Accurate computation of single scattering in participating media with refractive boundaries

Nicolas Holzschuch

To cite this version:

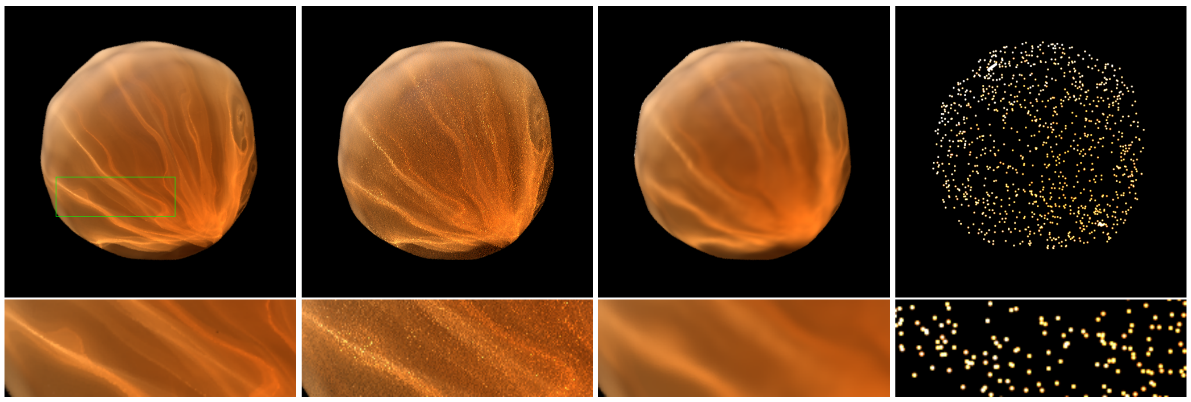
Nicolas Holzschuch. Accurate computation of single scattering in participating media with refractive boundaries. Computer Graphics Forum, Wiley, 2015, 34 (6), pp.48-59. 10.1111/cgf.12517 . hal-01083246v2

HAL Id: hal-01083246
https://hal.inria.fr/hal-01083246v2

Submitted on 25 Jun 2015

Accurate computation of single scattering in participating media with refractive boundaries

N. Holzschuch[†]



(a) Our algorithm (169 s) **(b)** Point sampling [WZHB09] (27 samples, 170 s) **(c)** Photon mapping (650 000 photons, 167 s) **(d)** bidirectional path tracing (520 samples per pixel, 171 s)

Figure 1: Single scattering: comparison between our algorithm and existing methods (equal computation time) on a translucent sphere illuminated by a point light source from behind. For Bi-Directional Path-Tracing, we replaced the point light source with a sphere so the algorithm actually produces a picture (its angular size is ≈ 3.2°).

Abstract
Volume caustics are high-frequency effects appearing in participating media with low opacity, when refractive interfaces are focusing the light rays. Refractions make them hard to compute, since screen locality does not correlate with spatial locality in the medium. In this paper we give a new method for accurate computation of single scattering effects in a participating media enclosed by refractive interfaces. Our algorithm is based on the observation that although radiance along each camera ray is irregular, contributions from individual triangles are smooth. Our method gives more accurate results than existing methods, faster. It uses minimal information and requires no precomputation or additional data structures.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing and texture

1. Introduction

Light traversing participating media interacts with it, being either scattered or absorbed by the medium. Combined with light concentration from occlusion or refractive interfaces, scattering results in high frequency effects, such as "god rays" or underwater caustics.

Observing these effects from outside the participating media adds another refractive interface between the camera and the media. This distorts the scattering effects, resulting in beautiful but challenging to render effects as shown in Figure 1. The two refractive interfaces between the light source

[†] INRIA Grenoble - Rhône-Alpes and LJK CNRS / Université de Grenoble

and the camera make it difficult to find light paths going from the light source to the camera after exactly one scattering event.

In this paper, we focus on objects enclosed by a refractive interface defined by triangles with interpolated vertex normals, a representation close to ubiquitous in computer graphics. For a given camera ray, refracted at the interface, the radiance is equal to the integral of radiance received along this ray and scattered in the camera direction. This out-scattered radiance varies quickly along the ray, making it difficult to compute the integral accurately. We observe that the radiance caused by an individual triangle on the surface varies smoothly; discontinuities correspond to triangle edges.

We exploit this property in a new algorithm for computing single scattering effects with refractive interfaces. For each triangle on the surface, we quickly identify whether it is making a contribution to this specific camera ray, and on which segment of the ray. We then sample regularly inside this segment. Our method works for highly tesselated objects and is significantly faster than existing methods while providing better quality results. It has been designed to be easily inserted into existing renderers, with minimal modifications. It does not require additional data structures or pre-processing.

We review previous work on single and multiple scattering in the next section. We describe our algorithm in details in Section 3. We study the behaviour of our algorithm and compare with previous work in Section 4. Finally, we conclude and present avenues for future work in Section 5.

2. Previous work

Pegoraro and Parker [PP09] provided an analytical solution for single scattering from a point light source in an homogeneous participating media. Their approach is useful for scenes filled with smoke or dust, with no refractive interfaces.

Volume caustics are caused by light being refracted at the interface between the refracted media and air. The most striking examples are caustics in turbid water. Nishita and Nakamae [NN94] described an algorithm to compute these caustics by creating a light shaft for each triangle on the surface, then accumulating the contributions from these light shafts using the accumulation buffer. Iwasaki et al. [IDN01] improved the method by using color blending between vertex values inside the light shaft. Ernst et al. [EAMJ05] further improved the light shaft method using non-planar shaft boundaries and computing illumination inside the shaft with fragment shaders. All these work focused on a single refractive interface, displaying volume caustics as seen from inside the media.

Hu et al. [HDI*10] traced light rays inside the participating media, after up to two specular events (reflections or refractions). They accumulate radiance from these light rays in a second step. Their method produces volume caustics interactively, but with no refractive interface between the camera and the volume caustics.

Ihrke et al. [IZT*07] presented a method to compute light transport in refractive objects by modelling the propagation of the light wavefront inside the object. Their method is interactive and handles objects made of inhomogeneous materials. It relies on a voxelised representation of light distribution inside the object. Sun et al. [SZS*08] also used a voxelized representation of light inside the object, but computed scattered light using photon propagation. For both methods, the resolution of the voxel grid limits the spatial resolution of volume caustics.

Sun et al. [SZLG10] introduced the *line space gathering* method. They first trace rays from the light source and store them as points in Plücker coordinates. Then, for each camera ray, they find the closest light rays using Plücker space and compute their scattering contributions. Their technique produces high quality results, both inside and outside translucent objects, taking into account refraction. The data structures used to store camera and light rays in Plücker space uses approximately 400 MB.

Walter et al. [WZHB09] used a Newton-Raphson iterative method to find all paths connecting a sample point inside the participating media to the light source through a refractive interface. They place sample points along camera rays inside the translucent object to compute volume caustics. They used a position-normal tree to cull triangles in the search. Our method can be seen as an extension of theirs; we remove the need for additional data structure and place sample points optimally for each triangle.

Jarosz et al. [JZJ08] introduced the *beam radiance estimate* for efficient rendering of participating media with photon mapping. It starts as conventional volumetric photon mapping, storing photons inside the participating media. In the gathering step, camera rays are represente as beams, gathering contributions from all photons around the beam. The technique was later extended to *photon beams*, by Jarosz et al. [JNSJ11]: light entering the participating media is also stored as a beam of light. In the gathering step, they compute contributions from light beams to camera beams, as well as from photons to camera beams. *Progressive photon beams*, by Jarosz et al. [JNT*11], extends this idea by using beams of varying width in successive rendering steps.

