphics primitive in Computer Graphics scenes is the triangle with interpolated normals. It is defined by three vertices $\{\boldsymbol{V}_i\}$ and their associated normals $\{\hat{\boldsymbol{n}}_i\}$. On these triangles, the coplanarity constraint forces M to move on a conic (ellipsis, hyperbola or parabola) drawn on the triangle.

To show why, we write the position of M using barycentric coordinates (α, β) (see Figure 4):

$$M = \boldsymbol{V}_0 + \alpha \overrightarrow{V_0 V_1} + \beta \overrightarrow{V_0 V_2}, \text{ with } (\alpha, \beta, \alpha + \beta) \in [0, 1]^3 \tag{3}$$

$$M = \begin{bmatrix} \overrightarrow{V_0 V_1} & \overrightarrow{V_0 V_2} & \boldsymbol{V}_0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ 1 \end{bmatrix} \tag{4}$$



(a) (b)

Fig. 4. A point inside a triangle in World Space (a), and in barycentric coordinate Space (b).

Any vector connecting M and a fixed point is also expressed as a matrix-vector product, for example:

$$\overrightarrow{PM} = \left[\overrightarrow{V_0V_1} \quad \overrightarrow{V_0V_2} \quad \overrightarrow{PV_0}\right] \begin{bmatrix} \alpha \\ \beta \\ 1 \end{bmatrix} \tag{5}$$

The normal at point M, $\boldsymbol{n}_M$ is also a matrix-vector product:

$$\boldsymbol{n}_M = \boldsymbol{n}_0 + \alpha(\boldsymbol{n}_1 - \boldsymbol{n}_0) + \beta(\boldsymbol{n}_2 - \boldsymbol{n}_0) \tag{6}$$

$$\boldsymbol{n}_M = \left[(\boldsymbol{n}_1 - \boldsymbol{n}_0) \quad (\boldsymbol{n}_2 - \boldsymbol{n}_0) \quad \boldsymbol{n}_0\right] \begin{bmatrix} \alpha \\ \beta \\ 1 \end{bmatrix} \tag{7}$$

$$\boldsymbol{n}_M = [\boldsymbol{N}] \begin{bmatrix} \alpha \\ \beta \\ 1 \end{bmatrix} \tag{8}$$

Without loss of generality, we express the coplanarity constraint using the triple product between the vectors $\overrightarrow{PL}$, $\overrightarrow{PM}$ and $\boldsymbol{n}_M$:

$$\boldsymbol{n}_M \cdot \left(\overrightarrow{PL} \times \overrightarrow{PM}\right) = 0 \tag{9}$$

$$\begin{bmatrix} \alpha & \beta & 1 \end{bmatrix} [\boldsymbol{N}]^{\mathrm{T}} [\boldsymbol{Q}] \begin{bmatrix} \alpha \\ \beta \\ 1 \end{bmatrix} = 0 \tag{10}$$

where $[\boldsymbol{Q}]$ is the matrix whose column vectors are: $\overrightarrow{PL} \times \overrightarrow{V_0V_1}$, $\overrightarrow{PL} \times \overrightarrow{V_0V_2}$ and $\overrightarrow{PL} \times \overrightarrow{PV_0}$. We use the fact that the dot product between two vectors can be expressed as a vector product between the transpose of the first vector and the second vector:

$$\boldsymbol{u} \cdot \boldsymbol{v} = \boldsymbol{u}^{\mathrm{T}}\boldsymbol{v} = u_x v_x + u_y v_y + u_z v_z \tag{11}$$

Equation 10 gives the equation for the set of points that satisfy the coplanarity constraint; it is the kernel of a quadratic form, of matrix $[\boldsymbol{N}]^{\mathrm{T}} [\boldsymbol{Q}]$, that is a conic. Each coefficient in the $[\boldsymbol{N}]^{\mathrm{T}} [\boldsymbol{Q}]$ matrix is a triple product. We expand Equation 10 into an equation of the second degree of two variables:

$$A\alpha^2 + B\beta^2 + C\alpha\beta + D\alpha + E\beta + F = 0 \tag{12}$$

3.3.3 Full surface. The surface around the participating medium is an assembly of triangles with interpolated normals. On each triangle, the solution for the coplanarity constraint is one or several pieces of a conic: the intersection between the conic solution of Equation 10 and the boundaries of the triangle (See Figure 5b).

The normals are defined at each vertex and interpolated linearly, so they vary continuously across the edge between two triangles. The curve segments for one triangle connect continuously with the curve segments for the neighboring triangles. Over the entire surface, the solution for the coplanarity constraint is a set of conic segments joined together (see Figure 3).

