Master research Internship

Internship report

Fast Next-Event Estimation for Reflection and Refraction on Triangles with Interpolated Normals

Domain: Graphics - Light Transport Simulation

Author:
Marco Freire

Supervisors:
Nicolas Holzschuch
Maverick - Inria Rhône-Alpes

école normale supérieure

Abstract:

Physically-based rendering creates photorealistic images based on light transport theory. Path tracing renders an image by casting light paths through each pixel into the scene and simulating their interactions until they hit a light. It slowly converges as the number of paths traced per pixel increases, creating noisy images when this parameter is small. Next-event estimation accelerates convergence by connecting each interaction to a light source, producing better results with the same number of samples. It fails when there is no direct line of sight between a light and the interaction. Broadening the range of use cases of next-event estimation is essential for path tracing. Algorithms have been proposed to extend it to situations where a refractive interface breaks the line of sight, or where the light is only visible on a reflective surface. They rely on a 2D Newton's method to find these paths, which is expensive and sometimes struggles to converge towards a solution. The purpose of this internship is to accelerate next-event estimation in these situations. By using the coplanarity condition stated in the reflection and refraction laws, the domain of the search can be restricted to a conic section. We can then reformulate the problem over this one-dimensional space, and use a 1D Newton's method to find the refracted or reflected paths. Their contribution to the radiance is computed and taken into account by the path tracer, improving its convergence in difficult lighting situations. In this internship report, we define next-event estimation in the context of light transport theory, and we present two techniques working in the reflective and refractive situations. We then present the contribution of the internship divided in three parts: reducing the dimensionality of the problem, finding its solutions and computing their contribution. Finally, we end with a discussion of the implementation of the algorithm and a summary of the contributions of the internship.

Contents

1 Introduction

In computer graphics, *rendering* is the process of taking a description of a scene and producing an image of that scene as seen by an observer. It has many different applications in multiple domains. The movie and video game industries rely on rendering algorithms to create the images displayed on screen. Rendering is also used in architecture to visualise models of buildings before they are built in the real world. In general, it is heavily employed for simulation and visualisation purposes. Depending on the application, rendering can either be done in real-time or offline. In the former case, the main focus is efficiency, since images have to be created fast enough so that the observer cannot see the individual frames. In the latter case, time is not a limitation so a lot of attention can be put into creating a realistic image. *Physically-based rendering* aims to create an image of the scene based on the physics of *light transport theory*. These laws rule the interactions of light with the different components of the scene. The main goal is to create photorealistic images, indistinguishable from a real photography of the scene. The most important interaction of light with the scene is scattering. By understanding how light scatters when it encounters a surface, we can simulate its behaviour and create a physically accurate image.

Path tracing is the most used physically-based rendering algorithm. It shoots light rays from the eye of the observer through every pixel of the image into the scene. These rays form a light path that bounces around in the scene, and whose interactions are simulated according to light transport theory. Finally, when the path meets a light source, the illumination provided by that light is propagated back to the initial pixel to compute its final value. How light paths bounce in the scene is determined stochastically. For this reason, there is no guarantee that a light path cast from the eye will ever reach a light source and contribute to the value of the pixel. This leads to noisy outputs in complex scenes rendered with regular path tracing. To avoid this, *next-event estimation* (NEE) techniques cast a ray from every interaction point towards a light source. NEE significantly improves the quality of the resulting image, since every path now contributes to the pixel's color.

Unfortunately, NEE only works if there is a direct line of sight between the interaction point and a light source. This line of sight is often obstructed by objects in the scene. Researchers have developed algorithms [36][16] to apply NEE through a refractive interface or after a bounce on a reflective surface. This extends the range of situations NEE can be used in. These algorithms are far from trivial, since straight paths between a point and a light traversing a refractive interface do not obey Snell's law of refraction. These methods find valid paths by searching the two-dimensional interface for the roots of an equation encoding the refraction law. A *2D Newton-Raphson method* is used for finding the roots. This is a non-linear problem which can struggle to converge to a solution in difficult situations and requires expensive computation to solve. The goal of this internship is to make NEE computation faster by searching for valid paths over a one-dimensional domain instead.

The reflection and refraction laws determine the relationship between the angle of incidence and reflection or refraction, but they also state that the corresponding rays and the normal to the surface are coplanar. Previous algorithms encoded these two statements in a 2D constraint function and looked for its roots. These conditions can be separated and evaluated in two stages. First, the coplanarity condition can be written as a quadratic form defining a conic section over each triangle of the interface. For a given triangle, all roots of the constraint function necessarily lie on that conic. Then, by reformulating the constraint function over the conic section, we can use a 1D Newton's method to find the solutions to the constraint function. Finally, we can compute the contribution of these light paths to the illumination of the scene.

In this internship report, we begin by giving the theoretical framework of light transport simu-

lation needed to understand how path tracing works and the reasons behind next-event estimation, on which the internship is based. Next, we explain why NEE is not applicable as soon as there is no line of sight between the interaction point and a light source. We describe two algorithms extending the use of NEE to situations where the two points are separated by refractive or reflective interfaces. The next sections describe the three main phases of the algorithm: reducing the dimensionality of the problem, finding its solutions and computing their contribution. Finally, the implementation of the algorithm is discussed and the contribution is summarised.

2 Prerequisites

Modern physically-based rendering algorithms such as *path tracing* are built upon the *ray tracing* algorithm. While light naturally propagates from the light sources to the eye of the observer, this algorithm shoots rays into the scene and computes their interactions with its components. For each pixel, a ray going from the eye of the observer and passing through the center of the pixel is cast into the scene. Then, the nearest intersection of the ray with the scene is computed. Finally, the illumination at that point is calculated by shooting a ray towards the light sources, to see whether the point is lit by them or not. By repeating this process for each pixel, an image of the scene is created. The ray tracing algorithm is illustrated in figure 1.

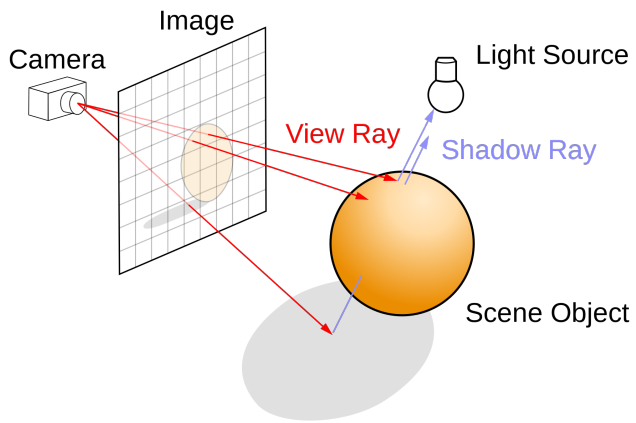


Figure 1: Ray tracing illustration taken from Wikipedia

The core idea of ray tracing was first introduced by Appel [4] in 1968. He devised an algorithm to determine which surfaces in a scene are visible from the point of view of an observer, and compute shadows on those surfaces by casting light rays. Then Whitted [37] improved the method by presenting a recursive ray tracing algorithm in 1980, often referred to as Whitted-style ray tracing. This algorithm simulates perfect reflection and refraction by recursively shooting new rays at each intersection point according to Snell's laws. Cook *et al.* [9] extended this method in 1984 to accurately simulate more advanced features such as glossy reflection and refraction, area light sources, depth of field and motion blur. Finally, Kajiya [24] formulated the rendering problem within a rigorous theoretical framework in 1986, where he introduced the modern path tracing algorithm. He reformulates the rendering problem as the resolution of an integral equation derived from energy conservation laws in the scene. The path tracing algorithm solves this equation by *Monte Carlo integration*.

2.1 Light transport theory

In this section we will introduce the concepts required to understand the path tracing algorithm and the importance of *next-event estimation*, which will be the main focus of this internship. We also introduce basic notions about light interactions with participating media, which will be useful later when computing the contribution of light paths in section 7. In the rest of this section, we will use the theoretical framework found in the textbooks [12] and [28].

2.1.1 Radiometric quantities

To begin to understand physically-based rendering, it is necessary to introduce the physical quantities that measure and describe light propagation in an environment. The field of radiometry studies the physical measurement of light and uses radiometric quantities for this purpose. For the sake of simplicity, we will only introduce the most essential radiometric quantities: the *radiance* and the *bidirectional reflection distribution function* (BRDF). To study light transport, we place ourselves in the model of geometrical optics and we compute the steady-state distribution of radiance in the scene. Radiometric quantities depend on wavelength, but we will not mention this dependence explicitly here.

Radiance Radiance is a five-dimensional quantity that measures how much radiant power Φ arrives or leaves at a surface point per unit solid angle $d\omega$ and per unit projected area $dA^{\perp}$. It is defined by:

$$L = \frac{\partial^2 \Phi}{\partial \omega \partial A^{\perp}} = \frac{\partial^2 \Phi}{\partial \omega \partial A \cos\theta} \quad [W \cdot sr^{-1} \cdot m^{-2}]$$

where radiant power Φ measures how much energy flows through a surface per unit time. We use $L(x \leftarrow \Theta)$ (resp. $L(x \rightarrow \Theta)$) to denote the radiance arriving to (resp. leaving from) the point x along the direction Θ.

Radiance is the fundamental quantity in light transport simulation. All other radiometric quantities can be derived from it by integration. The response of sensors such as a camera or an eye is proportional to the radiance incident upon them. This makes radiance the quantity to compute to obtain the render of a scene. Also, radiance is invariant along straight paths in a vacuum. This means that given a straight path connecting two points, the radiance leaving the first point towards the second one is equal to the radiance arriving to the second point from the first one. This does not hold when light travels through a medium. Light will interact with it, introducing new physical phenomena that invalidate this property. We will assume to work in a vacuum unless stated otherwise.

BRDF When light hits a surface at a point p with incident direction Ψ, it leaves the surface at a point q with outgoing direction Θ. We assume here that p and q are the same point, ignoring subsurface scattering.

The BRDF encodes the appearance of the materials in the scene by describing how incident energy is reflected by a surface depending on the incident and reflected directions. It is defined by:

$$f_r(x, \Psi \rightarrow \Theta) = \frac{\mathrm{d}L(x \rightarrow \Theta)}{\mathrm{d}E(x \leftarrow \Psi)} = \frac{\mathrm{d}L(x \rightarrow \Theta)}{L(x \leftarrow \Psi)\cos(N_x,\, \Psi)\,\mathrm{d}\omega_{\Psi}}$$

where $E = \frac{\mathrm{d}\Phi}{\mathrm{d}A}$ is the irradiance, i.e. the incident radiant power on a surface per unit surface area, and N_x is the normal to the surface at x.

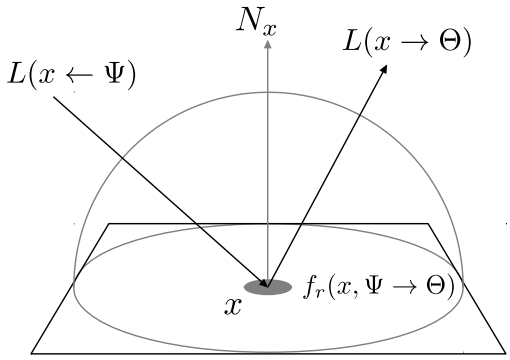


Figure 2: Illustration of radiometric quantities

2.1.2 The rendering equation

Now we have the necessary tools to understand the theoretical formulation of light transport simulation and the rendering equation. This equation was first introduced by Kajiya in [24]. The equation states that the radiance at a point x along a direction Θ is equal to the radiance L_e emitted at x towards Θ plus the radiance L_r reflected at x towards Θ. Intuitively, the reflected radiance can be seen as the integral of the radiance arriving at x from every direction, weighted by the BRDF between the incident direction and Θ.

The rendering equation comes in two main formulations, the hemispherical and area formulations. They only differ on the expression of the reflected radiance. In the former, reflected radiance is expressed as an integral over the unit hemisphere Ω_x at the surface point x, while in the latter it is an integral over the set A of surfaces in the scene.

The hemispherical formulation of the rendering equation can be written as follows:

$$
\begin{aligned}
L(x \rightarrow \Theta) &= L_e(x \rightarrow \Theta) + L_r(x \rightarrow \Theta); \\
L(x \rightarrow \Theta) &= L_e(x \rightarrow \Theta) + \int_{\Omega_x} f_r(x, \Psi \rightarrow \Theta) L(x \leftarrow \Psi) \cos(N_x, \Psi) \, \mathrm{d}\omega_\Psi; \\
L(x \rightarrow \Theta) &= L_e(x \rightarrow \Theta) + \int_{\Omega_x} f_r(x, \Psi \rightarrow \Theta) L(r(x, \Psi) \rightarrow -\Psi) \cos(N_x, \Psi) \, \mathrm{d}\omega_\Psi.
\end{aligned}
$$

The *ray-casting* function $r(x, \Psi)$ returns the point on the closest visible object along a ray starting at x in the Ψ direction. The invariance of radiance along straight paths gives us $L(x \leftarrow \Psi) = L(r(x, \Psi) \rightarrow -\Psi)$. This gives us a recursive equation of the exitant radiance $L(\cdot \rightarrow \cdot)$. The BRDF and the emitted radiance are specified in the scene description. We consider that emitted radiance is exclusively non-zero for light sources.