3. Single scattering computations

We consider an object, filled with an homogenous participating media, of refractive index η, mean free path $\ell = \sigma_T^{-1}$, albedo α and phase function p (see Figure 2). The boundary of the translucent object is a smooth dielectric interface. We assume that it is modelled as triangles with interpolated vertex normals.

We place ourselves within a path tracing framework: we

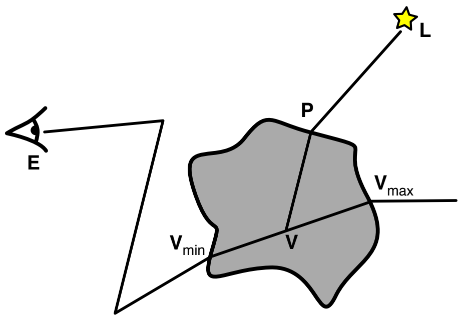


Figure 2: We consider a translucent object, under illumination from a light source **L**. We isolate a segment from the camera path crossing the object, and compute single scattering effects on this segment.

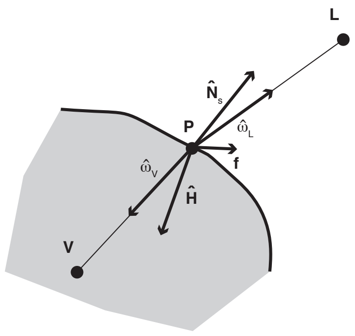


Figure 3: At point **P**, we define the half-vector $\widehat{\mathbf{H}}$, and compare it with the shading normal $\widehat{\mathbf{N}}_s$. Our goal is to find zeros of the function $\mathbf{f} = \widehat{\mathbf{H}} + \widehat{\mathbf{N}}_s$.

have a camera path, made of several connected segments. We take one of these segments, crossing through the translucent object, from an entry point $\mathbf{V}_{\min}$ to an exit point $\mathbf{V}_{\max}$.

We also take a point sample **L** on the light source: either its position for a point light source, or a random sample for an area light source.

Our goal is to compute the integral of single scattering effects from **L** on this specific segment of the camera path. This contribution will be combined with other lighting effects computed by the renderer.

3.1. Point sampling algorithm

First, we briefly review the algorithm of Walter et al. [WZHB09]. They place several sample points **V** along each camera ray. For each sample point, they compute all paths connecting **V** to the light source **L** (see Section 3.1.1), then compute their contribution and sum them (see Section 3.1.2). For clarity, we reuse their notations. $\widehat{\mathbf{v}}$ is a normalized vector: $\widehat{\mathbf{v}} = \mathbf{v}/\|\mathbf{v}\|$.

3.1.1. Iterative search for light paths

Given an individual triangle with interpolated vertex normals, a sample point **V** and a light source sample **L** the problem is to find all paths connecting **L** to **V** while following Snell's law. The key idea is to transform this into the search for zeros of a function **f**. Given a point **P** on the triangle, with shading normal $\widehat{\mathbf{N}}_S$, we define the normalized half-vector $\widehat{\mathbf{H}}$ and the function **f** (see Figure 3):

$$\begin{aligned}
d_V &= \|\mathbf{V} - \mathbf{P}\| && (1)\\
d_L &= \|\mathbf{L} - \mathbf{P}\| && (2)\\
\widehat{\omega}_V &= (\mathbf{V} - \mathbf{P})/d_V && (3)\\
\widehat{\omega}_L &= (\mathbf{L} - \mathbf{P})/d_L && (4)\\
\mathbf{H} &= \eta\widehat{\omega}_V + \widehat{\omega}_L && (5)\\
\widehat{\mathbf{H}} &= \mathbf{H}/\|\mathbf{H}\| && (6)\\
\mathbf{f}(\mathbf{P}) &= \widehat{\mathbf{H}} + \widehat{\mathbf{N}}_S && (7)
\end{aligned}$$

All points **P** such that $\mathbf{f}(\mathbf{P}) = \mathbf{0}$ correspond to a path connecting **V** and **L**, with the refraction angle following Snell's law.

To find the zeros of **f**, Walter et al. [WZHB09] used a Newton-Raphson iterative method, with one small change: since point **P** on the triangle uses barycentric coordinates (a, b), **f** is a function from $\mathbf{R}^2$ into $\mathbf{R}^3$. The Jacobian J of **f** is a non-square matrix and thus not invertible. They used the pseudo-inverse of J, J^+:

$$J^+ = (J^\intercal J)^{-1} J^\intercal \quad (8)$$

The iterative Newton-Raphson method becomes:

$$\mathbf{P}_{n+1} = \mathbf{P}_n - J^+\mathbf{f}(\mathbf{P}_n) \quad (9)$$

For this specific problem, it converges rapidly to a solution.

3.1.2. Contribution from each path

Once a path connecting **V** and **L** has been found, its contribution to the pixel is computed:

$$\text{contribution} = \frac{I_e F A p}{D} \quad (10)$$

where I_e is the intensity of the light source sample **L**, F is the Fresnel factor at the entry point $\mathbf{V}_{\min}$ and exit point **P**, A is the volume attenuation (the integral of $e^{-\sigma_T x}$ along $[\mathbf{V}_{\min}\mathbf{V}]$ and $[\mathbf{VP}]$), p is the phase function at **V** and D is the distance correction factor.

In the absence of participating media and refractive interface, D is the square of the distance between V and L, $D = \|\mathbf{L} - \mathbf{V}\|^2 = (d_V + d_L)^2$. With refractive media and constant geometric normal $\widehat{\mathbf{N}}_g$, D has a simple expression:

$$D = (d_V + \eta d_L)\left(\frac{|\widehat{\omega}_L \cdot \widehat{\mathbf{N}}_g|}{|\widehat{\omega}_V \cdot \widehat{\mathbf{N}}_g|} d_V + \frac{|\widehat{\omega}_V \cdot \widehat{\mathbf{N}}_g|}{|\widehat{\omega}_L \cdot \widehat{\mathbf{N}}_g|} d_L\right) \quad (11)$$

With shading normals, D has a more complicated expression, based on ray differentials at the interface [Ige99]. To compute D, Walter et al. take two vectors perpendicular to ω_V and each other: $\mathbf{u}_\perp$ and $\mathbf{u}_\|$, then compute how $\mathbf{L}$ changes when ω_V is perturbed along these vectors. D is the cross-product of these derivatives:

$$D = \left\| \frac{d\mathbf{L}}{d\mathbf{u}_\perp} \times \frac{d\mathbf{L}}{d\mathbf{u}_\|} \right\| \tag{12}$$

A complete expression of D can be found in [WZHB09]. We refer the interested readers to this article for more information.

3.1.3. Pruning triangles

The Newton-Raphson method converges with few iterations for each sample point $\mathbf{V}$ and each triangle. In theory, the search should be done for all triangles and all sample points, but that would be too costly. Walter et al. [WZHB09] used several methods to prune the triangles used in the search:

- Sidedness agreement: $\mathbf{V}$ must be in the negative half-space of the triangle, and $\mathbf{L}$ in the positive half-space, for both the geometric normal $\widehat{\mathbf{N}}_g$ and the shading normal $\widehat{\mathbf{N}}_s$.
- Spindle test: the angle between the incoming and the refracted ray is between $\frac{\pi}{2} + \arcsin(1/\eta)$ and π. For a given segment $[\mathbf{VL}]$, this restricts the potential solutions $\mathbf{P}$ to inside a surface of revolution around $[\mathbf{VL}]$, called the *spindle*.