3.3.4 Curve tangent. To guide our search or orient the curve, we will need to know the tangent to the curve. We use the following property of conics: if (α_0, β_0) is a point on the conic defined by the matrix $[\boldsymbol{M}]$, then the tangent at that point has the equation:

$$\begin{bmatrix}\alpha_0 & \beta_0 & 1\end{bmatrix}[\boldsymbol{M}]\begin{bmatrix}\alpha\\ \beta\\ 1\end{bmatrix} = 0 \tag{13}$$

This gives us the *direction vector* $\boldsymbol{t}$ of the tangent in barycentric coordinates:

$$\begin{bmatrix}u\\ v\\ w\end{bmatrix} = [\boldsymbol{M}]^{\mathrm{T}}\begin{bmatrix}\alpha_0 & \beta_0 & 1\end{bmatrix} \tag{14}$$

$$\boldsymbol{t} = \begin{bmatrix}-v\\ u\end{bmatrix} \tag{15}$$

3.4 Algorithm

We use the coplanarity constraint to search quickly for refracted paths through the surface.

3.4.1 Problem reformulation. We have restricted the search space to a set of points where $\overrightarrow{PM}$, $\overrightarrow{ML}$ and $\boldsymbol{n}_M$ are coplanar. On this curve, we are looking for the points where the refracted ray is colinear with the line joining M and L:

$$\text{refract}\left(\overrightarrow{PM}\right) \times \overrightarrow{ML} = \mathbf{0} \tag{16}$$

For simplification, we introduce $\hat{\boldsymbol{\omega}}_P$ and $\hat{\boldsymbol{\omega}}_L$:

$$\hat{\boldsymbol{\omega}}_P = \frac{\overrightarrow{MP}}{\|\overrightarrow{MP}\|} \tag{17}$$

$$\hat{\boldsymbol{\omega}}_L = \frac{\overrightarrow{ML}}{\|\overrightarrow{ML}\|} \tag{18}$$

The refracted vector coming out of the surface is refract $(-\hat{\boldsymbol{\omega}}_P)$. When M corresponds to a refracted path, refract $(-\hat{\boldsymbol{\omega}}_P)$ is equal to $\hat{\boldsymbol{\omega}}_L$. We write the cross-product between these two vectors:

$$|\sin\theta| = \|\text{refract}\,(-\hat{\boldsymbol{\omega}}_P) \times \hat{\boldsymbol{\omega}}_L\| \tag{19}$$

We're looking for the zeroes of this sine. This will be easier if we can give it a sign, instead of dealing with absolute values. We use the direction vector $\boldsymbol{t}$ of the tangent (Equation 15) to orient our vectors:

$$s = \begin{cases} 1 & \text{if} \quad \boldsymbol{t}\cdot(\text{refract}\,(-\hat{\boldsymbol{\omega}}_P) \times \hat{\boldsymbol{\omega}}_L) > 0 \\ -1 & \text{if} \quad \boldsymbol{t}\cdot(\text{refract}\,(-\hat{\boldsymbol{\omega}}_P) \times \hat{\boldsymbol{\omega}}_L) < 0 \end{cases} \tag{20}$$

With this orientation, we can define the function f:

$$f(t) = s\|\text{refract}\,(-\hat{\boldsymbol{\omega}}_P) \times \hat{\boldsymbol{\omega}}_L\| \tag{21}$$

where f is a function of the arc length on the curve, t. It is equal to the sine of the angle between the refracted vector and the vector connecting M to the light source. All zeroes of f correspond to potential refracted paths connecting P and L; our problem is to find all these zeroes.

Compared to previous approaches, we are searching for the zeroes of a function with real-valued parameters and real values, instead of the zeroes of a function of multi-dimensional parameters with multi-dimensional values.

3.4.2 Starting point. The coplanarity constraint reduces the space where we search for solutions; we just need to follow the curve until we reach a point where $f(M) = 0$ (see Equation 21). We need a starting point that is on the curve. The point M_0 on the intersection between the line (PL) and the refractive surface trivially satisfies the coplanarity constraint, since $\overrightarrow{PM_0} \times \overrightarrow{LM_0} = 0$. We use it as the starting point in our search.

This point has also been used as a starting point for several Newton-Raphson's methods looking for the zeroes of f [Hanika et al. 2015; Walter et al. 2009].

3.4.3 Walking along the curve. We compute the equation of the conic that is a solution to Equation 10, and then we compute its intersection with the edges of the current triangle, and use these to move to the next triangle.

Our algorithm is iterative. At each step, we are considering one triangle on the refractive surface, for which we know there is at least one point that satisfies the coplanarity constraint. We then compute the intersections between the curve and the edges of the triangle, by setting $\alpha = 0$, $\beta = 0$ or $\alpha = 1 - \beta$ in Equation 12 and solving a second-degree equation for the other barycentric coordinate:

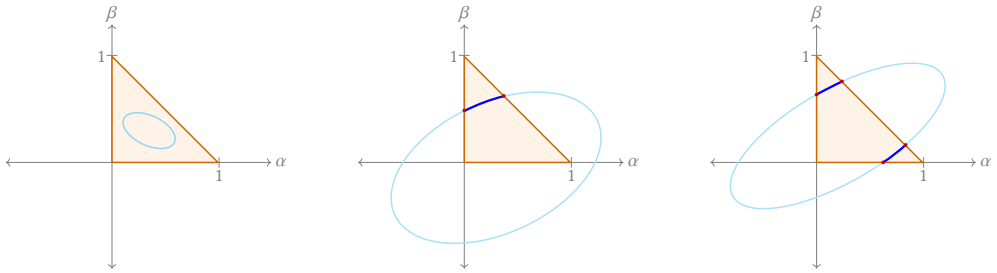
$$\alpha = 0 \quad \longrightarrow \quad B\beta^2 + D\beta + F = 0 \tag{22}$$

$$\beta = 0 \quad \longrightarrow \quad A\alpha^2 + C\alpha + F = 0 \tag{23}$$

$$\alpha + \beta = 1 \quad \longrightarrow \quad (A + B - C)\beta^2 + (C + E - 2A - D)\beta + A + D + F = 0 \tag{24}$$