The area formulation is the following:

$$
L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + \int_A f_r(x, \Psi \rightarrow \Theta) L(r(x, \Psi) \rightarrow -\Psi) V(x, y) G(x, y) \, \mathrm{d}A_y
$$

where the visibility term $V(x, y)$ is non-zero if x and y are mutually visible, and the geometry term $G(x, y)$ depends on the geometry of the scene.

2.1.3 Participating media

The previous equations depend on the invariance of radiance along straight paths. This property derives from the assumption that the light in the scene travels in a vacuum. Participating media interact with light passing through them in different ways. If the scene contains a participating medium, the previous equations are no longer valid. This section gives a quick introduction to the properties of participating media in light transport and explains how radiance decreases along straight paths due to attenuation phenomena.

Interactions with participating media come in two types: given a light ray traversing the medium, some interactions contribute to the radiance it carries, and some reduce it. Radiance can be lost either through *absorption* or *out-scattering*. Absorption occurs when the radiance carried by the ray is absorbed by the particles composing the medium and transformed into other forms of energy, such as heat. Out-scattering happens when the radiance carried by the ray is scattered in different directions by the particles. On the other hand, *emission* and *in-scattering* increase the radiance of the ray. The medium can contain emissive particles that increase the radiance of a ray passing through it. Also, the radiance lost to out-scattering can contribute to other rays passing through the medium, thus increasing their radiance by in-scattering. These four interactions are illustrated in figure 3. We only need to consider attenuation phenomena for the purpose of this report. For this reason, we will ignore emission and in-scattering in the rest of this section.

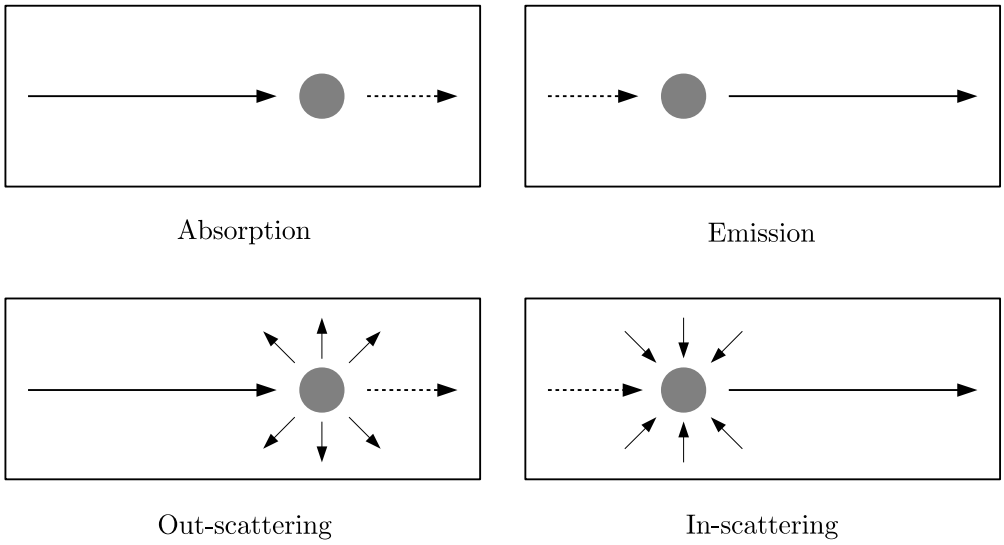


Figure 3: Volume scattering processes

The frequency and the impact of these phenomena are described by a few physical quantities. Absorption is described by the medium's *absorption cross section* $\sigma_a(p, \omega)$ $[m^{-1}]$, representing the probability that light is absorbed per unit distance traveled in the medium. Out-scattering is described by the *scattering coefficient* $\sigma_s(p, \omega)$ $[m^{-1}]$, representing the probability of an out-scattering event happening per unit distance. Usually, these two attenuation phenomena are described by the *attenuation factor* $\sigma_t(p, \omega) = \sigma_a + \sigma_s$, or the *mean free path* $1/\sigma_t$ which represents the average distance that a ray travels in the medium without interacting with it. These coefficients depend on the position inside the medium and the direction of propagation of the light.

The attenuation of radiance along a ray at a medium point p in a direction ω along a differential

length $\mathrm{d}t$ is given by the following differential equation:

$$\frac{\mathrm{d}L(p \to \omega)}{\mathrm{d}t} = -\sigma_t \cdot L(p \leftarrow -\omega).$$

The solution to this equation involves the *beam transmittance* $T_r(p \to p')$ which represents the fraction of radiance transmitted between two medium points p and p':

$$T_r(p \to p') = \exp(-\int_0^d \sigma_t(p + t\hat{\omega}, \omega)\,\mathrm{d}t)$$

where $\hat{\omega}$ is the normalised direction from p to p', and d the distance between them. This factor describes how radiance is attenuated along straight path inside of a participating medium:

$$L(p' \leftarrow -\omega) = T_r(p \to p') \cdot L(p \to \omega).$$

Finally, volume scattering inside a participating medium is described by the phase function $\rho(p, \Theta \to \Psi)$, similar to the BRDF $f_r(x, \Theta \leftrightarrow \Psi)$ for surface scattering.

2.2 Monte Carlo path tracing

In this section we will explain in detail how the path tracing algorithm works. The radiance arriving from the scene to the eye of the observer through a specific pixel will determine the value of that pixel. Then, by solving the rendering equation to calculate the value of that radiance, we can obtain the colour of every pixel in the final image. To solve the rendering equation, we need a way to compute integrals. This section briefly states the principles behind Monte Carlo methods to then explain how path tracing works and finally describe what next-event estimation is.

2.2.1 Monte Carlo integration

Let us take a function f over a domain Ω and I the integral to compute. Monte Carlo integration approaches the integral with an estimator $\langle I \rangle$ computed from samples (x_i) taken over the integration domain, each with probability $p_\Omega(x_i)$.

$$I = \int_\Omega f(x)dx \qquad \langle I \rangle = \frac{1}{N}\sum_{i=1}^{N} \frac{f(x_n)}{p_\Omega(x_n)}$$

$$\mathbb{E}[\langle I \rangle] = I \qquad \mathbb{V}[\langle I \rangle] = \frac{1}{N}\int_\Omega \left[\frac{f(x)}{p(x)} - I\right]^2 p_\Omega(x)\,\mathrm{d}x$$

As the number of samples N increases, $\langle I \rangle$ converges towards I since the integral to compute is its expected value. Moreover, the error in the estimator is proportional to its standard deviation and decreases as $\sqrt{N}$. Monte Carlo integration works independently of the dimension of the integration domain, unlike quadrature methods whose complexity scales with dimension.

To compute the lighting integrals, we just need a sampling strategy over the integration domains, which will be either hemispheres or surfaces. In practice, intelligent sampling of the domain accelerates the convergence of the estimator.

2.2.2 Simple Monte Carlo path tracing

In this section we present the simplest version of Monte Carlo path tracing by putting together the different concepts and tools introduced in the previous sections.

Algorithm This path tracing algorithm uses the hemispherical formulation of the rendering equation. Path tracing constructs light paths recursively as shown in figure 4. First, the algorithm casts a ray from the eye of the observer through a pixel. Then it computes the intersection of the ray with the scene. Finally, a direction is sampled over the hemisphere and a new ray is cast from the intersection point in that direction. At each bounce of the light path, the emitted radiance is added to the contribution of the path, and the radiance carried by the rest of the path is weighted by the BRDF. Path tracing stops after a certain number of bounces or when the contribution of future bounces drops below a fixed threshold.

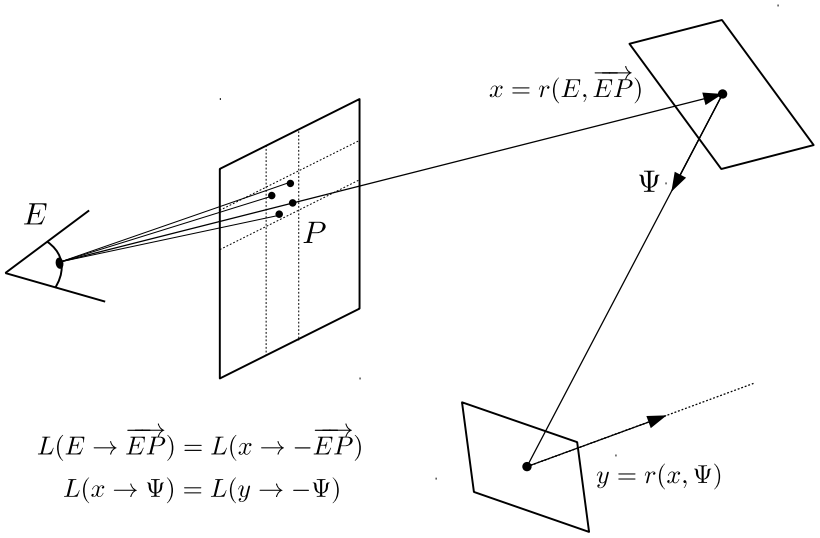


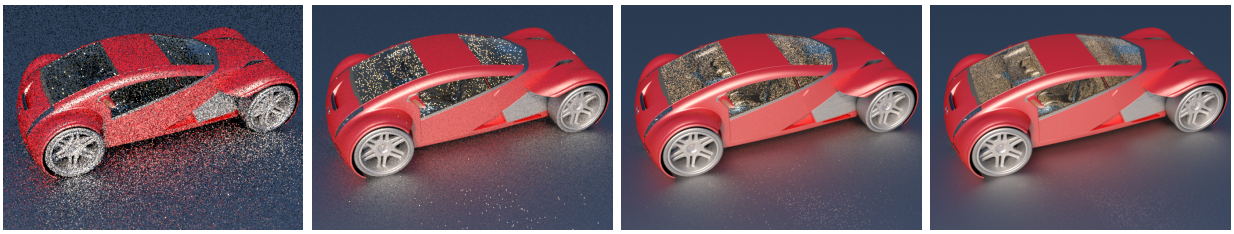
Figure 4: Path tracing algorithm

The number of *samples per pixel* (spp) is a parameter of the algorithm representing the number of rays cast through each pixel. A higher number of samples creates a clear image at a larger computational cost, while a lower number returns noisy images with a shorter execution time as illustrated in figure 5.

Drawbacks For the radiance of a path to be non-zero, it needs to hit a light source. Since lights occupy only a small fraction of the surfaces in the scene, most rays will not contribute to the final image. This algorithm produces very noisy images for this reason. Increasing the number of samples very slowly improves the render. Therefore, this version of the algorithm is inefficient and rarely used in practice.

2.2.3 Next-event estimation

Principle A more efficient way to compute the radiance separates the contribution of direct illumination and indirect illumination. Direct illumination is the light that arrives at a surface



(a) 1 spp (0.7 s) (b) 32 spp (21 s) (c) 256 spp (3 min) (d) 1024 spp (12.4 min)

Figure 5: Comparison of a car rendered with path tracing at different spp, taken from N. Holzschuch's website.

point directly from the light sources in the scene. Indirect illumination is the light that bounces at least once in the scene before arriving to the surface point. The rendering equation states that the exitant radiance at x along Θ is the sum of the emitted radiance at x along Θ and the radiance reflected at x along Θ. We can split the reflected term into direct and indirect contribution in the following way:

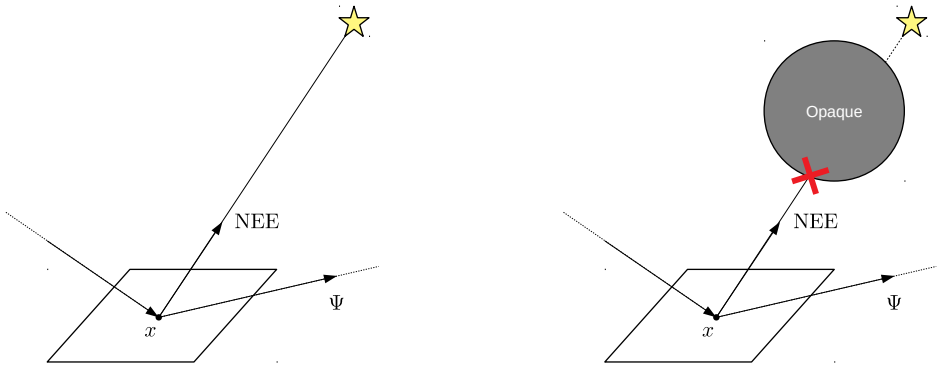
$$
\begin{aligned}
L(x \to \Theta) &= L_e(x \to \Theta) + L_r(x \to \Theta) \\
L_r(x \to \Theta) &= L_{direct}(x \to \Theta) + L_{indirect}(x \to \Theta) \\
L_{direct}(x \to \Theta) &= \int_{A_{LS}} f_r(x, \overrightarrow{xy} \to \Theta) L_e(y \to \overrightarrow{yx}) V(x,y) G(x,y) \, \mathrm{d}A_y \\
L_{indirect}(x \to \Theta) &= \int_{\Omega_x} f_r(x, \Psi \to \Theta) L_r(r(x,\Psi) \to -\Psi) \cos(N_x, \Psi) \, \mathrm{d}\omega_\Psi
\end{aligned}
$$

where A_{LS} is the combined area of the light sources in the scene and y is a point on a light source.