The spindle test doesn't depend on triangle normals and can be used to prune large parts of the scene. For the sidedness agreement, Walter et al. [WZHB09] used a position-normal tree built from the triangles upward.

Together, these pruning methods reduced the computation time to something acceptable, a few minutes on a 8-core Xeon.

3.1.4. Numerical issues

The algorithm described by Walter et al. [WZHB09] converges quickly for objects made of flat triangles and moderately curved surfaces. The authors reported numerical issues for more complex scenes such as the bumpy sphere (see Figure 1):

- They have to place more samples $\mathbf{V}$ along each camera ray, up to 128 samples for the bumpy sphere,
- Some samples had extremely large values, which resulted in noise in the picture. These values had to be clamped in order to reduce the noise.

Our experimental study (Section 3.2.1) explains these numerical issues: the function being integrated is highly irregular, with many spikes (see Figure 5). Sample points, placed randomly or regularly, are likely to miss the spikes. If one sample point happens to hit a narrow spike, it will receive disproportionate importance.

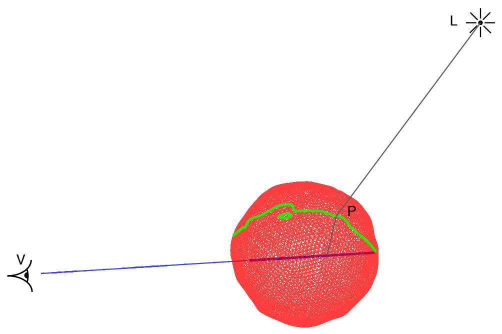


Figure 4: The set of points $\mathbf{P}$ corresponding to a refracted camera ray inside the object defines a curve (in green) on the surface.

3.2. Our algorithm

3.2.1. Motivation: study of a single camera ray

For a better understanding of single scattering effects with refraction, we ran an experimental study of radiance along a single camera ray. It is refracted as it enters the translucent object, then travels in a straight line until it exits the object. We placed a large number of sample points $\mathbf{V}$ along the refracted camera ray. The set of corresponding points $\mathbf{P}$ defines a curve on the surface of the object (see Figure 4). This curve is implicitly defined by $\mathbf{f} = 0$. It is irregular and can be made of several disconnected components: it is piecewise continuous.

Figure 5, left, shows the radiance reaching the camera for all sample points along the ray, taking into account attenuation, phase function and ray differentials. It is highly irregular, with many spikes of high value, defined on a small interval.

Zooming in on a small part of this curve (Figure 5, right), we observe that it is a combination of slowly varying contributions, with sharp discontinuities. These discontinuities correspond to triangle boundaries. This our key observation: the contribution from a single triangle to the outgoing radiance is a smooth function; discontinuities occur only as the curve enters and exits the triangle.

Several triangles can be contributing to the same sample point on the refracted ray: there is not a one-to-one mapping between the ray and the curve. For all points on the interval on Figure 5, right, there are at least two triangles connecting this point.

Note that the spikes do *not* correspond to an infinite amount of light over an infinitely small sampling area, but to a very large, but finite, amount of light over a small, but measurable, area. Despite the name "volume caustics", these are not, technically, caustics, since they are not points of infinite energy.

The pixel value is the integral of the radiance values at

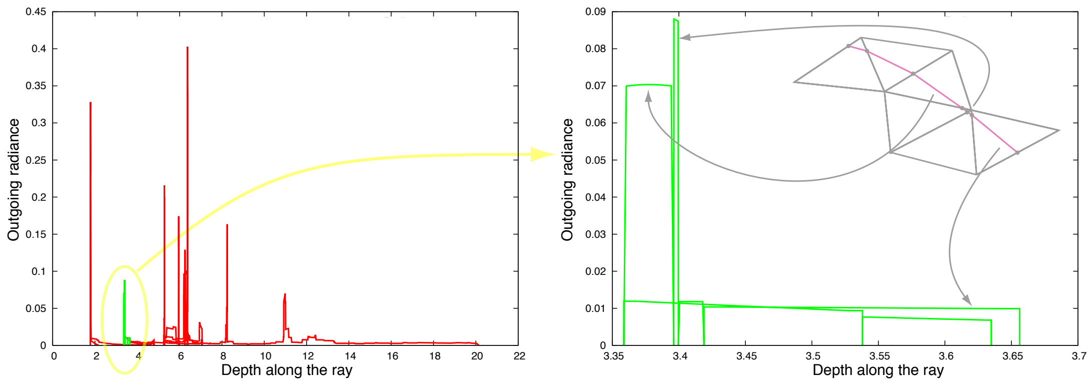


Figure 5: (left) Outgoing radiance, measured along a refracted ray inside the material. The radiance values are highly irregular: computing the integral accurately would requires many samples. If we isolate contributions from each triangle (right), each of them is smooth. Discontinuities occur only at the triangle boundaries.

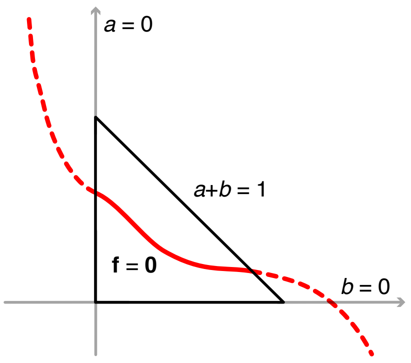


Figure 6: $\mathbf{f} = \mathbf{0}$ defines a curve in the triangle plane, parameterized by t. We search for intersections between this curve and the triangle (defined in barycentric coordinates).

all sample points, the area under the curve on Figure 5, left. Smooth parts of the curve (e.g. from depth = 12 to depth = 20) can contribute more than all the spikes together. But the spikes are making it difficult for point sampling method: to avoid noise, the number of samples should be inversely proportional to the minimum width of the spikes.

3.2.2. Interval analysis along the ray

Based on this observation, the key idea of our algorithm is the following:

> For each individual triangle, identify the limits of its contribution on the $[\mathbf{V}_{\min}, \mathbf{V}_{\max}]$ segment, using Newton-Raphson iterative methods. Once we have these limits, sample only inside the interval. This reduces the number of samples required.

We use the same function $\mathbf{f}$ defined in Equations 1 to 7, but with a third parameter, t, the depth of point $\mathbf{V}$ along the refracted ray. t is limited in range by the entry and exit points, $\mathbf{V}_{\min}$ and $\mathbf{V}_{\max}$, corresponding to $t_{\min}$ and $t_{\max}$ respectively:

$$\mathbf{V} = \mathbf{V}_{\min} + t\frac{\mathbf{V}_{\max} - \mathbf{V}_{\min}}{||\mathbf{V}_{\max} - \mathbf{V}_{\min}||} \tag{13}$$

$\mathbf{f}$ is now a function of three parameters: t and the barycentric coordinates (a, b) of point $\mathbf{P}$ on the triangle plane. $\mathbf{f} = \mathbf{0}$ is a polynomial equation of degree 6 (see Appendix A). It defines a curve parameterized by t in the plane (a, b). We need the intersection between this curve and the triangle defined by: $a \geq 0, b \geq 0, a + b \leq 1$ (see Figure 6). The coordinates in t of these intersections will give us the limits of the contribution for this triangle.