Each of these equations is an equation of the second degree with one variable; we compute both solutions and keep only the solutions that lie inside $[0, 1]$.

- If there are no intersection points, then the curve is an ellipse, entirely included inside the triangle, Figure 5a. We use a specific solving algorithm for this rare case.
- In the generic case, there are two intersection points, (see Figure 5b). We compute the value of f at these two points:
  - if f does not change sign, then we move to the next neighbouring triangle along this edge.
  - if f changes sign over the triangle, then there is a solution inside the triangle. We find it using dichotomy:
    - we subdivide the triangle in two sub-triangles, so that the new edge intersects the curve,
    - we compute the value of f at the new intersection point,
    - we select the sub-triangle where f changes sign and iterate.
- In theory, there can be triangles with more than two intersection points. This happens very rarely in our experiments. Specifically, this occurs when the curve enters one edge of the triangle, exits by another edge, but then re-enters the triangle and exits again, see Figure 5c. We solve this case by following the curve to find a neighboring triangle that has yet to be visited.



(a) Ellipse included in the triangle. (b) Two intersections. (c) Four intersections.

Fig. 5. Types of intersections between the conic and the triangle.

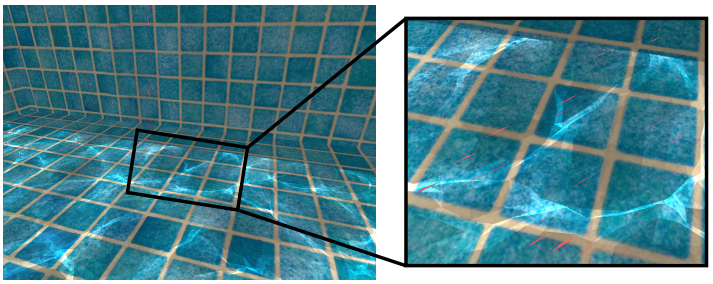


Fig. 6. In red, the cases where there is an ellipse inside a triangle.

3.4.4 Special case: ellipse included in the triangle. In some cases, the entire curve of points that satisfy the coplanarity constraint is included inside a single triangle. In these cases, our algorithm based on the intersections between the curve and the triangle edges fails. This special case happens rarely (see Figure 6) and is easy to detect. It can only happen on the first triangle of the search. We solve this specific case by iteratively subdividing the triangle by splitting an edge, until we find a triangle where f changes sign, then we apply the previous algorithm.

There may be cases with this effect, which our algorithm will not detect because it does not happen in the first triangle. Since there is no curve that intersects its edges, we cannot reach that triangle.

3.4.5 Contribution from a point. Once we have a point M on a refracted path connecting P and L, we compute the contribution from this path using path derivatives, as in Walter et al. [2009]:

$$contribution = \frac{I_e \, F \, A \, \rho}{D} \tag{25}$$

where I_e is the intensity of the light, F is the Fresnel factor, A is the volume attenuation, ρ is the BRDF at the shading point, and D is the distance correction factor, taking into account how the light is focused by the varying normals.

Because we have shading normals, we need to compute D using ray differentials. We start by computing two directions $\boldsymbol{\omega}_\perp$ and $\boldsymbol{\omega}_\parallel$, perpendicular to $\boldsymbol{\omega}_P$ and to each other. For each direction,

we compute the ray differential $\boldsymbol{L}'$ using:

$$
\begin{aligned}
d_P &= \|P - M\| && (26)\\
d_L &= \|L - M\| && (27)\\
M' &= d_P \left(\hat{\boldsymbol{\omega}}'_P - \frac{\hat{\boldsymbol{\omega}}'_P \cdot \hat{\boldsymbol{n}}_g}{\hat{\boldsymbol{\omega}}_P \cdot \hat{\boldsymbol{n}}_g} \hat{\boldsymbol{\omega}}_P \right) && (28)\\
\mu &= \hat{\boldsymbol{\omega}}_L \cdot \hat{\boldsymbol{n}}_s + \eta \hat{\boldsymbol{\omega}}_P \cdot \hat{\boldsymbol{n}}_s && (29)\\
\mu' &= \left(\eta^2 \frac{\hat{\boldsymbol{\omega}}_P \cdot \hat{\boldsymbol{n}}_s}{\hat{\boldsymbol{\omega}}_L \cdot \hat{\boldsymbol{n}}_s} + \eta \right) \left(\hat{\boldsymbol{\omega}}'_P \cdot \hat{\boldsymbol{n}}_s + \hat{\boldsymbol{\omega}}_P \cdot \hat{\boldsymbol{n}}'_s \right) && (30)\\
\hat{\boldsymbol{\omega}}'_L &= \eta \hat{\boldsymbol{\omega}}'_P + \mu' \hat{\boldsymbol{n}}_s + \mu \hat{\boldsymbol{n}}'_s && (31)\\
\boldsymbol{L}' &= M' - (M' \cdot \hat{\boldsymbol{\omega}}_L) \hat{\boldsymbol{\omega}}_L + d_L \hat{\boldsymbol{\omega}}'_L && (32)
\end{aligned}
$$