Algorithm The direct component of reflected radiance is computed by explicitly connecting each intersection point of the light path with the scene to the light sources. At each bounce, a sample is taken on the surface of the light source and a ray is cast from the interaction point towards the sample. Finding the connection between these points is called *next-event estimation* (NEE). Indirect illumination is then computed as in regular path tracing, by sampling a direction over the hemisphere and casting a ray along that direction. With this technique, almost every path will have a non-zero contribution, since each intersection point is deterministically connected to the light sources. For this reason, path tracing with NEE produces much clearer renders with the same number of samples at an increased computational cost.

Issues NEE breaks down when there is no direct line of sight between the surface point and the light sample. This happens when an object stands between these two points. In that case, the light sample does not contribute directly to the illumination of the surface point, since no connection between them can be found. This situation is illustrated in figure 6.

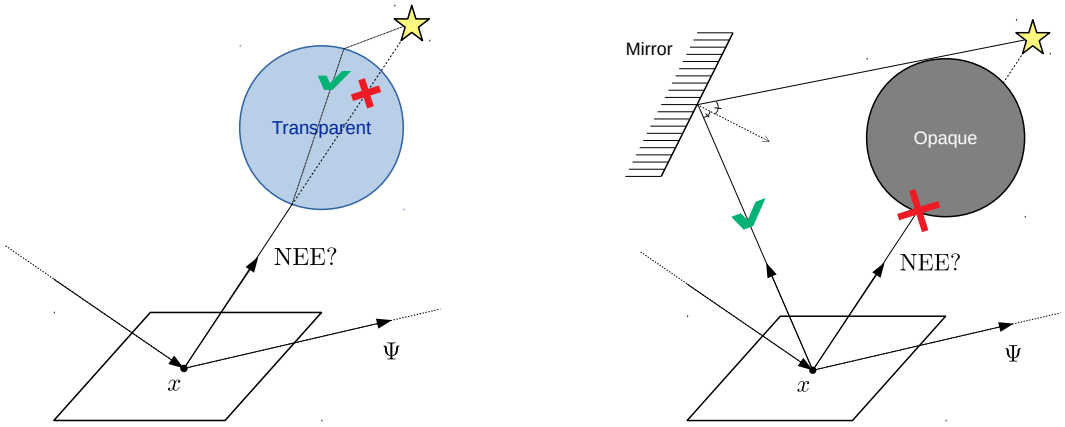
Even without direct line of sight, there are specific situations where NEE is still possible, as seen in figure 7. In the first situation, paths between the point and the light are blocked by a transparent object. In the second situation, the light is only visible from the point after reflection



(a) Unobstructed line of sight (b) Obstructed line of sight

Figure 6: Regular NEE breaks down with no direct line of sight

on a mirror. Since refraction and reflection obey Snell's laws, it is possible to trace paths towards the light through these obstacles.



(a) With a transparent object (b) With a mirror

Figure 7: NEE extended to refractive interfaces and reflective surfaces

Unfortunately, finding these paths is no easy task. For example, there is no guarantee that a ray cast from the surface point towards a transparent interface will be refracted exactly in the direction of the light. In the following section, we explain how advanced NEE algorithms manage to find these complicated paths.

3 State of the art

This section presents two different algorithms extending next-event estimation to more complex situations involving refractive interfaces and reflective surfaces. These two algorithms are global NEE and manifold NEE, respectively published by Walter *et al.* [36] in 2009 and Hanika *et al.* [16] in 2015.

3.1 Global next-event estimation (GNEE)

3.1.1 Single scattering through a refractive interface

Walter *et al.* [36] propose an algorithm for NEE through a single refractive interface. We consider a translucent object filled with a homogeneous participating medium of constant refractive index η, whose boundary is a refractive interface represented by a triangle mesh with interpolated vertex normals. Given a path passing through the translucent object and a sample $\mathbf{V}$ on the path inside the object, we want to find all rays connecting the sample to a light source $\mathbf{L}$ through the interface as shown in figure 8. This means finding every point $\mathbf{P}$ on the interface such that $\mathbf{LPV}$ is a valid refracted path from the light to the sample. Then we can compute the contribution of such paths to the illumination at $\mathbf{V}$.

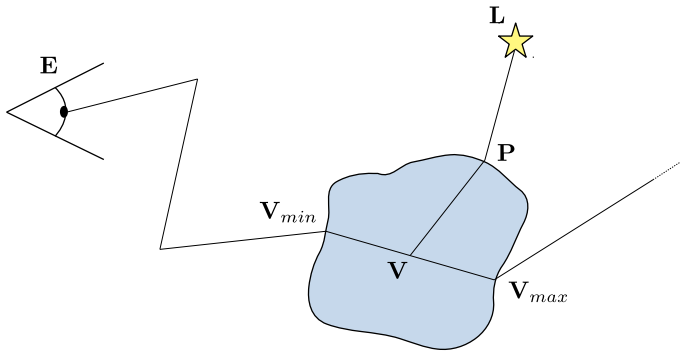


Figure 8: Single scattering through a refractive interface

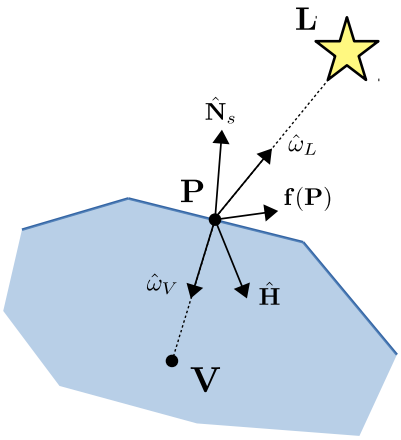
3.1.2 Half-vector formulation

Refraction through an interface between media of different indices of refraction obeys Snell's law. This law can be reformulated as a constraint on the *refractive half-vector* $\hat{\mathbf{H}}$ introduced by Walter *et al.* [35]. Since the mesh triangles have shading normals $\hat{\mathbf{N}}_s$ different from their geometric normals $\hat{\mathbf{N}}_g$, Snell's law must be applied to the shading normals. The definition of the half-vector and the *refraction constraint function* $\mathbf{f}$ are given in figure 9.

The formulation follows the convention that the half-vector and the shading normal have opposite directions, with the half-vector pointing towards the medium. Finding a point $\mathbf{P}$ on the boundary such that the path $\mathbf{LPV}$ satisfies Snell's law is then equivalent to solving for the roots of the constraint function $\mathbf{f}(\mathbf{P}) = 0$. While the article only gives examples for refractive interfaces, the exact same approach works for specular interfaces. The half-vector formulation just needs to be modified to take into account reflection instead of refraction.

3.1.3 Algorithm

To compute the contribution of this kind of path, the algorithm places several sample points $\mathbf{V}$ along the ray inside the medium. For each sample $\mathbf{V}$, it finds all refracted paths connecting $\mathbf{V}$ to the light source $\mathbf{L}$. Then, it computes and sums their contributions to the radiance along the path. These paths are found by iterating over every triangle in the mesh. Solving for the roots of $\mathbf{f}$ over



$$\hat{\omega}_V = \frac{\mathbf{V}-\mathbf{P}}{\|\mathbf{V}-\mathbf{P}\|}$$
$$\hat{\omega}_L = \frac{\mathbf{L}-\mathbf{P}}{\|\mathbf{L}-\mathbf{P}\|}$$
$$\hat{\mathbf{H}} = \frac{\eta\hat{\omega}_V + \hat{\omega}_L}{\|\eta\hat{\omega}_V + \hat{\omega}_L\|}$$
$$\mathbf{f}(\mathbf{P}) = \hat{\mathbf{H}} + \hat{\mathbf{N}}_s.$$

Figure 9: Refractive half-vector formulation

a triangle gives us all valid paths passing through that triangle. The roots are computed with a 2D Newton-Raphson method over its surface.

Instead of searching over the whole surface of the triangle, the authors use a split-and-prune approach. They recursively split the triangle into four smaller subtriangles and prune those that cannot contain a solution, i.e. where the bounding cones for $-\hat{\mathbf{H}}$ and $\hat{\mathbf{N}}_s$ do not overlap. Then, the Newton-Raphson method is applied to the subtriangles. The subdivision stops according to a fast heuristic which stops splitting triangles when the sum of the cone angles for the two vectors is smaller than thirty degrees. This heuristic is not exact, but works well in practice. The authors also provide a rigorous stopping condition which is much more expensive to compute.

For small meshes, the algorithm can be applied to every triangle but this solution does not scale well with the size of the mesh. For large meshes, whole groups of triangles that cannot contain a solution are pruned according to three tests. These tests rely on an additional data structure to obtain information about the position and normals of the triangles in the scene. The sidedness test checks whether $\mathbf{L}$ and $\mathbf{V}$ are on the correct side of the shading normal $\hat{\mathbf{N}}_s$. The spindle test is based on the fact that refraction cannot bend light by more than a straight angle, and often less for small values of η. If we think of the segment $\mathbf{LV}$ as the chord of a circle, the surface of revolution produced by the arc subtended by that chord contains all triangles where a solution can exist. The cone overlap test is similar to the one used in the split-and-prune phase, but position is bounded by a 3D box instead of a 2D triangle. If any of these tests fail for a given triangle, then it can be discarded since no refracted path can go through it.

This algorithm works well for simple scenes, but the authors report that in complex scenes the number of samples along each camera ray has to be very high to obtain a clear image. Also, some samples had extremely high values and had to be clamped to prevent noise in the final image.

3.1.4 Results

In figure 10 we see a comparison of a scene rendered with four different algorithms. Regular path tracing renders a very noisy but physically-accurate result after 1.4 hours of rendering and 32768 samples per pixel. NEE by approximating refracted paths by straight lines does not manage to capture the intricate details of the scene. The images created with photon mapping and GNEE in this figure took roughly the same time to compute. Compared to photon mapping, GNEE captures

finer details on the inside of the sphere and uses far less memory.

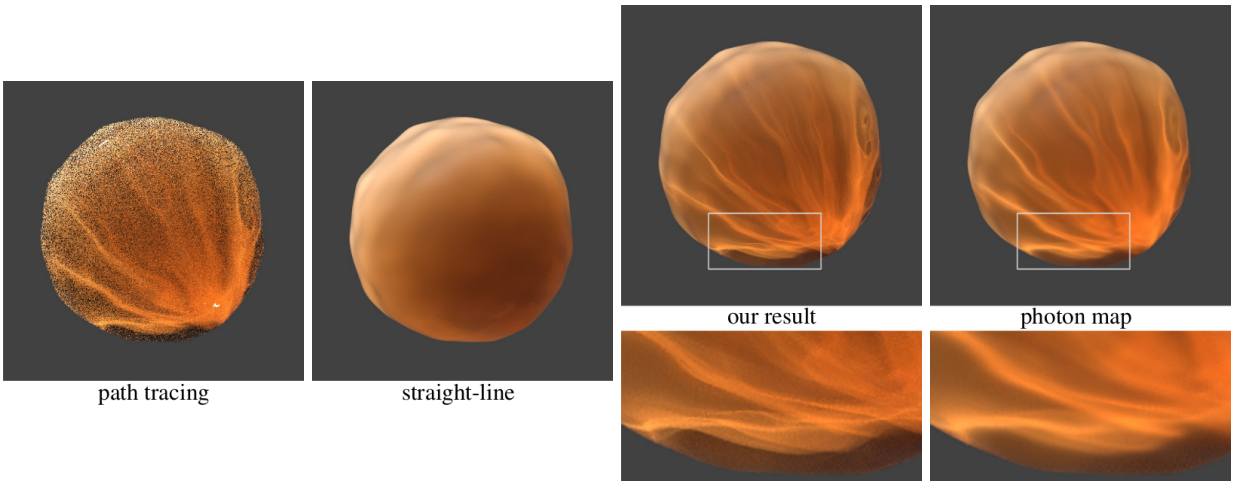


Figure 10: Equal time comparison (one minute) renders of a back lit bumpy sphere created with different algorithms, taken from [36].

3.2 Manifold next-event estimation (MNEE)

3.2.1 Rendering of specular-diffuse-specular paths

Hanika *et al.* [16] propose an algorithm for a related but slightly different problem. They aim to efficiently render specular-diffuse-specular (SDS) paths. These paths contain a diffuse vertex between two specular scattering events (such as reflection or refraction) as shown in figure 11. In order to efficiently sample these paths, the authors propose another next-event estimation algorithm. While the previous algorithm only handles refraction through a single perfect refractive interface, this algorithm works on multiple layers of potentially rough interfaces.