We compute the intersection between the curve $\mathbf{f} = \mathbf{0}$ and the triangle edges using Newton-Raphson iterative methods. For this, we need the Jacobian of $\mathbf{f}$:

$$J = \begin{pmatrix} \frac{\partial \mathbf{f}}{\partial t} & \frac{\partial \mathbf{f}}{\partial a} & \frac{\partial \mathbf{f}}{\partial b} \end{pmatrix} \tag{14}$$

We compute it using repeated applications of the following rule: the derivative of a normalized vector $\widehat{\mathbf{v}} = \mathbf{v}/||\mathbf{v}||$ with respect to any of its parameters q is:

$$\frac{\partial \widehat{\mathbf{v}}}{\partial q} = \frac{1}{||\mathbf{v}||}\left(\frac{\partial \mathbf{v}}{\partial q} - \left(\widehat{\mathbf{v}} \cdot \frac{\partial \mathbf{v}}{\partial q}\right)\widehat{\mathbf{v}}\right) \tag{15}$$

Note that ω_L and $\mathbf{N}_S$ are only functions of (a, b), thus their derivative with respect to t is null.

$\mathbf{f}$ is a function of 3 parameters. We are looking for a point on the edge of the triangle, which removes one degree of freedom: one of the barycentric coordinate is either equal to 0 (for the edges defined by $a = 0$ and $b = 0$) or connected to the other (for the edge defined by $a + b = 1$). We solve for the remaining two parameters: t and the other barycentric coordinate.

We use the same technique as before for finding zeros of $\mathbf{f}$, with the pseudo-inverse of the Jacobian. The only difference is that we start with a 3×3 matrix, then convert it into a 3×2

matrix by removing the column corresponding to the fixed parameter, then compute the pseudo-inverse of this matrix.

Once we have an intersection, we compute the tangent to the curve using the Jacobian of $\mathbf{f}$ (see Section 3.2.3 for details). We use it to tell whether the value of t we found is an upper or a lower bound. We update the interval $[t_{\min}, t_{\max}]$ accordingly:

- If t is an upper bound, $t_{\max} \leftarrow t$
- If t is a lower bound, $t_{\min} \leftarrow t$

We treat all boundaries of the triangle, in a loop, until we either have an empty interval for t or a segment where the curve enters and leaves the triangle. If we have an empty interval, we move to the next triangle. If we have a valid segment, we sample it regularly and compute the contribution from each sample point.

Our algorithm is summarized in Algorithm 1. A single loop over the triangle edges is usually sufficient to identify the entry and exit points. The curve occasionally enters and leaves the triangle on the same edge, requiring a second loop.

Although the curve defined on the entire object is complex and does not have a one-on-one correspondance with points on the $[\mathbf{V}_{\min}, \mathbf{V}_{\max}]$ segment, the curve for a given triangle plane is much simpler, and has this one-on-one correspondance. We do not explicitly compute the full curve on the object, but instead its intersection with every triangle.

The next three sections present specific details of our algorithm: how to compute the tangent to the curve and use it to check whether we have an upper or a lower bound for t (Section 3.2.3), a pruning algorithm that does not require complex data structures, but can use a bounding volume hierarchy if one is provided by the renderer (Section 3.2.4), and how we sample inside the restricted interval for t (Section 3.2.5).

```
Start with interval [$t_{\min}, t_{\max}$]
numIntersections = 0;
repeat
    for each triangle edge $E$
        Compute $(t, a, b)$ intersection between $E$ and curve
        Check if value of $t$ is a min or a max (Section 3.2.3)
        Update interval [$t_{\min}, t_{\max}$] accordingly
        If interval is empty: return 0
        If $(a, b)$ inside triangle: numIntersections++;
    end for
until numIntersections = 2;
// Now, we sample the curve segment
Sum = 0
for $t$ regularly sampled in [$t_{\min}, t_{\max}$]
    Find $(a, b)$ such that $\mathbf{f}(t, a, b) = 0$
    Sum += contribution from point $(a, b)$
end for
return Sum
```

Algorithm 1: Computing single-scattering effects

3.2.3. Tangent to the curve

Once we have found a point $(t, a, b)^\intercal$ on the curve of equation $\mathbf{f} = \mathbf{0}$ on the triangle plane, we compute the tangent at this point. We express it as: $(t, a, b)^\intercal + \lambda \mathbf{g}$ $(\lambda \in \mathbf{R})$ (we explain how to compute $\mathbf{g}$ using the Jacobian of $\mathbf{f}$ at the end of this section). We write the coordinates of $\mathbf{g}$ as $(g_t, g_a, g_b)^\intercal$.

We use this tangent:

- to find better starting points for the other triangle edges. We compute the intersections between the tangent and the other triangle edges, and use them as starting points for the Newton-Raphson methods on these edges. This speeds up the computations.
- to decide whether the point corresponds to an upper or lower bound for t. Variations in t are connected to variations in a (resp. b) by the ratio g_t/g_a (resp. g_t/g_b). The sign of this ratio tells us whether we have a minimum or a maximum, depending on the edge:
  - For $a = 0$, t is a maximum if $g_t/g_a < 0$.
  - For $b = 0$, t is a maximum if $g_t/g_b < 0$.
  - For $a + b = 1$, t is a maximum if $g_t/(g_a + g_b) > 0$.

To compute $\mathbf{g}$, we use the linear approximation of $\mathbf{f}$ with the Jacobian:

$$\mathbf{f}(t + \delta t, a + \delta a, b + \delta b) \approx \mathbf{f}(t, a, b) + J \begin{pmatrix} \delta t \\ \delta a \\ \delta b \end{pmatrix} \qquad (16)$$

For a point on the curve, $\mathbf{f}(t, a, b) = 0$. Equation 16 defines the tangent to the curve:

$$J \begin{pmatrix} \delta t \\ \delta a \\ \delta b \end{pmatrix} = \mathbf{0} \qquad (17)$$

The Jacobian J at a point on the curve is thus of rank at most 2, and its kernel is the tangent to the curve. If J is of rank 2, there are at least two columns in J such that their cross-product is non-null. This cross-product $\mathbf{g}$ is the basis for the kernel of J and the direction of the tangent to the curve (see, for example, Reeder [Ree11]). If J was of rank 1 or 0, then locally the tangent would not be defined; this has never happened in all our test scenes.

3.2.4. Pruning triangles and restricting the interval

We developped two strategies to restrict the search by quickly rejecting triangles:

- One is based on the spindle test and uses only the geometric position of objects; it can be applied to both triangles and bounding volumes.
- The other is based on the shading normals; it can be applied only to triangles.

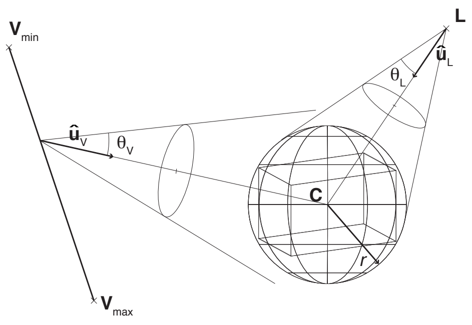


Figure 7: Based on the bounding sphere $(\mathbf{C}, r)$ of the current hierarchical level, we build bounding cones for $\widehat{\omega}_V$ and $\widehat{\omega}_L$.

Both use the fact that the interval for $\mathbf{V}$ is restricted by the entry and exit points of the refracted camera ray, $[\mathbf{V}_{\min}, \mathbf{V}_{\max}]$.