Where we use the notation that a' is the derivative of a along $\boldsymbol{\omega}_\perp$ (resp. $\boldsymbol{\omega}_\parallel$). Finally:

$$D = \|\boldsymbol{L}'_\perp \times \boldsymbol{L}'_\parallel\| \quad (33)$$

4 IMPLEMENTATION DETAILS

Unless otherwise specified, all the numbers and figures in this paper are computed on an Intel Xeon with 2.40 GHz, 32 cores and an NVIDIA RTX A4000 graphics card. We implemented our algorithm on the GPU using the Vulkan ray-tracing extensions. All results are rendered with a resolution of 960x720 pixels.

For our algorithm to work, it is necessary to have a mesh of triangles with interpolated normals. Additionally, we need to know the location of each triangle with respect to its neighbors. For this, we use a buffer containing the information of the three neighbors of each triangle and their respective locations. This buffer is pre-computed for each mesh and accessed in real-time.

In all our scenes, we extended the surface of the water to be slightly larger than the pool walls so that when we send a ray from the bottom toward the light, we always have an intersection with the plane of the water. The extended water surface is not visible since it is occluded by the floor.

When we compute the starting point on the curve, we send a ray from point P to the light source; here we are only interested with an intersection between that ray and the water surface. We use Vulkan masking feature to only check for intersections with this surface. This speeds up the computation and allows us to deal with objects floating on the water surface or inside the water (see Figure 15b). It also avoids numerical accuracy issues when the ray-surface intersection is close to another object.

Our algorithm has two additional ray evaluations in addition to the usual ray tracing: the first when we send the ray from the shading point toward the light to find the starting point, and the second is to determine whether the point found as a specular path is visible from the light.

5 RESULTS AND COMPARISON

5.1 Validation: comparison with Specular Manifold Sampling on the CPU

To have a validation and comparison of our method, we have a CPU implementation version, so we can compare in similar conditions with other CPU based methods. For a comparison with Specular Manifod Sampling (SMS) [Zeltner et al. 2020], we implemented our algorithm inside Mitsuba 2 [Nimier-David et al. 2019].

Zeltner et al. [2020] provide two variants of Specular Manifold Sampling: the biased and unbiased versions. Both are parameterized by the number of samples per pixel; in the biased version, it is possible to set the number of solutions to search for, for each sample. Our method only searches for

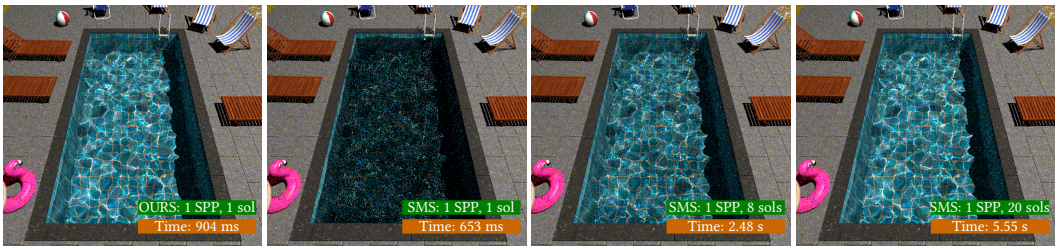


Fig. 7. Comparison between our algorithm and Specular Manifold Sampling (biased variant) inside Mitsuba 2. 1 sample per pixel (SPP), 7 ray-tracing bounces.

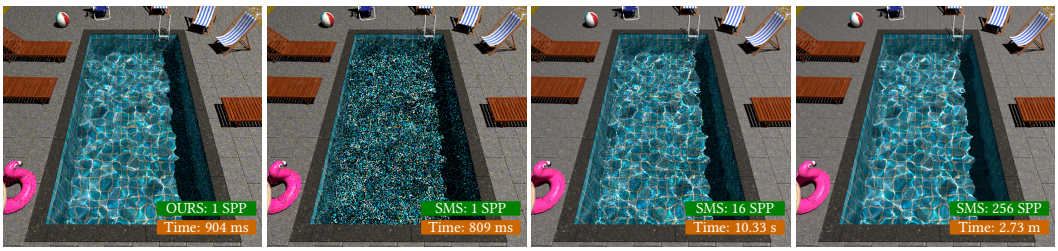


Fig. 8. Comparison between our algorithm and Specular Manifold Sampling (unbiased variant) inside Mitsuba 2. 7 ray-tracing bounces.

one solution per sample. All the images in this section compute up to 7 bounces from the camera to the light source.