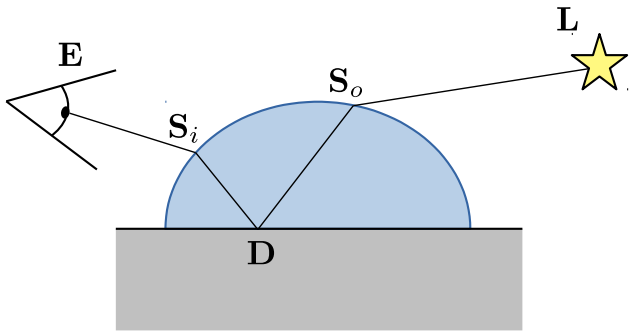


Figure 11: SDS path with specular scattering events at $\mathbf{S}_i$, $\mathbf{S}_o$ and a diffuse scattering event at $\mathbf{D}$

3.2.2 Manifold exploration

Manifold next-event estimation is based on the *manifold exploration* (ME) technique introduced by Jakob *et al.* [23]. If we consider a path of length k as a sequence $(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ of k vertices in the scene, we can denote the space of paths of length k by Ω_k. Path space $\Omega = \bigcup_k \Omega_k$ is then the set of paths of every possible length. Solving the rendering equation by path tracing is reformulated by Veach [33] as solving a rendering equation where the integration domain is path space. Instead of taking samples over a hemisphere or a surface, the samples are light paths themselves. The contribution of each one of those sampled paths is computed and used to create the final image. Given an interesting path in the scene, ME is used to explore other paths close to it in path space.

In a path $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$, refraction at a vertex $\mathbf{x}_i$ can be encoded as a constraint on the half-vector $\hat{\mathbf{H}}$ at that vertex depending only on the surrounding vertices $\mathbf{x}_{i-1}$ and $\mathbf{x}_{i+1}$. This constraint is represented in the previous section by the function $\mathbf{f}$. Similarly, we can define a constraint function C encoding these specular constraints for the whole path such that the path is valid only if it belongs to $\{\bar{\mathbf{x}} = (\mathbf{x}_1, \ldots, \mathbf{x}_n) \in \Omega \text{ s.t. } C(\bar{\mathbf{x}}) = 0\}$. Therefore the constraint function C implicitly defines a manifold called a *specular manifold*. This is a lower-dimensional space of valid paths embedded in path space.

Given a path on this specular manifold, ME allows us to locally explore the manifold around that path. If an endpoint of this path is modified, a predictor/corrector algorithm first propagates the perturbation to the rest of the path by moving in the tangent space to the manifold (*prediction*). This produces a similar path that is no longer on the manifold. This path is then projected back onto the manifold by ray tracing from the other endpoint so that it satisfies the specular constraints and all vertices land on the geometry (*correction*). Figure 12 illustrates this algorithm in scene space and path space.

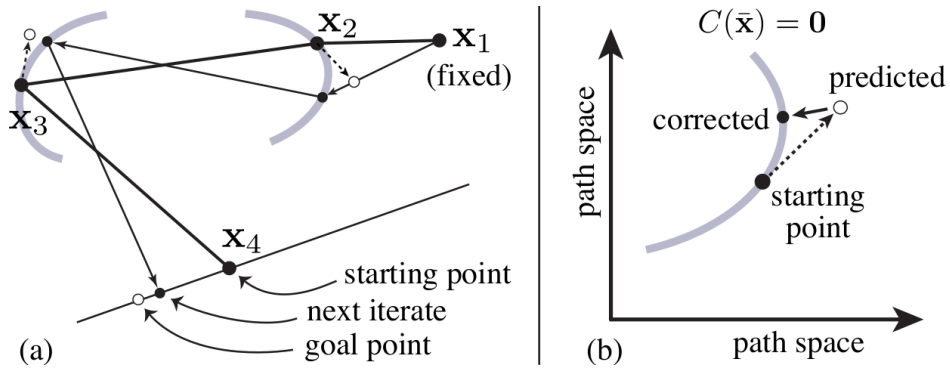


Figure 12: Image taken from [23], **(a)** shows an iteration of the algorithm in scene space where an endpoint of the path is altered, **(b)** shows the same iteration in path space

In other words, it is possible to alter a valid path and obtain through successive iterations a similar valid path taking into account that alteration. This method behaves like the Newton-Raphson method and exhibits quadratic convergence when near the solution.

3.2.3 Algorithm

MNEE uses a similar approach to manifold exploration but represents paths with half-vectors instead of vertices. First, a straight seed ray is shot from the diffuse vertex towards the light source, ignoring

specular events. Then, a set of target half-vectors verifying the specular constraints are chosen. Finally the predictor/corrector algorithm creates a valid path from the seed path, getting closer to the target constraints with each iteration. This is shown in figure 13. MNEE can be applied to reflective surfaces by using reflection constraints instead of refraction in the function C for some vertices in the path.

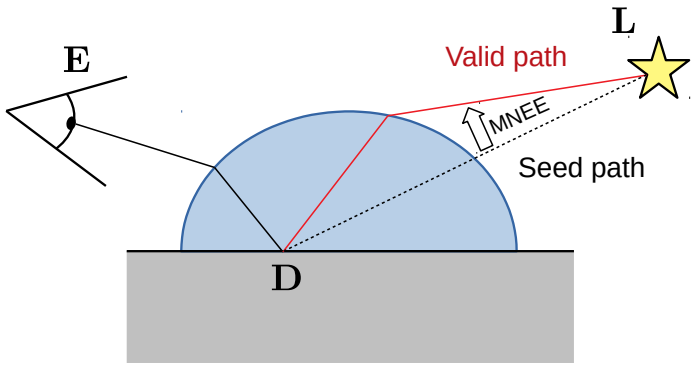


Figure 13: MNEE algorithm

The constraint function C for a path of length n with p specular vertices is represented as a block tridiagonal matrix of size $2n$ by $2p$. The algorithm needs to calculate this matrix and its inverse to find a valid path, which is computationally expensive. Also, the convergence of the algorithm is quadratic only as long as the seed path is close to the solution. If the seed path is too far off, the algorithm may not converge in a reasonable number of iterations. The authors suggest that a maximum number of 15 iterations works well in practice. In a difficult scene presented in the article, a maximum of 50 iterations leads to a rate of successful convergence of 59%. In general, MNEE is very sensitive to the handling of geometry by the renderer and numerical stability, but produces very good results in the right conditions.

3.2.4 Results

In figure 10 we see an equal time comparison of a scene rendered with four different algorithms in one minute. As expected, path tracing renders a very noisy image at a very high cost. Regular NEE accurately renders the plane but struggles to render the droplet to which this technique does not apply. MNEE creates a very clear render in the same amount of time, except for some noise that can be seen in the red and blue inlets.

3.3 Discussion

In this section we have presented two different next-event estimation methods: GNEE and MNEE. GNEE finds all paths refracted through an interface contributing to the radiance carried along a ray inside a participating medium. The paths are found by encoding the refraction law with a constraint function depending on a point on the boundary. Then a 2D Newton-Raphson method is used to find the roots of this function over the whole boundary, excluding triangles where no solutions can exist thanks to pruning tests. This method only handles perfect refractive surfaces and single scattering, but it returns better results than regular path tracing in a reasonable time.

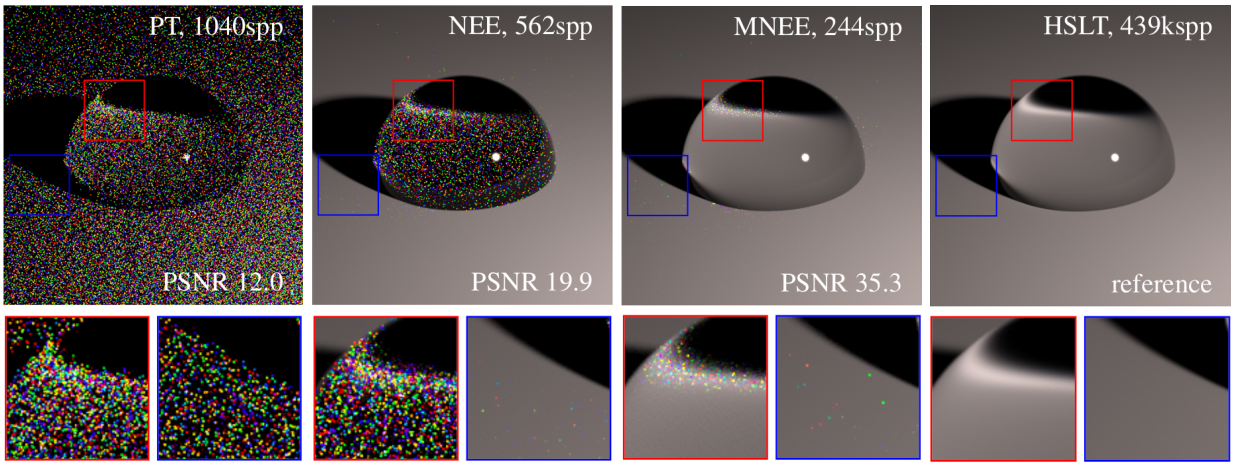


Figure 14: Equal time comparison (one minute) renders of a water droplet on a flat plane created with different algorithms, taken from [16].

MNEE solves single scattering through an interface to efficiently render specular-diffuse-specular paths, which are notoriously difficult to sample. Their method finds a refracted path between the diffuse vertex and the light sample starting from a seed path. This seed path is then refined by a predictor/corrector algorithm based on manifold exploration that converges towards a valid refracted path. This algorithm behaves as a 2D Newton-Raphson method in terms of complexity. MNEE is designed to handle rough scattering through multiple layers, but works better in practice on single scattering through a single low-roughness interface. The convergence of the algorithm is very sensitive to the geometry of the scene, and it requires heavy computation on matrices.

Both of these algorithms can also be extended to deal with reflection instead of refraction, but the articles mostly provide examples of the refractive case. In summary, these algorithms showcase a range of different methods each used in different contexts, which proves the versatility and usefulness of NEE in many situations arising naturally in path tracing.

In the previous sections we have explained the importance of next-event estimation in path tracing algorithms by introducing the theoretical framework of light transport theory on which they rely. Next-event estimation accelerates convergence in a wide variety of situations, but fails when there is no direct line of sight to the light sources. We have presented two algorithms that extend NEE to work through refractive interfaces and with reflective surfaces. These algorithms successfully render complex situations where regular path tracing struggles to create a clear image, but NEE increases the computational cost of the algorithm for the same number of samples. The goal of this internship is to develop a faster next-event estimation technique for these situations.

4 Overview

4.1 Motivation

To understand the idea behind the new NEE technique, let us restate the problem first and summarise the previous approaches. Given a point **V** inside a medium and a point **L** on a light source separated by a refractive interface, we want to find a point **P** on the interface such that **LPV** is a valid refracted path. Both previously presented methods rely on finding the roots of a function

defined over the two-dimensional interface. The search for the roots is slow and does not always converge, as shown by Hanika *et al.* [16] with convergence rates as low as 59% in difficult examples. Also, the constraint is non-linear in its parameters due to normalisation, making it much harder to optimise. This internship aims to reduce the dimension of the search space by taking advantage of the coplanarity of the incident and outgoing rays with the shading normal at the interaction point.

Reflection and refraction obey the law of reflection and Snell's law respectively, each consisting of two parts. The first part states that the incident ray, the outgoing ray and the normal to the surface are coplanar. The second part then states a relationship between the angles of incidence and reflection or refraction. In both GNEE and MNEE, these conditions are encoded together in the constraint function whose roots we want to find. We aim to improve the performance of NEE by using both conditions separately.

Coplanarity between $\overrightarrow{\mathbf{VP}}$, $\overrightarrow{\mathbf{PL}}$ and the shading normal at $\mathbf{P}$ is a necessary condition for $\mathbf{f}(\mathbf{P})$ to be zero. Since these three vectors are linear in $\mathbf{P}$ before normalisation, the coplanarity condition can be written as a quadratic form in $\mathbf{P}$. This quadratic form defines a conic section over the plane containing each mesh triangle. The boundary point of any valid path will be on this curve. This reduces the dimension of the space search to one dimension. This condition can also be used for pruning triangles where a solution cannot exist. Indeed, if the conic section does not intersect a triangle, then it cannot contain a solution and it can be safely ignored.

By using coplanarity to reduce the dimensionality of the search space and prune non-relevant triangles, we hope to accelerate NEE computation and improve over the results of the previous methods.