If a bounding volume hierarchy (BVH) is present in the renderer, we use it in a hierarchical descent: starting with the largest node containing the translucent object, we test whether it passes the first test. If yes, we iterate on its descendants. If not, we stop the search. We keep descending recursively until we reach triangles that have passed the first test. We apply to them the second test based on shading normals.

If the renderer does not have a BVH, we test the triangles directly.

The output of the second test is both a boolean and a validity interval for t. If the boolean is true, we apply Algorithm 1 to this triangle.

For both tests, we build bounding cones for $\widehat{\omega}_L$ and $\widehat{\omega}_V$ (see Figure 7). We then use these cones to discard triangles where it is impossible to have a refracted path. We build these bounding cones using the the bounding sphere of the object (whether it is a triangle or a bounding volume), defined by its center $\mathbf{C}$ and radius r:

- $\widehat{\omega}_L$ is inside a cone of axis $\widehat{\mathbf{u}}_L$ and angle θ_L:

$$\begin{aligned} d_{LC} &= \|\mathbf{L} - \mathbf{C}\| \\ \widehat{\mathbf{u}}_L &= (\mathbf{L} - \mathbf{C}) / d_{LC} \\ \theta_L &= \arcsin(r/d_{LC}) \end{aligned}$$

- $\widehat{\omega}_V$ is bounded by a sweeping cone, whose normalized axis moves from $\widehat{\mathbf{u}}_{V\min}$ to $\widehat{\mathbf{u}}_{V\max}$ and whose angle is θ_V:

$$\begin{aligned} \widehat{\mathbf{u}}_{V\min} &= \frac{\mathbf{V}_{\min} - \mathbf{C}}{\|\mathbf{V}_{\min} - \mathbf{C}\|} \\ \widehat{\mathbf{u}}_{V\max} &= \frac{\mathbf{V}_{\max} - \mathbf{C}}{\|\mathbf{V}_{\max} - \mathbf{C}\|} \\ \theta_V &= \arcsin(r/d_{\min}) \\ \text{where } d_{\min} &= \min\left(\|\mathbf{V} - \mathbf{C}\|\right) \text{ for } \mathbf{V} \in [\mathbf{V}_{\min}, \mathbf{V}_{\max}] \end{aligned}$$

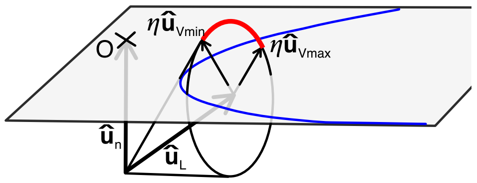


Figure 8: The half-vector $\widehat{\mathbf{H}}$ lives inside a cone of axis $\widehat{\mathbf{u}}_H$ and angle θ_H. $\widehat{\mathbf{u}}_H$ lives on an elliptical cone. This cone intersects the plane perpendicular to $\widehat{\mathbf{u}}_N$ in an hyperbola. Existence of a point such that $\widehat{\mathbf{u}}_N + \widehat{\mathbf{H}} = 0$ becomes a point-to-hyperbola distance in that plane.

Geometry-based test The spindle test tells us that the angle between $\widehat{\omega}_V$ and $\widehat{\omega}_L$ must be between $\frac{\pi}{2} + \arcsin(1/\eta)$ and π. Alternatively, the angle between $\widehat{\omega}_V$ and $-\widehat{\omega}_L$ must be smaller than $\frac{\pi}{2} - \arcsin(1/\eta)$.

We compute the minimum angle between $-\widehat{\mathbf{u}}_L$ and $\widehat{\mathbf{u}}_V$ and compare it with $\frac{\pi}{2} - \arcsin(1/\eta) + \theta_V + \theta_L$. If it is larger, the entire bounding sphere is outside of the spindle, and we can stop the search.

Shading-normal based test When testing against a triangle, we start by computing a sweeping cone bounding the half-vector $\widehat{\mathbf{H}}$. We also build a cone bounding the shading normal, with axis $\widehat{\mathbf{u}}_N$ and angle θ_N and check for intersection between the cones.

The tip of the normalized vector $\widehat{\omega}_L$ lies inside a sphere of radius $r_L = 2\sin(\theta_L/2)$, centered on the tip of $\widehat{\mathbf{u}}_L$ (similarly for $\widehat{\omega}_V$). Thus, the tip of $\mathbf{H} = \eta\widehat{\omega}_V + \widehat{\omega}_L$ is inside a sphere of radius $\eta r_V + r_L$, centered on a moving vector $\eta\widehat{\mathbf{u}}_V + \widehat{\mathbf{u}}_L$.

We denote $H_{\min}$ the minimal value of the norm of $\eta\widehat{\mathbf{u}}_V + \widehat{\mathbf{u}}_L$. The normalized half-vector $\widehat{\mathbf{H}}$ lies inside a cone of axis $\widehat{\mathbf{u}}_H$ and angle θ_H:

$$\begin{aligned} \widehat{\mathbf{u}}_H &= \text{normalize}(\eta\widehat{\mathbf{u}}_V + \widehat{\mathbf{u}}_L) \\ \theta_H &= \arcsin\left((\eta r_V + r_L)/H_{\min}\right) \end{aligned}$$

The axis $\widehat{\mathbf{u}}_H$ lies on a portion of an elliptic cone defined by $\widehat{\mathbf{u}}_L$ and the circle on which $\eta\widehat{\mathbf{u}}_V$ varies (see Figure 8). It is possible to have $\widehat{\mathbf{H}} + \mathbf{N}_S = \mathbf{0}$ only if $-\mathbf{u}_H$ falls inside the cone of axis $\widehat{\mathbf{u}}_N$ and angle $\theta_N + \theta_H$.

Rather than having to compute the intersection of these two cones, we move to the plane perpendicular to $\widehat{\mathbf{u}}_N$, defined by $\mathbf{M} \cdot \widehat{\mathbf{u}}_N = 1$. The intersection between this plane and the cone $(\widehat{\mathbf{u}}_N, \theta_N + \theta_H)$ is a circle, of center $\mathbf{O}$ and radius $\tan(\theta_N + \theta_H)$. The intersection between this plane and the elliptic cone carrying $\mathbf{u}_H$ is a hyperbola. The two cones can intersect only if the curves intersect in the plane: if the minimum distance between $\mathbf{O}$ and the hyperbola is larger than $\tan(\theta_N + \theta_H)$ we discard the triangle.

Restricting the interval for t A hyperbola defined as the intersection between a cone and a plane has two branches.

Only one branch is relevant for us, the one corresponding to $\widehat{\mathbf{u}}_N \cdot \widehat{\mathbf{u}}_H < 0$. The point $\mathbf{V}$ such that $\widehat{\mathbf{u}}_N \cdot \widehat{\mathbf{u}}_H = 0$ defines the boundary between the two branches. We can discard the other part of the interval, which reduces the search interval for t.

3.2.5. Computing the integral

Once we have $t_{\min}$ and $t_{\max}$ for a given triangle, we sample this interval regularly in t. For K samples, we take:

$$t_i = t_{\min} + \frac{i + \frac{1}{2}}{K}(t_{\max} - t_{\min}) \tag{18}$$

For each t_i we compute $\mathbf{V}_i$ using Equation 13, then $\mathbf{P}_i$ on the surface such that $\mathbf{V}_i\mathbf{P}_i\mathbf{L}$ is a valid path following Snell's law. We shoot a shadow ray from $\mathbf{P}_i$ to $\mathbf{L}$. If the points are visible from each other, we add the contribution associated to that path, C_i, to the contribution of the triangle:

$$C_{\text{triangle}} = \frac{t_{\max} - t_{\min}}{K} \sum_i C_i \tag{19}$$

The number K of samples in each interval is a parameter. In practice, we have used $K = 1$ for all our test scenes with no visible impact on quality.