We first compare our algorithm with the biased variant of Specular Manifold Sampling (see Figure 7). At their minimal quality settings (one solution per sample), the computation time is similar to ours, but the quality is much better with our method. As we increase the number of solutions per sample, the quality of Specular Manifold Sampling increases, along with computation time. At approximately 20 solutions per sample, the quality is comparable, but our algorithm is six times faster.

We also compare our method with the unbiased variant of Specular Manifold Sampling (see Figure 8). At the minimal quality settings (one sample per pixel), the computation time is again similar to ours, but the quality is much better with our method. As with the biased version, increasing the number of samples per pixel improves the quality, at the expense of computation time. The unbiased variant tends to produce noisier images for low quality settings; it takes 256 samples per pixel to obtain a noise-free image comparable to ours, taking more than two minutes to generate, compared to less than one second for our method.

5.2 Comparison with Photon Mapping and Path Guiding on the CPU

We also compared our algorithm, running on the CPU, with Photon Mapping [Jensen 1996] and Practical Path Guiding [Müller et al. 2017], both implemented inside Mitsuba 0.5 [Jakob 2010]. All timings in this section are computed on a 2 GHz Quad-Core Intel Core i5 MacBook Pro with 32 GB memory. See Figure 9 for a visual comparison. Both Photon Mapping and Practical Path Guiding compute underwater caustics with a good quality. For this specific scene, our method is much faster for computing caustics, with a 50 to 300 times speedup.

Fig. 9. Comparison between our algorithm (left) and Photon Mapping (center) and Practical Path Guiding without denoising (right). Both methods render good quality caustics; our algorithm is much faster for the caustics on this specific scene.

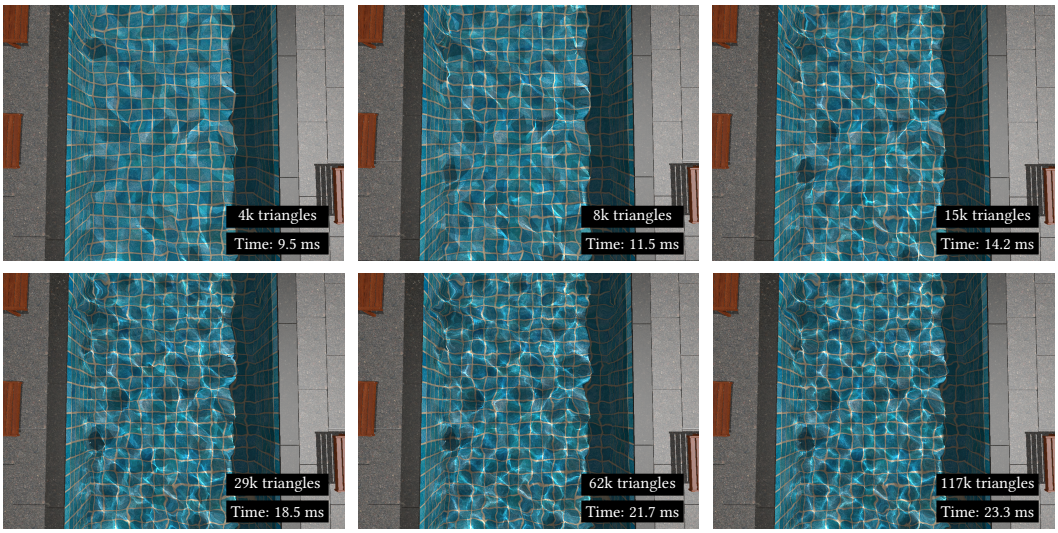


Fig. 10. Influence of the number of triangles for the water surface on the aspect of caustics.

5.3 Influence of scene complexity

Scene complexity is the main parameter when estimating real-time ray-tracing algorithms. We have designed a series of test scenes, all with the same geometry, except for the water surface, for which we increase the tessellation from a few dozen triangles to more than 2 million. Increasing the complexity of the water surface has an influence on the appearance of the caustics, up to a certain point: after 100,000 triangles, the differences are no longer visible when viewing the pool from the top (see Figure 10). This limit corresponds to triangles that are roughly 5 cm wide. When viewing the caustics at the bottom of the pool, as in Figure 13 and 14, increasing the number of triangles has a visible impact on quality even at very high tessellation levels.