4.2 Algorithm overview

Let us consider a scene containing a participating medium with index of refraction η enclosed in a refractive interface represented as a triangle mesh with interpolated vertex shading normals. Given a sample $\mathbf{V}$ in the medium and a sample $\mathbf{L}$ on a light source across the refractive interface, we want to find every point $\mathbf{P}$ on the interface such that $\mathbf{LPV}$ is a valid refracted light path. Our algorithm finds solutions on one triangle at a time, and it is applied to every triangle in the mesh. It also provides an additional culling criterion that allows us to discard triangles that cannot contain a solution early on in the algorithm.

Refraction can be encoded by an equation of the form $\mathbf{f}(\mathbf{P}) = 0$, where the *refraction constraint function* $\mathbf{f}$ depends on a point $\mathbf{P}$ on the plane of the considered triangle. Snell's law also states that the incident ray, the outgoing ray and the shading normal at the intersection point must be coplanar. This constraint defines a *coplanarity conic* section in the triangle plane, on which all solutions to the refraction problem necessarily lie. This restricts the search space to a one-dimensional curve on the triangle plane.

The refraction constraint can be reformulated as a univariate polynomial on the position of a point over any line in the plane. By approximating the coplanarity conic by its chords, we can then define the refraction constraint over this approximation. A single chord is a poor approximation of a conic section. By recursively splitting chords whose distance to the conic is larger than a user-specified threshold, we can progressively refine the approximation.

Then, for each one of these chords, we compute the corresponding refraction constraint and find its roots over the chord. Real-root isolation methods provide information to Newton's method on the number of roots of the function and their location on the chord. Once the roots have been

found, they are projected onto the conic section for accuracy. Finally, once all solutions have been found, we compute the contribution of these refracted light paths to the illumination of the scene.

The algorithm can be split in three different parts, which will structure the contribution. The first part discusses every aspect related to reducing the dimensionality of the problem. The second part then explains how to find the solutions to the problem using its new formulation. Finally, the third describes how the contribution of the newfound refracted light paths is computed.

5 Dimensionality reduction

The first stage of the algorithm consists in reducing the dimensionality of the problem. In order to do this, we first show that the coplanarity constraint provided by Snell's law can be represented as a conic section over the triangle plane. Then, we define the refraction constraint as a univariate polynomial on the position of a point on a straight line, successfully reducing the dimensionality of the problem.

5.1 Refraction constraint function

We first formally define the parameters of the problem, and formulate the refraction problem as a function of these parameters. We provide a geometric visualisation to help developing an intuition about the complexity of the problem.

Let us consider a scene containing a light source and a participating medium enclosed in a refractive interface represented as a triangle mesh with interpolated normals. Let us denote the medium sample by $\mathbf{V}$ and the light sample by $\mathbf{L}$. For the algorithm, we will consider a single triangle defined by three vertices $\mathbf{P}_0$, $\mathbf{P}_1$ and $\mathbf{P}_2$, with corresponding normalised vertex shading normals $\hat{\mathbf{n}}_0$, $\hat{\mathbf{n}}_1$ and $\hat{\mathbf{n}}_2$. In the following, $\mathbf{n}_s(\mathbf{P})$ will denote the interpolated shading normal at $\mathbf{P}$, and $\hat{\mathbf{n}}_g$ the normalised geometric normal of the triangle. Finally, η will denote the index of refraction of the medium. The main parameters are illustrated in figure 15.

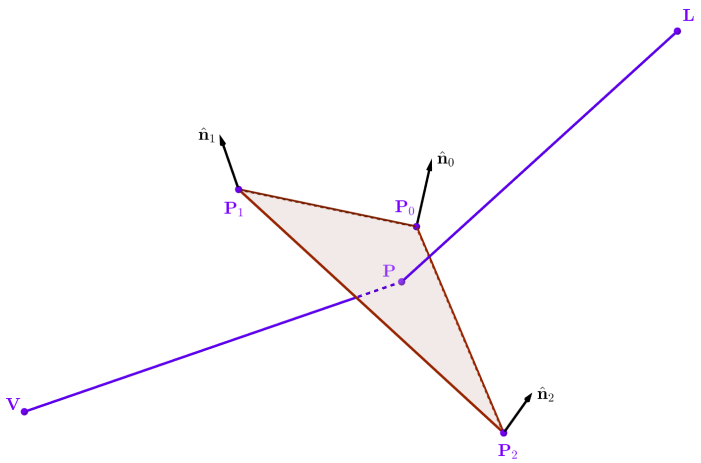


Figure 15: Illustration of the scene

Before introducing the algorithm, we need to define the problem we intend to solve. Snell's law of refraction can be expressed as a function of the scene parameters defined above by replacing the sines by the corresponding cross products. The previous approach from Walter *et al.* [36]

used the half-vector formulation to encode the refraction constraint. This formulation included the coplanarity constraint. Since we are treating coplanarity separately, using a scalar constraint function will result in an easier root-finding process.

$$
\begin{aligned}
& \eta \sin\theta_i = \sin\theta_o \\
\Leftrightarrow \quad & \eta \frac{\big\|(\mathbf{V}-\mathbf{P}) \wedge \mathbf{n}_s(\mathbf{P})\big\|}{\|\mathbf{V}-\mathbf{P}\|\big\|\mathbf{n}_s(\mathbf{P})\big\|} = \frac{\big\|(\mathbf{L}-\mathbf{P}) \wedge \mathbf{n}_s(\mathbf{P})\big\|}{\|\mathbf{L}-\mathbf{P}\|\big\|\mathbf{n}_s(\mathbf{P})\big\|} \\
\Leftrightarrow \quad & \eta\|\mathbf{L}-\mathbf{P}\|\big\|(\mathbf{V}-\mathbf{P}) \wedge \mathbf{n}_s(\mathbf{P})\big\| - \|\mathbf{V}-\mathbf{P}\|\big\|(\mathbf{L}-\mathbf{P}) \wedge \mathbf{n}_s(\mathbf{P})\big\| = 0 \\
\Leftrightarrow \quad & \eta^2\|\mathbf{L}-\mathbf{P}\|^2\big\|(\mathbf{V}-\mathbf{P}) \wedge \mathbf{n}_s(\mathbf{P})\big\|^2 - \|\mathbf{V}-\mathbf{P}\|^2\big\|(\mathbf{L}-\mathbf{P}) \wedge \mathbf{n}_s(\mathbf{P})\big\|^2 = 0 \\
\Leftrightarrow \quad & \mathbf{f}(\mathbf{P}) = 0.
\end{aligned}
$$

The problem of finding all valid paths refracted through a triangle according to Snell's law is equivalent to finding the roots of the constraint function $\mathbf{f}$ over the triangle such that $\overrightarrow{\mathbf{PV}}$, $\overrightarrow{\mathbf{PL}}$ and $\mathbf{n}_s$ are coplanar. A point $\mathbf{P}$ on the triangle can be represented by two of its barycentric coordinates (u, v, w). Then the $\|\mathbf{X}-\mathbf{P}\|^2$ terms are quadratic polynomials, and every coordinate of the vectors $(\mathbf{X}-\mathbf{P}) \wedge \mathbf{n}_s(\mathbf{P})$ is a quadratic polynomial, so the squared norm of these vectors are quartic polynomials. The constraint function is then a bivariate sixth degree polynomial in u and v. Finding the roots of a polynomial function with numerical methods is often easier and faster than for other functions, which accounts for the squaring step in the previous calculation.

Snell's law also states that if $\mathbf{P}$ is a solution to our problem, then $\overrightarrow{\mathbf{PV}}$, $\overrightarrow{\mathbf{PL}}$ and $\mathbf{n}_s(\mathbf{P})$ are coplanar. Coplanarity is a necessary condition for refraction, so the solutions to our problem will all lie in the set of points $\mathbf{P}$ satisfying the coplanarity constraint. Moreover, this constraint defines a conic section over the triangle plane. Our approach consists in using this condition to reduce the dimensionality of the problem in order to make the algorithm faster and more stable than the previous bidimensional methods.

By plotting the constraint polynomial zero set and the coplanarity conic in the triangle plane, we can get a better idea of the geometric complexity of the problem. The images in figure 16 were obtained by modifying the values of the vertex shading normals $\hat{\mathbf{n}}_0$, $\hat{\mathbf{n}}_1$ and $\hat{\mathbf{n}}_2$ with all other parameters fixed. The goal of our algorithm is to find the intersection of the red curve and the blue curve, i.e. the constraint function zero set and the coplanarity curve. As we can see, slight modifications to the parameters of the scene can have huge repercussions in the considered curves. Although the geometric properties of conic sections are well known, there is little information on the properties of the zero set of bivariate polynomials.

This reasoning is also valid for the law of reflection. Let us consider for a moment that $\mathbf{L}$ and $\mathbf{V}$ lie on the same side of the triangle, and that our goal is to find reflected paths $\mathbf{LPV}$ such that $\mathbf{P}$ belongs to the triangle. The law of reflection states that the incidence angle θ_i is equal to the reflection angle θ_o. Since these angles belong to the interval $[0, \frac{\pi}{2}]$ and the sine function is bijective over the interval, this is equivalent to saying that $\sin\theta_i = \sin\theta_o$. This allows us to define the reflection constraint as a function of $\mathbf{P}$ by setting η to one in the refraction constraint function $\mathbf{f}(\mathbf{P})$. Since the coplanarity condition is also necessary to reflection, the whole next-event estimation algorithm can also be applied to reflection. The reflection version needs more sophisticated culling tests for it to scale to large scenes, since the algorithm would have to be applied to every reflective surface in the scene. In the refraction case, we only need to apply the algorithm to the refractive interface. In the rest of the report, we will exclusively refer to the refraction case.

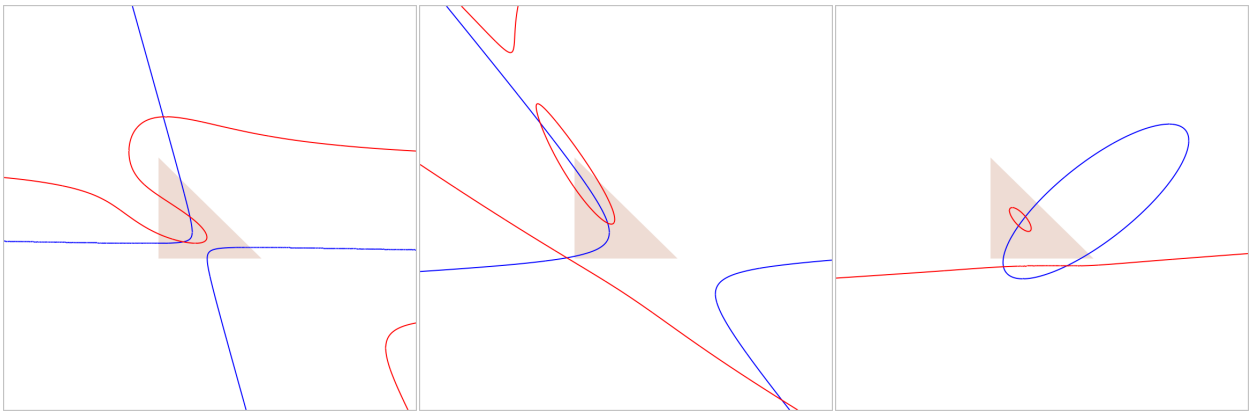


Figure 16: Geometric representation of the problem in the triangle plane. The zero set of the constraint function is plotted in red and the coplanarity conic is plotted in blue. The goal of the algorithm is to find the intersection of these two curves.

5.2 Coplanarity conic section

As we have seen, the complexity of the problem in its natural form pushes us to find a simplified form of the problem. Since coplanarity is a necessary condition for refraction, we need only look at the roots of the constraint function over the one-dimensional coplanarity conic. This simplifies the problem by allowing us to use the well-known geometric properties of conic sections.

5.2.1 Conic equation

Our goal in this section is to reformulate the coplanarity constraint of Snell's law as an equation of the form $Au^2 + Buv + Cv^2 + Du + Ev + F = 0$, where the coefficients A through F depend on the parameters of the problem and (u, v) are the barycentric coordinates of $\mathbf{P}$.

A point $\mathbf{P}$ on the triangle plane and the shading normal $\mathbf{n}_s(\mathbf{P})$ at that point can be expressed as a function of the triangle's barycentric coordinates (u, v). These coordinates are illustrated in figure 17. They can also be interpreted as the ratio of the area of each subtriangle to the area of the larger triangle.

$$\begin{cases} \mathbf{P}(u,v) = \mathbf{P}_0 + u \cdot \mathbf{p}_{10} + v \cdot \mathbf{p}_{20}, \text{ where } \mathbf{p}_{10} = \mathbf{P}_1 - \mathbf{P}_0 \text{ and } \mathbf{p}_{20} = \mathbf{P}_2 - \mathbf{P}_0 \\ \mathbf{n}_s(u,v) = \hat{\mathbf{n}}_0 + u \cdot \mathbf{n}_{10} + v \cdot \mathbf{n}_{20}, \text{ where } \mathbf{n}_{10} = \hat{\mathbf{n}}_1 - \hat{\mathbf{n}}_0 \text{ and } \mathbf{n}_{20} = \hat{\mathbf{n}}_2 - \hat{\mathbf{n}}_0 \end{cases}$$

$$\mathbf{P}(u,v) = \bar{P} \cdot \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \text{ and } \mathbf{n}_s(a,b) = \bar{N} \cdot \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}; \quad \text{where } \bar{P} = \begin{bmatrix} | & | & | \\ \mathbf{p}_{10} & \mathbf{p}_{20} & \mathbf{P}_0 \\ | & | & | \end{bmatrix} \text{ and } \bar{N} = \begin{bmatrix} | & | & | \\ \mathbf{n}_{10} & \mathbf{n}_{20} & \hat{\mathbf{n}}_0 \\ | & | & | \end{bmatrix}.$$

In barycentric space, $\mathbf{P}_0$, $\mathbf{P}_1$ and $\mathbf{P}_2$ respectively occupy the coordinates $(0, 0)$, $(1, 0)$ and $(0, 1)$ of the plane, allowing us to work in a space independent from the shape of the triangle itself.