4. Results and comparison

Unless otherwise specified, pictures and timings in this paper were generated on a quad-core Intel Xeon W3250 at 2.66 GHz with 6 GB of memory, running Windows 7. We used the Mitsuba renderer [Jak10] for our algorithm, photon mapping and bi-directional path-tracing, and the original implementation of Walter et al.'s algorithm [WZHB09].

There is no post-processing in any of the pictures: no clamping, no filtering. We compute single scattering effects, along with transmitted and reflected light, as well as internal reflections.

4.1. Equal time comparison

The key advantage of our algorithm is that it provides high quality pictures of single scattering effects quickly. Figures 1 and 9 show a side-by-side comparison between our algorithm and other algorithms for approximately the same computation time. Other algorithms do not provide the same quality:

- **Point sampling**, the algorithm described in [WZHB09], computes an exact solution at several points along the refracted camera ray. With equal time computation, there are too little samples, and point artefacts are clearly visible (see Figures 1b and 9b).
- **Photon mapping** uses the implementation in the Mitsuba renderer, which relies on Beam-Radiance Estimates [JZJ08]. It provides a picture without artefacts, but the individual features characteristic of single scattering have been averaged (see Figures 1c and 9c).

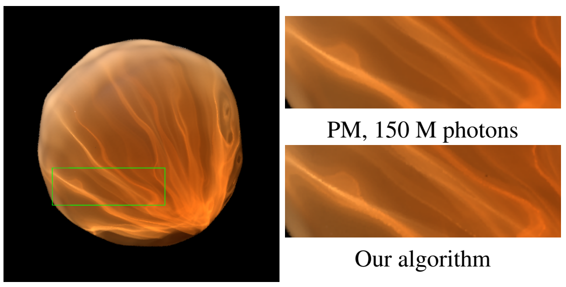


Figure 10: Comparison with a high quality solution: Photon mapping with 150 M photons (3.53 h on a dual Intel Xeon E5345, 2.33 GHz with 48 Gb RAM).

- **Bi-directional path-tracing** also uses the implementation in the Mitsuba renderer. For scattering effects, we connect paths from the light source and paths from the camera inside the medium. For single scattering, the two paths must intersect exactly, which has a zero probability for a point light source. To produce a picture, we replaced it with a spherical light source (see Figure 1d). Picture quality is much lower than for the other methods.

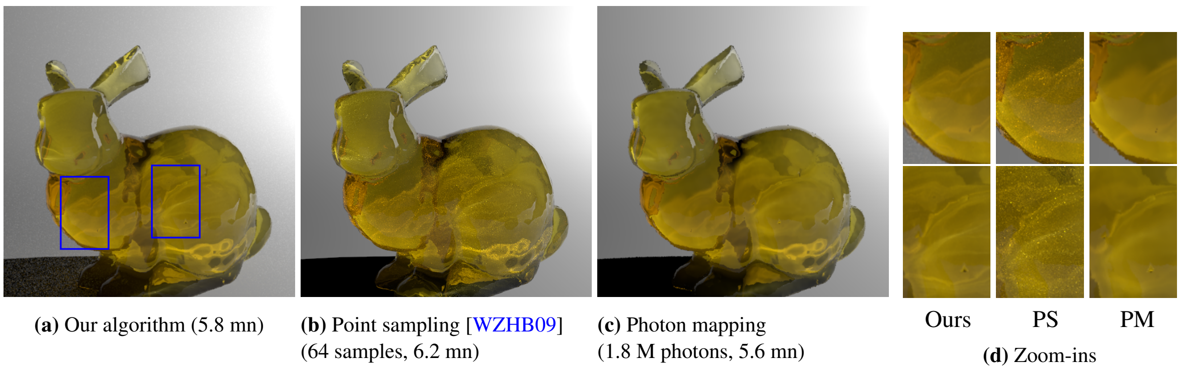
4.2. Validation: comparison with high-quality solution

Given the difference between our results and those of other methods, we need confirmation that we are actually computing the right solution. Photon mapping with a large number of photons provides this validation. The computer we used in our tests ran out of memory after 15 million photons. We used a different computer, with 48 GB of memory, to compute a picture with 150 million photons (see Figure 10). This high quality picture is identical to the one we compute in 169 s, up to the thinner details.

4.3. Equal-quality comparison

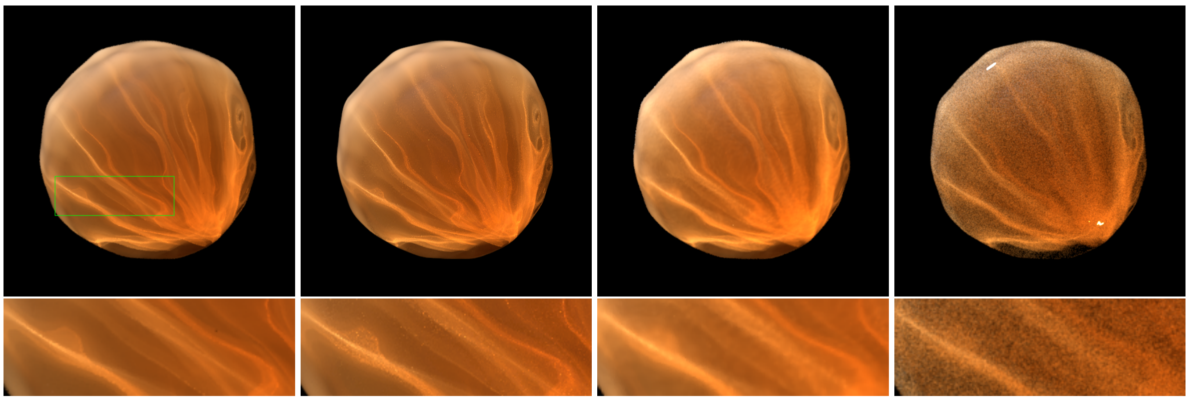
Increasing the number of samples or photons increases the quality of the other algorithms, along with their computation time. Figure 11 shows pictures with approximately the same quality. The computation time is at least 17 times larger, and the image quality is still not as good as ours. Note that bi-directional path tracing also converges to the same solution, but much slower than the other methods.

Equal quality comparisons with photon mapping is difficult because of the memory cost: we are limited to 15 million photons on our testing computer. We allocated all of them to the volume photon map, specifically for single scattering. In a more realistic scenario, part of the photons would be used for surface effects and for multiple scattering, reducing the quality of single scattering simulation. On the other hand, photon mapping can be used for rendering more complex scenarios, that are beyond the scope of our method.



(a) Our algorithm (5.8 mn) **(b)** Point sampling [WZHB09] (64 samples, 6.2 mn) **(c)** Photon mapping (1.8 M photons, 5.6 mn) **(d)** Zoom-ins

Figure 9: Comparison between our algorithm for single-scattering effects and existing methods, for approximately equal time. Bunny model with 16 301 triangles.



(a) Our algorithm (169 s) **(b)** Point sampling [WZHB09] (512 samples, 3 188 s, 18×) **(c)** Photon mapping (15 M photons, 2 940 s, 17×) **(d)** Bi-Directional Path-Tracing (500 K samples per pixel, 331 776 s, 1940×)

Figure 11: Comparison between our algorithm for single-scattering effects and existing methods, for approximately equal quality (Photon Mapping: we used the maximum number of photons for our computer, 15 millions; Bi-Directional Path-Tracing: we use a sphere instead of a point light source).