Figure 11 shows the total computation cost per frame when running our algorithm, including animation, rendering, and computing caustics. To estimate only the cost induced by producing the caustics using our algorithm, we calculate the computation time per frame without refraction and caustics computation and take the difference. These times are computed with the viewing point

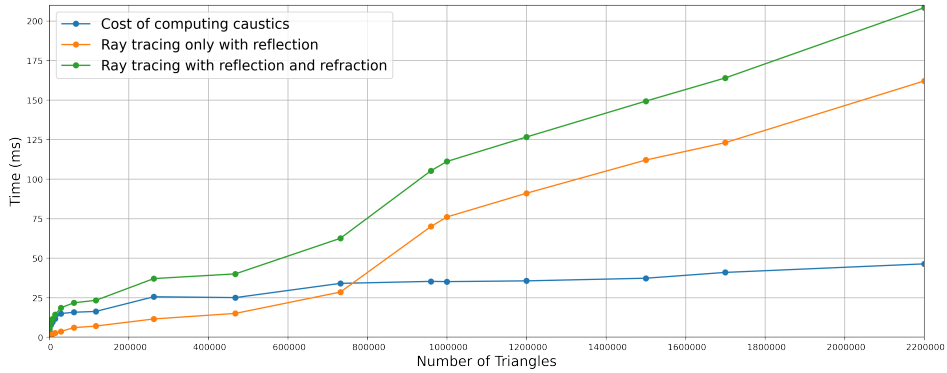


Fig. 11. Computation time per frame for our algorithm (green), compared to rendering a frame without refraction and caustics (orange). The difference (blue) corresponds to the cost of computing caustics using our algorithm. This cost increases with the number of triangles, but much slower than the cost of ray-tracing the entire image. The number of triangles on the horizontal axis corresponds to the tessellation of the water surface.

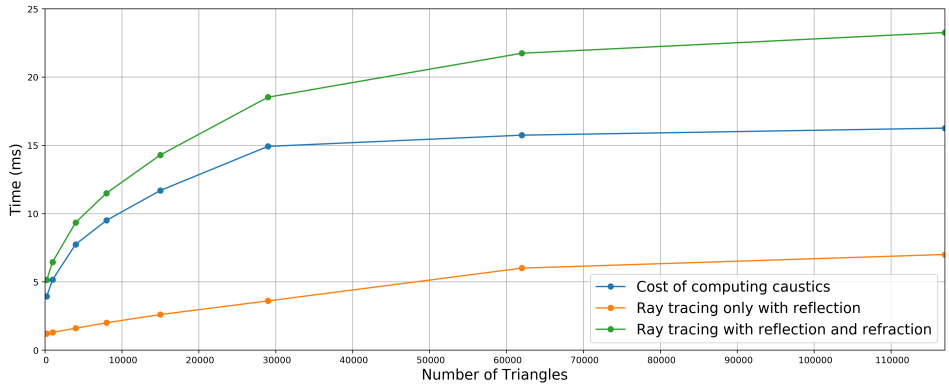


Fig. 12. Computation time per frame for our algorithm (green), compared to rendering a frame without refraction and caustics (orange), focusing on low-resolution scenes (below 117,000 triangles). The number of triangles on the horizontal axis corresponds to the tessellation of the water surface.

outside the pool. With a point of view from inside the water, these times increase, but the increase of time when the number of triangles increases is similar.

Figure 12 shows the same data but focuses on low-resolution scenes with water tessellation levels below 117,000 triangles. There is a cost in using our algorithm to compute caustics; it is relatively important for low-resolution scenes. That extra cost is unavoidable: we are computing more information (both reflection *and* refraction), and we have to shoot one more shadow ray to compute the incoming light at the point where the refracted path leaves the surface. The overall cost remains manageable: below 25 ms per frame for our scenes.

Interestingly, as we further increase the number of triangles, the cost associated specifically with computing caustics increases much slower than the cost for ray-tracing the scene. At 1 million triangles, only a third of the total computation time is used to compute the caustics, and two-thirds for ray-tracing the rest of the image. Besides computing illumination at the new point, the cost of

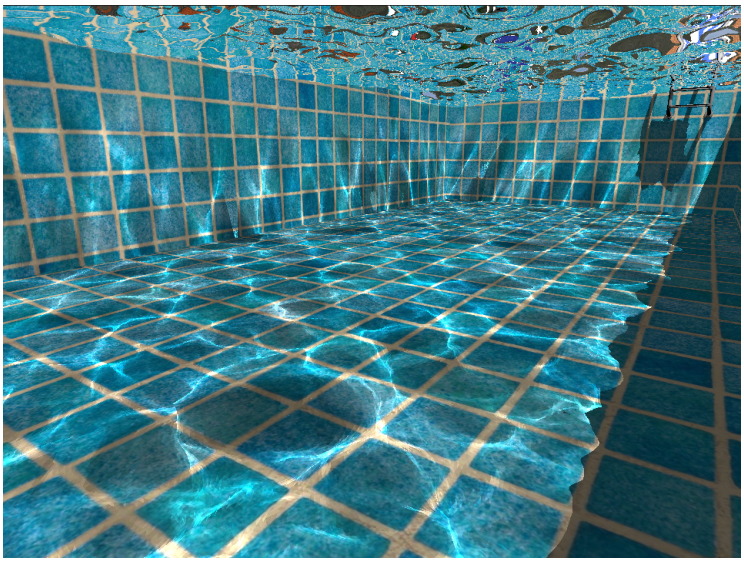


Fig. 13. Caustics seen from under the water of a pool. Image generated using a water plane of 732,000 triangles. Time: 82.4ms

our algorithm is related to the number of triangles we have to cross from the starting point to the solution, including the triangles generated by the subdivision step. As we increase the tessellation levels, we have to cross more triangles before we reach the triangle where the solution is, but there will be fewer subdivision steps once we reach it.