Three vectors $\mathbf{x}_0$, $\mathbf{x}_1$ and $\mathbf{x}_2$ are coplanar if and only if $\mathbf{x}_0 \cdot \mathbf{x}_1 \wedge \mathbf{x}_2 = 0$. If $\mathbf{s} = \mathbf{L} - \mathbf{V}$, then

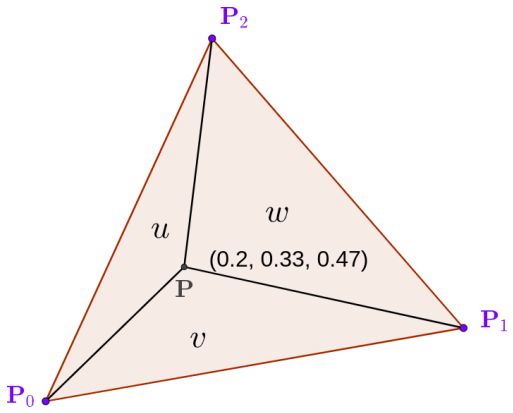


Figure 17: Barycentric coordinates

$\mathbf{n}_s(\mathbf{P})$, $\overrightarrow{\mathbf{PL}}$ and $\overrightarrow{\mathbf{PV}}$ are coplanar if and only if:

$$
\begin{aligned}
& \mathbf{n}_s(u,v) \cdot [(\mathbf{L} - \mathbf{P}(u,v)) \wedge (\mathbf{V} - \mathbf{P}(u,v))] = 0 \\
\Leftrightarrow\quad & \mathbf{n}_s(u,v) \cdot (\mathbf{L} \wedge \mathbf{V} - \mathbf{L} \wedge \mathbf{P}(u,v) - \mathbf{P}(u,v) \wedge \mathbf{V}) = 0 \\
\Leftrightarrow\quad & \mathbf{n}_s(u,v) \cdot \mathbf{L} \wedge \mathbf{V} + \mathbf{n}_s(u,v) \cdot \mathbf{P}(u,v) \wedge (\mathbf{L} - \mathbf{V}) = 0 \\
\Leftrightarrow\quad & [\mathbf{n}_{10} \cdot \mathbf{p}_{10} \wedge \mathbf{s}] \cdot u^2 + \\
& [\mathbf{n}_{10} \cdot \mathbf{p}_{20} \wedge \mathbf{s} + \mathbf{n}_{20} \cdot \mathbf{p}_{10} \wedge \mathbf{s}] \cdot uv + \\
& [\mathbf{n}_{20} \cdot \mathbf{p}_{20} \wedge \mathbf{s}] \cdot v^2 + \\
& [\hat{\mathbf{n}}_0 \cdot \mathbf{p}_{10} \wedge \mathbf{s} + \mathbf{n}_{10} \cdot \mathbf{P}_0 \wedge \mathbf{s} + \mathbf{n}_{10} \cdot \mathbf{L} \wedge \mathbf{V}] \cdot u + \\
& [\hat{\mathbf{n}}_0 \cdot \mathbf{p}_{20} \wedge \mathbf{s} + \mathbf{n}_{20} \cdot \mathbf{P}_0 \wedge \mathbf{s} + \mathbf{n}_{20} \cdot \mathbf{L} \wedge \mathbf{V}] \cdot v + \\
& [\hat{\mathbf{n}}_0 \cdot \mathbf{P}_0 \wedge \mathbf{s} + \hat{\mathbf{n}}_0 \cdot \mathbf{L} \wedge \mathbf{V}] = 0 \\
\Leftrightarrow\quad & Q(u,v) = Au^2 + Buv + Cv^2 + Du + Ev + F = 0.
\end{aligned}
$$

This defines a conic section in the barycentric space of the triangle. The conic section necessarily contains all the solution points $\mathbf{P}$ such that $\mathbf{LPV}$ is a valid light path refracted through the triangle. This provides us with a one-dimensional search space for Newton's method to operate on.

5.2.2 Conic classification

Conic sections come in different types: they can be ellipses, parabolas or hyperbolas, either degenerate or non-degenerate. Some geometrical properties of the conic depend on its type. For this reason it is important that we identify the type of the coplanarity conic and adapt the algorithm to the type at hand.

Figure 18 summarises all of the different possible cases and the classification criteria. All of the cases are illustrated in figure 19. The conic section can be classified by looking at the determinant of the following matrices:

$$
A_Q = \begin{bmatrix} A & B/2 & D/2 \\ B/2 & C & E/2 \\ D/2 & E/2 & F \end{bmatrix}; \qquad A_{33} = \begin{bmatrix} A & B/2 \\ B/2 & C \end{bmatrix}.
$$

	$\lvert A_Q\rvert \neq 0$	$\lvert A_Q\rvert = 0$
$\lvert A_{33}\rvert < 0$	Hyperbola	Intersecting lines
$\lvert A_{33}\rvert = 0$	Parabola	Parallel lines†
$\lvert A_{33}\rvert > 0$	Ellipse*	Single point

$$(*)\begin{cases}(A+C)|A_Q|<0\text{: real ellipse}\\(A+C)|A_Q|>0\text{: imaginary ellipse}\end{cases};\qquad(\dagger)\begin{cases}D^2+E^2>4(A+C)F\text{: real and distinct}\\D^2+E^2=4(A+C)F\text{: real and coincident}\\D^2+E^2<4(A+C)F\text{: non-existent in }\mathbb{R}\end{cases}$$

Figure 18: Conic section classification, taken from [38, 39]

Figure 19: Illustration of the different types of conic sections and their degenerate counterparts

In practice, we will only consider hyperbolas and ellipses. A conic section can only be degenerate or a parabola if one of the determinants is equal to zero. Compared to the other possibilities, these pathological cases are extremely infrequent. The current implementation of the algorithm does not take them into account: if the coplanarity conic for a triangle is singular, the triangle is then discarded. From now on, we will consider that the conic section is either a hyperbola or an ellipse, also called *central conics*. Pathological cases arising from similar situations will be discussed in section 8.2.

5.2.3 Triangle-conic intersection

For each triangle, our algorithm only looks for solutions lying inside of the triangle. Now that we have the equation of the conic section, we must find which parts of it are inside of the triangle. The easiest way of doing this is by computing the intersections between the triangle and the conic.

The intersection points are given by the roots in $[0, 1]$ interval of the following quadratic equa-

tions: $Q(u, 0) = 0$, $Q(0, v) = 0$ and $Q(u, 1 - u) = 0$ with $u, v \in \mathbb{R}$, where Q is the quadratic form defining the conic section. If no roots are found, then the conic does not intersect the border of the triangle. This only happens when the conic is either fully inside or fully outside of the triangle. By taking any point on the conic and testing whether it lies inside of the triangle, this provides a very fast method for culling triangles that cannot contain any solution to the problem.

Since each one of these equations can have up to two distinct real roots, a conic section can intersect the triangle at most six times. Note that it is theoretically possible that these equations have double roots. A double root means that the conic section is tangent to the corresponding side of the triangle. As for the parabolic and degenerate cases, that this is an infrequent pathological case not currently handled by the implementation.

Discounting the double root case, there is always an even number of intersections between the conic section and the triangle. Still, the triangle and the conic section can intersect in a multitude of ways. Assuming that the conic is never tangent to the triangle, a complete categorisation of the possible cases is given in figure 20. Note that for the hyperbola, it is important to consider the number of intersections of each branch. From a combinatorial point of view, the only missing case in this classification is a hyperbola intersecting the triangle four times with one branch and twice with the other branch. A sketch of the proof that this case is impossible is given in appendix B.

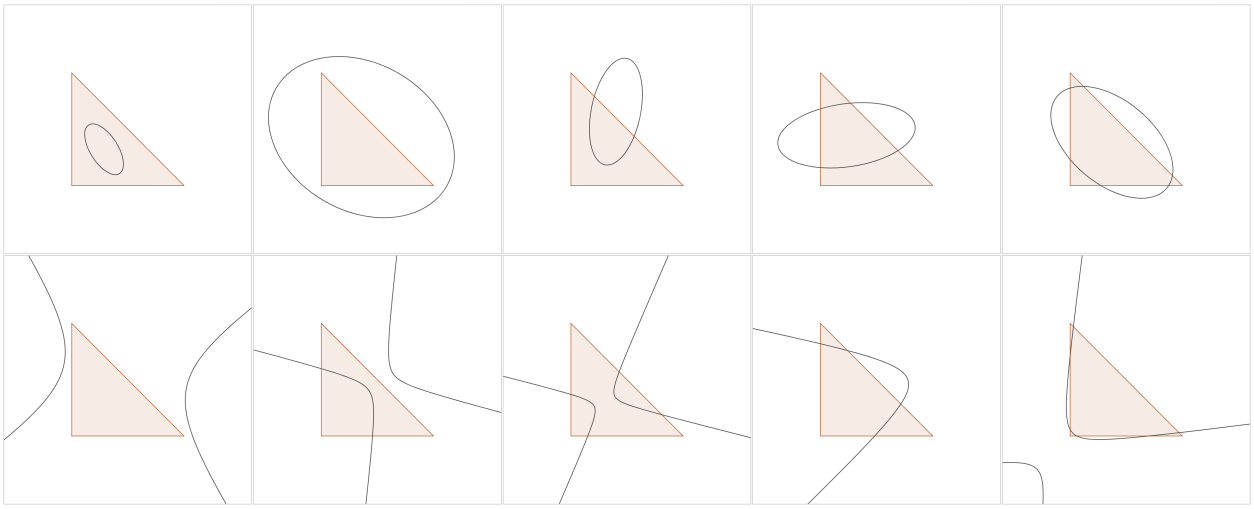


Figure 20: Triangle-conic intersection classification: a triangle and an ellipse can have zero intersections (ellipse completely inside or outside the triangle), two intersections, four intersections and six intersections; a triangle and a hyperbola can have zero intersections, two intersections, four intersections (two on each branch or all four on the same branch) and six intersections all on the same branch

5.2.4 Intersection pairs

Depending on the configuration of the triangle and the conic, there can be up to three conic arcs lying inside of the triangle. Each one of these arcs is delimited by a pair of intersection points, but the list of these points does not tell us directly which parts of the conic lie inside the triangle.The construction of these pairs of points is not trivial and depends on the exact configuration of the

triangle and the conic. The explanation is divided depending on the number of intersections found at the previous stage of the algorithm.

The implicit form $Q(u, v) = 0$ of the conic section is not appropriate when manipulating and choosing points on the conic section. Instead, we use a parametric form of the conic, where each point of the conic corresponds to a value of a parameter t. Trigonometric functions provide a very simple parametrisation for circles and square hyperbolas, i.e. $(\cos t, \sin t)$ and $(\pm \cosh t, \sinh t)$ respectively. By applying an affine transformation to the coplanarity conic, we can transform it into its unit counterpart and compute the value of t. The details on how the affine transformation is computed are explained in appendix A.

No intersection points If there is no intersection between the triangle and the conic, then the triangle can be discarded except in the case where the ellipse is completely inside of the triangle. This case can be simply detected by taking any point on the conic section and checking if it is inside of the triangle

Two intersection points This is the easiest case to handle since we can only choose a single pair of intersection points. These automatically delimit the conic arc inside the triangle.

Four and six intersection points This is the hardest case to handle, and it can happen in multiple ways. The easiest to deal with is a hyperbola where each branch intersects the triangle twice. The pairs of points can simply be created by pairing the intersection points lying on the same branch together. The other cases are an ellipse intersecting the triangle four or six times, and a hyperbola where a single branch intersects the triangle four or six times.

Given the list of the triangle-conic intersection points, we can compute the value of their respective conic parameters. By sorting this list by increasing parameter value, each point in the list is surrounded by two points contiguous to it on the conic section. In other words, by taking two consecutive points in the list, there is no intersection point on the conic arc delimited by these points. Note that the last and first intersection points in the list are considered as being consecutive in the case of an ellipse.