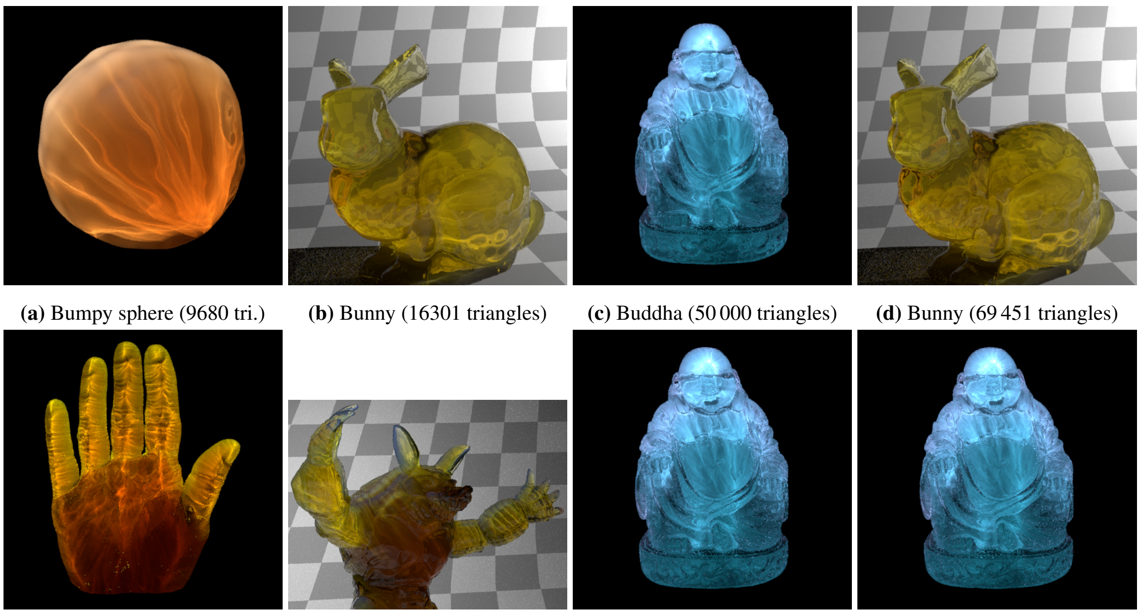
A strong advantage of our algorithm is that it has no additional memory cost: we do not use any data structure beyond what is already present in the renderer: the geometry of the object and the bounding volume hierarchy.

4.4. Computation time

We used five test scenes, for a total of eight different polygon counts: the bumpy sphere from [WZHB09], the Stanford bunny at two different resolutions, a Buddha statue at three different resolutions, the Stanford armadillo and a model of a hand (see Figure 12). Triangle count ranges from 10^4 to 10^6 triangles. For the Buddha and the hand, we show computation times for two different light positions: from the top and from behind. We also tried different light positions for the bumpy sphere and the bunny, but the difference in computation time was within the margin of error.

Figure 13a displays the average computation time for each camera ray that reached the translucent object. This measure is independent of the screen coverage of the model. We show the computation time with and without shadow rays for each sample point. We show separately the average time for internally reflected rays; 40 % to 55 % of the camera rays entering the object result in internal reflections (except for the bumpy sphere with only 20 %):

- The total computation time is dominated by finding the zeros of **f**. Shooting shadow rays accounts for 3 % of computation time. This could be caused, in part, by the simplicity of our test scenes: they contain a large translucent object and few other objects.
- The influence of light position depends on the object shape. For compact objects, such as the sphere, the bunny and the Buddha, it has little influence (at most 10 %).



(a) Bumpy sphere (9680 tri.) **(b)** Bunny (16301 triangles) **(c)** Buddha (50 000 triangles) **(d)** Bunny (69 451 triangles)

(e) Hand (310 000 triangles) **(f)** Armadillo (346 000 triangles) **(g)** Buddha (0.5 million tri.) **(h)** Buddha (1 million triangles)

Figure 12: Our test scenes (by increasing order of complexity).

For branching objects, such as the hand and armadillo, it has a strong influence: computation time is several times smaller than for a compact object with the same triangle count if each branch can be treated independently (armadillo, hand illuminated from behind). It is slower than for a comparable compact object when branches interfere with each other (hand from the top).

- Scattering computations for internal reflections can be either slower or faster than for direct rays, depending on light position. The behavior evolves consistently with object resolution. Internal reflections are focusing computations on part of the object, which can be more or less complex than the average.

Figure 13b displays the same results in log-log space. For compact objects and identical lighting conditions, computation time for each sample evolves roughly as $O(n^{3/4})$, where n is the number of polygons. A smooth curve drawn on a spherical object tesselated into n triangles crosses $O(n^{1/2})$ triangles. This is our lower bound. The upper bound is exploring all triangles: $O(n)$. Complexity is half-way between upper and lower bounds.

Comparing Figures 12b and 12d shows the effect of changing object complexity: caustics become more convoluted. The effect is less visible on the Buddha model (Figures 12c, 12g and 12h) probably because the first model is already of high accuracy.

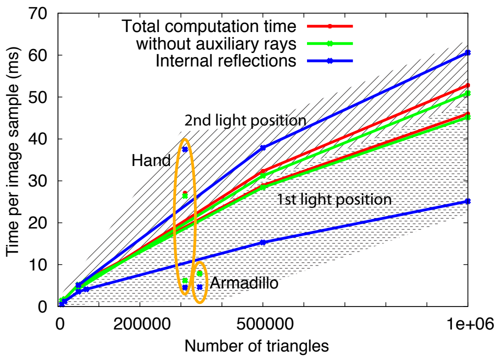
5. Conclusion

We have described a new algorithm for accurate computation of single scattering effects inside a translucent object with refractive boundary. It is an extension of Walter et al.'s algorithm [WZHB09]. Our main idea is to compute the limits of the influence of each triangle over each refracted ray, then sample only inside these limits. Our algorithm proves to be both faster and more accurate than existing methods, with a smaller memory footprint.

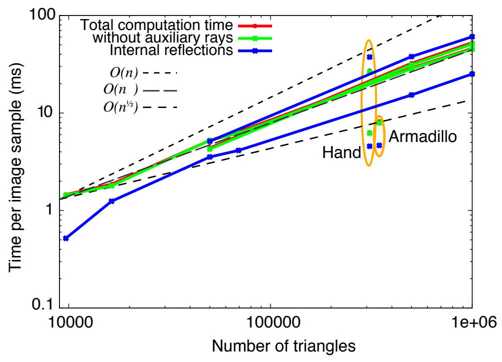
Our method does not require any additional memory structure beyond what is already available in a renderer: the geometry of each triangle, and a bounding volume hierarchy. This makes it easier to implement inside existing renderers, but impacts performance: with adjacency information between triangles, we could reuse computations from neighboring triangles for shared edges. This should reduce computation time by 50 %. Using the results from neighboring pixels as a starting point for the Newton-Raphson iteration would also speed up computations. This information would also help for adaptive anti-aliasing.

Our algorithm does not allow a speed-quality tradeoff. In future work we want to introduce this possibility, based on a hierarchical representation of surface details. This would also be connected to an extension to rough specular interface. We are currently limited to smooth dielectric.

Finally, we want to extend our algorithm to more complex configurations. There are no limitations on the camera ray, but we are restricted to one refractive interface between the



(a) Linear representation



(b) Log-log space. Time per ray evolves approximately as $n^{3/4}$.