5.4 Temporal consistency

A key advantage of our method is that it produces noise-free images and is temporally consistent. The companion video shows our method running on an animated scene, with an animated point of view. As can be seen, the images have no noise, and there is no flickering.

5.5 Additional Results

So far, we have only shown results and comparisons on scenes with a viewing point outside of the water, but it is interesting to see how our algorithm behaves underwater. Figure 13 illustrates this situation, showing that the caustics are rendered correctly and that our method also works in this scenario. For this point of view, we also added a comparison without the textures to be able to appreciate only the effect of the caustics, as can be seen in Figure 14. Here, we compare our method implemented on CPU, using 1 SPP and the reference SMS [2020] also on CPU, with their unbiased variant using 256 SSP. We can see that the caustics generated by our method correspond to those of the reference method.

Our algorithm works with all scenes that include a water surface, regardless of the underwater terrain. Figure 15 shows the results of a lagoon with irregular topography. As can be seen, the caustics are generated correctly also in this type of scene. It also can handle more complicated scenes, such as when objects float in the water. Figure 15b shows an example of this, where we can appreciate the caustics on the bottom of the lagoon and the shadows of the fish floating in the water.

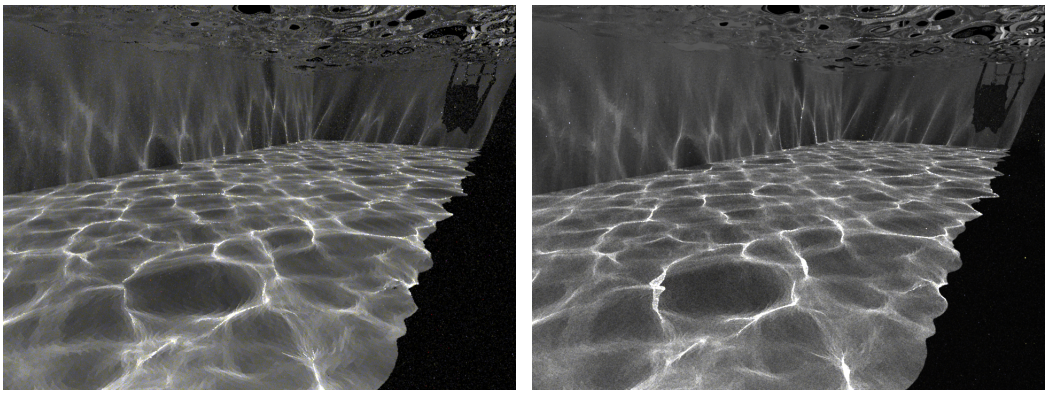
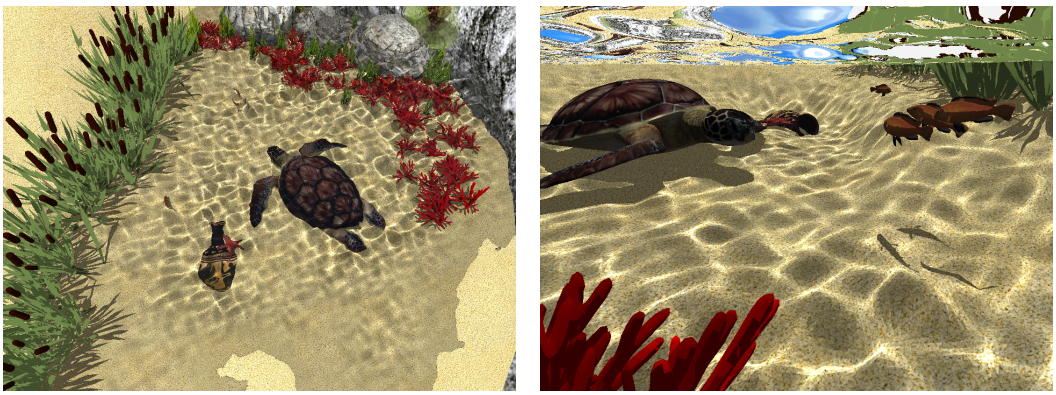


Fig. 14. Visual comparison between our algorithm (left) with SMS (right) showing only the generated caustics (both algorithms running on CPU). Using a water plane of 732,000 triangles.



(a) (b)

Fig. 15. Results of our algorithm in an irregular topography, viewed from top (left) or underwater (right). See also the companion video. Our method handles occluders between the floor and the surface, such as the turtle and floating fish.

6 LIMITATIONS AND DISCUSSION

The strength of our algorithm is the coplanarity constraint, which restrains the space where we search for solutions: once we have a point on the curve, we just need to follow that curve until we find a solution. But we need that starting point. This is not an issue for refraction, where we can use the intersection between (PL) and the interface, but it is an issue for reflection, where we do not have an easy way to find a point that satisfies the constraint. The easiest method is to find a point M such that $\boldsymbol{n}_M$ is aligned with (MP) or (ML), but that requires a position-normal hierarchy, and in our experiments is too expensive for interactive rendering.