Also, at each intersection point the curve crosses the border of the triangle, so on one side of the border the conic lies inside of it and on the other side it lies outside of it. Then, given a pair of consecutive points, if any point on the arc defined by these points lies inside of the triangle, then the whole conic arc does.

Let us explain the method on an example: an ellipse with four intersection points $[\mathbf{Q}_0, \mathbf{Q}_1, \mathbf{Q}_2, \mathbf{Q}_3]$ sorted by increasing conic parameter as illustrated in figure 21. If $\widehat{\mathbf{Q}_0\mathbf{Q}_1}$ lies inside of the triangle, then $\widehat{\mathbf{Q}_1\mathbf{Q}_2}$ lies outside the triangle, $\widehat{\mathbf{Q}_2\mathbf{Q}_3}$ lies inside, and $\widehat{\mathbf{Q}_3\mathbf{Q}_0}$ lies outside. If $\widehat{\mathbf{Q}_0\mathbf{Q}_1}$ laid outside, then the result would be the opposite for every conic arc.

More generally, sorting the list by parameter value guarantees that the first two points being an inside pair determines whether any pair of consecutive points in the list is an inside pair or not. This method applies to any of the aforementioned triangle-conic configurations.

This concludes the part of the algorithm that consists in restricting the search space to a one-dimensional space. The solutions we are looking for all lie on a conic arc inside of the triangle delimited by a pair of points belonging to the border of the triangle and the coplanarity conic.

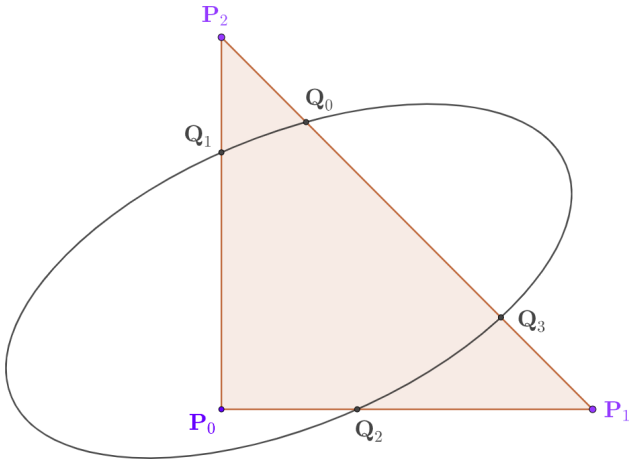


Figure 21: Four triangle-conic intersections

5.3 Dimensionality reduction

Now that the search space has been reduced as much as possible, we need to find a way to define the refraction constraint function over this space. A conic arc can be approximated up to an arbitrary precision by conic chords, i.e. line segments whose endpoints both lie on the conic. Our approach is to define the constraint function over conic chords instead of conic arcs. If the approximation of the arc by chords is sufficiently precise, due to the continuous nature of the problem, the roots of the constraint function on the chord must necessarily be close to the roots on the conic arc.

5.3.1 Constraint over chords

Let $\mathbf{Q}_0$ and $\mathbf{Q}_1$ be points in barycentric space and $\mathbf{Q}(t) = (1-t)\mathbf{Q}_0 + t\mathbf{Q}_1$ a point on the line passing through these two points. By substituting (u, v) in the constraint function by the coordinates of $\mathbf{Q}(t)$, we obtain the new formulation of the constraint as a function of the position t of a point $\mathbf{Q}(t)$ on the line defined by $\mathbf{Q}_0$ and $\mathbf{Q}_1$. In the following, $\mathbf{Q}_0$ and $\mathbf{Q}_1$ are written in homogeneous form and $\mathbf{X}$ is a placeholder for either $\mathbf{V}$ or $\mathbf{L}$. $\bar{P}$ and $\bar{N}$ are those defined in section 5.2.1.

$$
\begin{aligned}
\mathbf{Q}(t) &= (1-t)\mathbf{Q}_0 + t\mathbf{Q}_1 = \mathbf{Q}_0 + t\Delta = (u, v, 1)^T, \text{ where } \Delta = \mathbf{Q}_1 - \mathbf{Q}_0 \\
\mathbf{P}(t) &= \bar{P} \cdot (u, v, 1)^T = \bar{P}\mathbf{Q}_0 + t\bar{P}\Delta = \mathbf{P}_c + t\mathbf{P}_t, \text{ where } \mathbf{P}_c = \bar{P}\mathbf{Q}_0 \text{ and } \mathbf{P}_t = \bar{P}\Delta \\
\mathbf{n}_s(t) &= \bar{N} \cdot (u, v, 1)^T = \bar{N}\mathbf{Q}_0 + t\bar{N}\Delta = \mathbf{n}_c + t\mathbf{n}_t, \text{ where } \mathbf{n}_c = \bar{N}\mathbf{Q}_0 \text{ and } \mathbf{n}_t = \bar{N}\Delta
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{g}_\mathbf{X}(t) = \left\|\mathbf{X} - \mathbf{P}(t)\right\|^2 &= \left\|(\mathbf{X} - \mathbf{P}_c) - t\mathbf{P}_t\right\|^2 \\
&= \|\mathbf{X} - \mathbf{P}_c\|^2 - 2t\langle \mathbf{X} - \mathbf{P}_c, \mathbf{P}_t \rangle + t^2 \|\mathbf{P}_t\|^2
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{h}_{\mathbf{X}}(t) = \left\|\mathbf{n}_s(t) \wedge (\mathbf{X} - \mathbf{P}(t))\right\|^2 &= \left\|(\mathbf{n}_c + t\mathbf{n}_t) \wedge (\mathbf{X} - \mathbf{P}_c - t\mathbf{P}_t)\right\|^2 \\
&= \left\|\mathbf{n}_c \wedge (\mathbf{X} - \mathbf{P}_c) + t[\mathbf{n}_t \wedge (\mathbf{X} - \mathbf{P}_c) + \mathbf{P}_t \wedge \mathbf{n}_c] + t^2 \mathbf{P}_t \wedge \mathbf{n}_t\right\|^2 \\
&= \left\|\mathbf{n}_c \wedge (\mathbf{X} - \mathbf{P}_c)\right\|^2 + \\
&\quad 2\langle \mathbf{n}_c \wedge (\mathbf{X} - \mathbf{P}_c), \mathbf{n}_t \wedge (\mathbf{X} - \mathbf{P}_c) + \mathbf{P}_t \wedge \mathbf{n}_c \rangle t + \\
&\quad \left[\left\|\mathbf{n}_t \wedge (\mathbf{X} - \mathbf{P}_c) + \mathbf{P}_t \wedge \mathbf{n}_c\right\|^2 + 2\langle \mathbf{n}_c \wedge (\mathbf{X} - \mathbf{P}_c), \mathbf{P}_t \wedge \mathbf{n}_t \rangle\right] t^2 + \\
&\quad 2\langle \mathbf{n}_t \wedge (\mathbf{X} - \mathbf{P}_c) + \mathbf{P}_t \wedge \mathbf{n}_c, \mathbf{P}_t \wedge \mathbf{n}_t \rangle t^3 + \\
&\quad \left\|\mathbf{P}_t \wedge \mathbf{n}_t\right\|^2 t^4
\end{aligned}
$$

The constraint function is defined in terms of these factors as:

$$\mathbf{f}(t) = \eta^2 \cdot \mathbf{g}_{\mathbf{L}}(t) \cdot \mathbf{h}_{\mathbf{V}}(t) - \mathbf{g}_{\mathbf{V}}(t) \cdot \mathbf{h}_{\mathbf{L}}(t).$$

Since $\mathbf{g}_{\mathbf{X}}$ is a quadratic polynomial, and $\mathbf{h}_{\mathbf{X}}$ is a quartic polynomial, their product is a sixth degree polynomial. Therefore, $\mathbf{f}(t)$ is a sixth degree univariate polynomial in t. We will refer to these polynomials as *chord polynomials*. This parametrisation successfully reduces the dimensionality of the problem. Tests have shown that the coefficients of this polynomial can span multiple orders of magnitude, going from 10^0 to 10^{10} for some triangles in our test scene. This may raise numerical accuracy problems when manipulating chord polynomials.

Our original solution was to use the trigonometric parametrisation presented in appendix A. Unfortunately, this substitution introduces circular or hyperbolic trigonometric functions into the constraint function, which loses its polynomial form. Since the polynomial form of the problem is very useful for numerical algorithms, we opted for the chord parametrisation instead.

Now remains the problem of approximating a conic arc by its chords and finding the roots of the constraint function over them.

5.3.2 Chord approximation

The refraction problem formulation over straight lines has useful properties for numerical algorithms. Unfortunately, since our search space consists of conic arcs, we need to find a way to approximate them with straight lines. A conic arc can be approximated by the chord passing through the points defining the arc. In most cases, this is a very poor approximation of a conic arc. Our approach consists in recursively splitting this chord in two until the distance of every resulting chord to the conic is smaller than a user-specified threshold.

For a given arc, the algorithm starts with the chord defined by its triangle-conic intersection pair. The quality of the approximation is given by the maximum orthogonal distance between the chord and the arc. Then, we compute the point on the arc maximising this distance. The two chords defined by the original points and the new point provide a better approximation. This process is illustrated in figure 22. The subdivision is repeated recursively until every chord is sufficiently close to the conic arc it subtends. We only need a method to compute the point maximising the orthogonal distance between the chord and the conic, and a way to compute this distance.

The point can be found by computing the conic parameters of the two points defining the chord, taking their average and converting the resulting parameter to a point in barycentric space. Let us prove that this method works. The point on the conic maximising the orthogonal distance to the

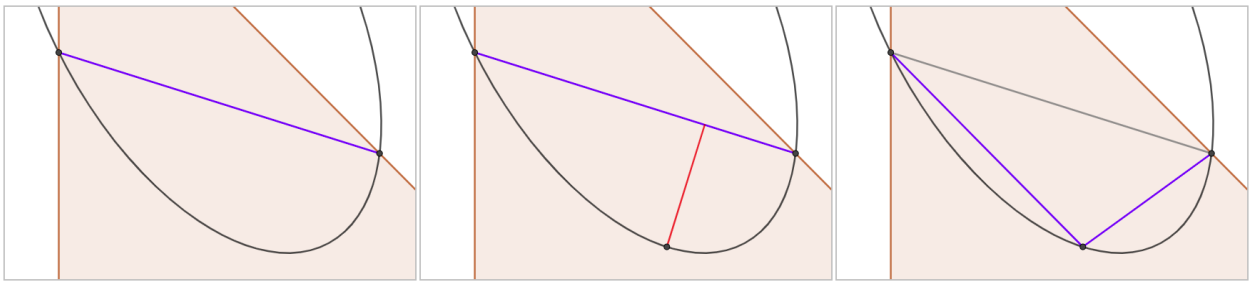


Figure 22: Chord subdivision algorithm: the successive chord approximations are drawn in blue, the maximum distance between a chord and the conic arc is represented in red

chord is the point whose tangent is parallel to the chord. Since parallelism is preserved by affine transformations, we only need to prove it for circles and unit hyperbolas.

The proof for hyperbolas and circles follows the same structure and reasoning. For this reason, we will only prove it for a hyperbolic arc. Let us define a unit hyperbolic branch consisting of the points $\mathbf{X}(t) = (\cosh t, \sinh t)^T$. Let a, b be two parameters corresponding to two points $\mathbf{X}(a), \mathbf{X}(b)$ on the branch defining a chord. Our goal is to find the parameter t such that the orthogonal distance between $\mathbf{X}(t)$ and the chord is maximal. If we let $\Delta = \mathbf{X}(b) - \mathbf{X}(a)$, we can define $\Delta^\perp = (\Delta_y, -\Delta_x)^T$ such that $\langle \Delta, \Delta^\perp \rangle = 0$. Then t is such that:

$$
\begin{aligned}
& \langle \mathbf{X}'(t), \Delta^\perp \rangle = 0 \\
\Leftrightarrow \quad & \sinh t(\sinh b - \sinh a) = \cosh t(\cosh b - \cosh a) \\
\Leftrightarrow \quad & (e^t - e^{-t})(e^b - e^a - e^{-b} + e^{-a}) = (e^t + e^{-t})(e^b - e^a + e^{-b} - e^{-a}) \\
\Leftrightarrow \quad & e^t(2e^{-a} - 2e^{-b}) + e^{-t}(2e^a - 2e^b) = 0 \\
\Leftrightarrow \quad & e^{2t} = \frac{e^a - e^b}{e^{-b} - e^{-a}} \\
\Leftrightarrow \quad & e^{2t} = \frac{e^{\frac{a+b}{2}}(e^{\frac{a-b}{2}} - e^{\frac{b-a}{2}})}{e^{-\frac{a+b}{2}}(e^{\frac{a-b}{2}} - e^{\frac{b-a}{2}})} \\
\Leftrightarrow \quad & e^{2t} = e^{a+b} \\
\Leftrightarrow \quad & t = \frac{a+b}{2}
\end{aligned}
$$

Finally, given a chord defined by the points $\mathbf{Q}_0$ and $\mathbf{Q}_1$, and a point $\mathbf{Q}$ on the arc $\widehat{\mathbf{Q}_0\mathbf{Q}_1}$, the orthogonal distance between them is equal to $||\overrightarrow{\mathbf{Q}_1\mathbf{Q}_0} \wedge \overrightarrow{\mathbf{Q}\mathbf{Q}_0}||/||\overrightarrow{\mathbf{Q}_1\mathbf{Q}_0}||$.