Figure 13: Average computation time for each camera ray reaching the object, as a function of scene complexity, for two different light positions. Compact objects (bumpy sphere, bunny, Buddha) behave similarly (connected curves). For branching objects (hand, armadillo), complexity depends on light position.

camera ray segment and the light source. We would like to handle more complex cases, with several interfaces from the light source to the segment.

Acknowledgments

This work was supported in part by a grant ANR-11-BS02-006 "ALTA". The Bunny and Armadillo models are provided by the Stanford Computer Graphics Lab data repository. The bumpy sphere model and the original [WZHB09] implementation were kindly provided by Bruce Walter. The Agnes Hand model is provided courtesy of INRIA by the AIM@SHAPE repository. The Buddha model is provided courtesy of VCG-ISTI by the AIM@SHAPE repository.

References

[EAMJ05] Ernst M., Akenine-Möller T., Jensen H. W.: Interactive rendering of caustics using interpolated warped volumes. In *Graphics Interface* (2005), pp. 87–96. 2

[HDI*10] Hu W., Dong Z., Ihrke I., Grosch T., Yuan G., Seidel H.-P.: Interactive volume caustics in single-scattering media. In *Symposium on Interactive 3D Graphics and Games* (2010), ACM, pp. 109–117. 2

[IDN01] Iwasaki K., Dobashi Y., Nishita T.: Efficient rendering of optical effects within water using graphics hardware. In *Pacific Conference on Computer Graphics and Applications* (2001), pp. 374–383. 2

[Ige99] Igehy H.: Tracing ray differentials. In *SIGGRAPH '99* (1999), ACM, pp. 179–186. 3

[IZT*07] Ihrke I., Ziegler G., Tevs A., Theobalt C., Magnor M., Seidel H.-P.: Eikonal rendering: Efficient light transport in refractive objects. *ACM Transactions on Graphics (proc. Siggraph) 26*, 3 (July 2007), 59:1 – 59:9. 2

[Jak10] Jakob W.: Mitsuba renderer, 2010. `http://www.mitsuba-renderer.org`. 8

[JNSJ11] Jarosz W., Nowrouzezahrai D., Sadeghi I., Jensen H. W.: A comprehensive theory of volumetric radiance estimation using photon points and beams. *ACM Transactions on Graphics 30*, 1 (January 2011), 5:1–5:19. 2

[JNT*11] Jarosz W., Nowrouzezahrai D., Thomas R., Sloan P.-P., Zwicker M.: Progressive photon beams. *ACM Transactions on Graphics (proc. Siggraph Asia) 30*, 6 (December 2011), 181:1–181:12. 2

[JZJ08] Jarosz W., Zwicker M., Jensen H. W.: The beam radiance estimate for volumetric photon mapping. *Computer Graphics Forum (Proc. Eurographics 2008) 27*, 2 (2008), 557 – 566. 2, 8

[NN94] Nishita T., Nakamae E.: Method of displaying optical effects within water using accumulation buffer. In *SIGGRAPH '94* (1994), ACM, pp. 373–379. 2

[PP09] Pegoraro V., Parker S. G.: An Analytical Solution to Single Scattering in Homogeneous Participating Media. *Computer Graphics Forum (Proc. Eurographics 2009) 28*, 2 (2009), 329–335. 2

[Ree11] Reeder M.: The kernel of a three-by-three matrix. `https://www2.bc.edu/~reederma/Linalg15.pdf`, 2011. 6

[SZLG10] Sun X., Zhou K., Lin S., Guo B.: Line space gathering for single scattering in large scenes. *ACM Transactions on Graphics (proc. Siggraph 2010) 29*, 4 (July 2010), 54:1–54:8. 2

[SZS*08] Sun X., Zhou K., Stollnitz E., Shi J., Guo B.: Interactive relighting of dynamic refractive objects. *ACM Transactions on Graphics (proc. Siggraph 2008) 27*, 3 (Aug. 2008), 35:1–35:9. 2

[WZHB09] Walter B., Zhao S., Holzschuch N., Bala K.: Single scattering in refractive media with triangle mesh boundaries. *ACM Transactions on Graphics (proc. Siggraph 2009) 28*, 3 (July 2009), 92:1–92:8. 1, 2, 3, 4, 8, 9, 10, 11

Appendix A: Complexity of the equation $\mathbf{f} = \mathbf{0}$

$\mathbf{f} = \mathbf{0}$ defines a polynomial equation of degree at most 6 in the parameters t, a and b.

Proof

Each vector $\mathbf{V}$, $\mathbf{P}$ and $\mathbf{N}_s$ (before normalization) can be expressed linearly with a parameter vector $\mathbf{x} = (t, a, b, 1)^\intercal$:

$$\mathbf{V} = \mathbf{V}_{\min} + t\frac{\mathbf{V}_{\max} - \mathbf{V}_{\min}}{\|\mathbf{V}_{\max} - \mathbf{V}_{\min}\|} = \mathbf{M}_V\mathbf{x} \tag{20}$$

$$\mathbf{P} = \mathbf{P}_0 + a(\mathbf{P}_1 - \mathbf{P}_0) + b(\mathbf{P}_2 - \mathbf{P}_0) = \mathbf{M}_P\mathbf{x} \tag{21}$$

$$\mathbf{N}_s = \mathbf{N}_0 + a(\mathbf{N}_1 - \mathbf{N}_0) + b(\mathbf{N}_2 - \mathbf{N}_0) = \mathbf{M}_N\mathbf{x} \quad (22)$$

Vectors $\mathbf{L} - \mathbf{P}$ and $\mathbf{V} - \mathbf{P}$ are also linear:

$$\mathbf{L} - \mathbf{P} = \mathbf{L} - \mathbf{M}_P\mathbf{x} = \mathbf{M}_{LP}\mathbf{x} \quad (23)$$

$$\mathbf{V} - \mathbf{P} = \mathbf{M}_V\mathbf{x} - \mathbf{M}_P\mathbf{x} = \mathbf{M}_{VP}\mathbf{x} \quad (24)$$

Solutions for $\mathbf{f} = \mathbf{0}$ must satisfy Snell's law:

$$\eta \sin\theta_i = \sin\theta_o \quad (25)$$

We express this law with our variables using cross products:

$$\eta \frac{||(\mathbf{V} - \mathbf{P}) \times \mathbf{N}_s||}{||\mathbf{V} - \mathbf{P}||||\mathbf{N}_s||} = \frac{||(\mathbf{L} - \mathbf{P}) \times \mathbf{N}_s||}{||\mathbf{L} - \mathbf{P}||||\mathbf{N}_s||} \quad (26)$$

All points solutions of the equation $\mathbf{f} = \mathbf{0}$ satisfy:

$$\eta^2(\mathbf{L} - \mathbf{P})^2||(\mathbf{V} - \mathbf{P}) \times \mathbf{N}_s||^2 = (\mathbf{V} - \mathbf{P})^2||(\mathbf{L} - \mathbf{P}) \times \mathbf{N}_s||^2 \quad (27)$$

$(\mathbf{L} - \mathbf{P})^2$ and $(\mathbf{V} - \mathbf{P})^2$ are polynomials of degree 2. Each coordinate of $(\mathbf{V}-\mathbf{P})\times\mathbf{N}_s$ and $(\mathbf{L}-\mathbf{P})\times\mathbf{N}_s$ is a polynomial of degree 2. The square of their norm is a polynomial of degree 4. Thus, Equation 27 is a polynomial equation of degree 6. $\square$