Our algorithm only finds one refracted path for a given P. Zeltner et al. [2020] show that there can be multiple paths, especially with reflection on curved surfaces. Our algorithm is comparable to their biased algorithm: we trade accuracy in exchange for speed and reduced noise in the image. Both of these properties are important for real-time rendering of animated scenes, where noise would result in temporal artefacts, but would be an issue for high-quality offline rendering.

Once we have computed the exit point through the refractive surface, we trace a ray towards the light source to check whether this exit point is in shadow. This strategy fails if there is another intersection with the refractive surface between the exit point and the light source, which can happen if the light source is close to the horizon, or with rough waters.

Like most other Manifold Next-Event Estimation methods, our method computes purely specular caustics, connecting a point on the receiving surface to a point on the light source. It works well with point light sources and specular surfaces, but glossy surfaces and area light sources will likely create noisy results. It could be combined with path guiding methods [Li et al. 2022] around the representative specular path for these scenes.

7 CONCLUSIONS

We have presented an algorithm to compute light paths through a refractive specular interface. Our algorithm is based on a reduction of the dimensionality of the problem, using a coplanarity constraint. This constraint gives us a curve on the refractive interface, and we restrain our search for solutions to that curve. Our algorithm is fast and requires only the geometry of the specular interface, making it ideally suitable to compute caustics through an animated surface. It does not require computing the path derivatives, making it easier to implement inside a constrained environment like the GPU.

In future work, we want to extend this approach to reflection and to multiple specular interfaces, as well as glossy surfaces using a combination with path guiding. We also want to explore using half-resolution computations for the caustics part, to reduce the impact on computation time.

ACKNOWLEDGMENTS

The lagoon scene was created by Laurence Boissieux. All objects in the swimming pool scene are licensed under CC-BY-4.0 (CC Attribution): Swimming Pool by CN Entertainment, Pool chairs pack by assetfactory, Beach Ball by Maggatron Floats by Ainss100.

REFERENCES

Johannes Hanika, Marc Droske, and Luca Fascione. 2015. Manifold Next Event Estimation. *Computer Graphics Forum* 34, 4 (July 2015), 87–97. https://doi.org/10.1111/cgf.12681

Wenzel Jakob. 2010. Mitsuba renderer. http://www.mitsuba-renderer.org.

Wenzel Jakob and Steve Marschner. 2012. Manifold exploration: a Markov Chain Monte Carlo technique for rendering scenes with difficult specular transport. *ACM Transactions on Graphics* 31, 4 (Aug. 2012), 1–13. https://doi.org/10.1145/2185520.2185554

Henrik Wann Jensen. 1996. Global Illumination using Photon Maps. In *Rendering Techniques '96*, Xavier Pueyo and Peter Schröder (Eds.). Springer Vienna, Vienna, 21–30. https://doi.org/10.1007/978-3-7091-7484-5_3

He Li, Beibei Wang, Changhe Tu, Kun Xu, Nicolas Holzschuch, and Ling-Qi Yan. 2022. Unbiased Caustics Rendering Guided by Representative Specular Paths. In *SIGGRAPH Asia 2022 Conference Papers*. Article 41, 8 pages. https://doi.org/10.1145/3550469.3555381

Guillaume Loubet, Tizian Zeltner, Nicolas Holzschuch, and Wenzel Jakob. 2020. Slope-Space Integrals for Specular Next Event Estimation. *Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 39, 6 (Dec. 2020). https://doi.org/10.1145/3414685.3417811

Don Mitchell and Pat Hanrahan. 1992. Illumination from Curved Reflectors. *SIGGRAPH Comput. Graph.* 26, 2 (jul 1992), 283–291. https://doi.org/10.1145/142920.134082

Thomas Müller, Markus Gross, and Jan Novák. 2017. Practical Path Guiding for Efficient Light-Transport Simulation. *Computer Graphics Forum* 36, 4 (July 2017), 91–100. https://doi.org/10.1111/cgf.13227

Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. 2019. Mitsuba 2: A Retargetable Forward and Inverse Renderer. *Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 38, 6 (Dec. 2019). https://doi.org/10.1145/3355089.3356498

Eric Veach and Leonidas J. Guibas. 1997. Metropolis light transport. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques - SIGGRAPH '97*. ACM Press, Not Known, 65–76. https://doi.org/10.1145/258734.258775

Bruce Walter, Shuang Zhao, Nicolas Holzschuch, and Kavita Bala. 2009. Single Scattering in Refractive Media with Triangle Mesh Boundaries. *ACM Trans. Graph.* 28, 3, Article 92 (jul 2009), 8 pages. https://doi.org/10.1145/1531326.1531398

Xueqing Yang and Yaobin Ouyang. 2021. Real-Time Ray Traced Caustics. In *Ray Tracing Gems II.* Apress Berkeley, CA, 469–497.

Tizian Zeltner, Iliyan Georgiev, and Wenzel Jakob. 2020. Specular manifold sampling for rendering high-frequency caustics and glints. *ACM Transactions on Graphics* 39, 4 (Aug. 2020). https://doi.org/10.1145/3386569.3392408