In this section, we have taken Snell's law of refraction and separately considered the relationship between the angles of incidence and refraction, and the coplanarity condition on the shading normal, the incident ray and the outgoing ray. From the first part of the law, we derived the constraint function which we then expressed as a one-dimensional function of the position of a point on a line. The second part of the law helped us restrict the search space for our algorithm to the intersection between a conic section and a triangle. The approximation of this conic section by straight chords allows us to unify these two parts and define the constraint function over the reduced search space. The next part of the algorithm relies on this new formulation of the problem.

6 Finding refracted paths

In the last section, we reduced the problem to finding the roots of the chord polynomials defined on the approximation of the conic arcs inside of the triangle. Given a specific chord polynomial, we want to find all of its roots in the interval $[0, 1]$. Newton's method provides an increasingly good approximation of a single root if the function is sufficiently regular. If there are multiple roots in the interval, Newton's method will converge towards one of them. For this reason, we first need information on the number and the location of the roots of the chord polynomial in $[0, 1]$. In this section, we will explain how root-isolation algorithms can give us the necessary information to find all of the roots of the chord polynomial with Newton's method.

6.1 Real-root isolation algorithms

Finding the roots of a polynomial is a very frequent problem in numerical analysis. Polynomial functions arise in many situations and their simplicity is often an advantage. The roots for polynomials of degree up to four can be computed explicitly. Unfortunately, the Abel-Ruffini [1] theorem states that there is no solution in radicals for arbitrary polynomials of degree five or higher. For this reason, root-finding algorithms are usually iterative and compute an increasingly accurate estimation of the roots of the polynomial.

Real-root isolation algorithms take a polynomial as their input and return a set of disjoint intervals, each containing one and only one root of the polynomial, such that their union contains all of the roots. They provide information not only about the number of roots of a polynomial, but also their location, making them ideal for our algorithm. These algorithms come in two categories. The earliest ones were based in *Sturm's theorem* [32] proven by Sturm in 1829. The state-of-the art real-root isolation algorithms mostly rely on *Descartes' rule of signs* [10]. Sturm's theorem provides the exact number of roots of a polynomial in an interval, while Descartes' rule only provides an upper bound.

Both types of algorithm have the same computational complexity, but Descartes-based algorithms have been shown to be faster than Sturm-based algorithms in practice. We present both algorithms in this section, since one of them could have an advantage over the other for our use case, but our initial implementation uses the Descartes-based algorithm. For this reason, the explanation for the Descartes-based algorithm is more complete and thorough. Delving further in the numerical algorithms literature could provide more insight on choosing the best algorithm for our purpose.

In our case, the real-root isolation algorithm is applied to the sixth degree chord polynomial over $[0, 1]$. If the chord polynomial has no roots, the chord can be discarded, avoiding further calculations. If the polynomial has roots in the interval, knowing their approximate location can provide good initial guesses for Newton's method. This section states both theorems and explains the real-root isolation algorithms that build upon them.

These algorithms are intended for square-free polynomials with integer coefficients, but they theoretically work with floating-point coefficients. A polynomial is square-free if it does not have as a factor any square of a non-unit polynomial in its decomposition. In $\mathbb{R}[X]$, it is equivalent to the polynomial being separable, i.e. having no repeated roots. Our implementation considers that chord polynomials are always square-free, but there exist algorithms such as Yun's algorithm [40] that take a polynomial and generate a decomposition in square-free factors.

6.1.1 Vincent-Collins-Akritas algorithm

An overview of the different methods relying on Descartes' rule can be found in [2]. The version we will be referring to is called the *Vincent-Collins-Akritas bisection method* (VCA). The details of the implementation are provided in [29]. VCA takes a polynomial and an interval and returns a set of disjoint intervals each containing a single root, whose union contains all roots of the polynomial in the interval. It relies on Descartes' rule to obtain an upper bound on the number of roots of the polynomial in an interval. In this section, we will consider that VCA always takes $[0, 1]$ as the initial interval.

Descartes' rule of signs Descartes' rule of signs states that the number of sign variations of the coefficients of a polynomial is an upper bound on the number of its positive roots. Let us define what we exactly mean by sign variations and then state the rule.

Definition 6.1 *We define the* $\text{sign}(a)$ *of an element* $a \in \mathbb{R}$ *as follows:*

$$\text{sign}(a) = \begin{cases} 1 & \text{if } a > 0; \\ -1 & \text{if } a < 0; \\ 0 & \text{if } a = 0. \end{cases}$$

We then define the number of sign changes $V(A)$ *of a list* $A = (a_1, \dots a_k)$ *of elements of* $\mathbb{R}^*$ *by induction:*

$$V(a_1) = 0 \text{ and } V(a_1, \dots a_k) = \begin{cases} V(a_1, \dots a_{k-1}) + 1 \text{ if } \text{sign}(a_{k-1}a_k) = -1; \\ V(a_1, \dots a_{k-1}) \text{ if } \text{sign}(a_{k-1}a_k) = 1. \end{cases}$$

Finally, we extend this definition to a list A *of elements of* $\mathbb{R}$. $V(A)$ *is equal to* $V(B)$ *where* B *is the list of elements of* $\mathbb{R}^*$ *obtained by removing the zeroes in* A.

Theorem 6.2 *Let* $P = \sum_{i=0}^{d} a_i X^i \in \mathbb{R}[X]$ *be a polynomial represented by the list of its coefficients* $P = (a_0, \dots a_d)$, *and* $\text{pos}(P)$ *the number of positive real roots of* P *counted with multiplicities. Then* $\text{pos}(P) \leq V(P)$ *and* $V(P) - \text{pos}(P)$ *is even.*

Corollary 6.2.1 *If* $V(P) = 1$ *or* 0 *then* $\text{pos}(P) = 1$ *or* 0 *respectively, i.e. the bound is exact.*

The corollary gives us the base case of the algorithm. If the rule returns either zero or one, then the algorithm stops. If the bound is exactly one, the current interval is returned. If the bound is larger than one, then the interval is bisected and the algorithm is recursively applied to both halves. This bisection step gives its name to the algorithm.

Variable transformations Descartes' rule only applies to positive real roots, but we only want information on roots contained in the interval $[0, 1]$. Similarly, for the bisection step, we want to obtain a bound on the number of roots in $[0, \frac{1}{2}]$ and $[\frac{1}{2}, 1]$. By applying transformations to the variable of the polynomial, we can obtain information on the number of roots of the polynomial in any interval.

Definition 6.3 *Let* $P \in \mathbb{R}[X]$. *We define the following transformations:*

- $R(P)(X) = X^{\deg P} P(1/X)$;
- $H_c(P)(X) = P(cX)$;
- $T_c(P)(X) = P(X + c)$.

We will drop the parentheses after the transformation for the sake of convenience.

Descartes' rule applied to $T_1 R(P) = (X+1)^{\deg P} P(1/(X+1))$ provides information on the roots of the original polynomial P in the interval $]0, 1]$. We can test if 0 is a root of the polynomial independently. To prove this, let us define a bijection h between $[0, +\infty[$ and $]0, 1]$ as

$$h : \left| \begin{array}{ccc} [0,+\infty[& \to &]0,1] \\ x & \mapsto & \frac{1}{x+1} \end{array} \right. .$$

Then

$$\begin{array}{rl} & T_1 RP(x) = 0 \text{ for } x \in [0, +\infty[\\ \Leftrightarrow & (x+1)^{\deg P} P(1/(x+1)) = 0 \\ \Leftrightarrow & P(1/(x+1)) = 0 \\ \Leftrightarrow & P(h(x)) = 0 \text{ where } h(x) \in]0, 1]. \end{array}$$

Bisection Let us assume that we already have a method `DescartesBound` for computing Descartes' bound on the number of roots of a polynomial P over $[0, 1]$. Then, the bisection step only requires us to apply this method to two polynomials Q_L, Q_R whose roots in $[0, 1]$ are in bijection with the roots of P in $[0, \frac{1}{2}]$ and $[\frac{1}{2}, 1]$ respectively. These polynomials are $H_{1/2}P$ and $T_1 H_{1/2} P$. Indeed,

$$P(x) = 0 \text{ for } x \in [0, \tfrac{1}{2}] \Leftrightarrow P(y/2) = H_{1/2}P(y) \text{ for } y \in [0, 1];$$
$$P(x) = 0 \text{ for } x \in [\tfrac{1}{2}, 1] \Leftrightarrow H_{1/2}P(y) \text{ for } y \in [1, 2] \Leftrightarrow H_{1/2}P(z+1) = T_1 H_{1/2} P(z) \text{ for } z \in [0, 1].$$

Pseudo-code The pseudo-code of the VCA algorithm is shown in algorithm 1. The 2^n factor in the bisection step prevents the coefficients from vanishing after a few recursive calls without altering the roots. The version provided in [29] for polynomials with τ-bit integer coefficients has a complexity of $\widetilde{O}(n^4\tau^2)$ where n is the degree of the polynomial and $\widetilde{O}$ represents the complexity ignoring logarithmic factors. The proof is given in [31].

The only missing pieces are a guarantee that the algorithm terminates and a method for computing the coefficients of $T_1 RP$ to implement `DescartesBound`.

Termination This algorithm only terminates if Descartes' bound becomes eventually zero or one after a finite number of bisections. This is guaranteed by Vincent's theorem.

Theorem 6.4 *Let $P \in \mathbb{R}[X]$ be a polynomial of degree n. There exists a $\delta > 0$ such that for all $a, b \in \mathbb{R}_+, |b - a| < \delta$, every polynomial of the form*

$$(1+X)^n P\left(\frac{a + bX}{1 + X}\right)$$

has exactly zero or one sign variations in its coefficients. The second case is only possible if $P(X)$ has a simple root in $]a, b[$.

Algorithm 1: VCA bisection method

input : A polynomial P of degree n and an interval $[a, b]$
output : A list isolList of isolating intervals
functions: `DescartesBound(`P`)` applies Descartes' rule to T_1RP
`AddToIsol(`$[a, b]$`)` adds interval $[a, b]$ to isolList

```
isolList ← ∅;
b ← DescartesBound(P);
if b > 0 then
    if b = 1 then AddToIsol([a, b]);
    else
        Q_L ← 2^n H_{1/2} P;
        VCA(Q_L, [a, (a+b)/2]);
        Q_R ← T_1 Q_1;
        if Q_R(0) = 0 then AddToIsol([(a+b)/2, (a+b)/2]);
        VCA(Q_R, [(a+b)/2, b]);
    end
end
return isolList;
```

This form of Vincent's theorem is proven in [3]. The family of polynomials to which Descartes' rule is applied in the VCA algorithm applied to P correspond to the form indicated in the theorem. The distance $|b - a|$ decreases with each recursive call to the algorithm.

Efficient computation of T_cP To obtain information on the roots of P in the $[0, 1]$ interval, we must compute the number of sign changes in the list of coefficients of $T_1R(P)$. In order to do this, we need to know the effect of the transformations T_c and R on the coefficients of P. While R simply reverses the list of coefficients of a polynomial without changing their value, T_c is much harder to handle. The `DescartesBound` method provided in [29] provides a fast computation of the coefficients of T_cP.

The original article provides no explanation on how and why the method works. For this reason, we have decided to provide an in-depth explanation of the reasoning behind the method. It relies on the Horner polynomials appearing in Horner's algorithm for polynomial evaluation. The notation for Horner polynomials $H_j(X)$ is not to be confused with the notation for the variable transformation $H_cP(X)$, since the transformation is not used in this section.

Definition 6.5 *The initial polynomial $P(X)$ and the translated polynomial $T_cP(X) = P(X + c)$ are defined as:*

$$P(X) = \sum_{i=0}^{n} a_i X^i$$

$$T_cP(X) = P(X + c) = \sum_{i=0}^{n} a_i (X + c)^i = \sum_{i=0}^{n} h_i X^i$$

Definition 6.6 *For $j \in [\![0, n]\!]$, the j-th Horner polynomial associated with P is defined by:*

$$H_j(X) = \sum_{i=j}^{n} a_i X^{i-j} = \sum_{k=o}^{n-j} a_{k+j} X^k$$

such that

$$\begin{cases} H_n(X) = a_n; \\ H_j(X) = XH_{j+1}(X) + a_j; \\ H_0(X) = P(X). \end{cases}$$

Our goal then becomes computing the coefficients of the polynomial $H_0(X + c) = P(X + c) = T_c P(X)$. For this we will use the following property that immediately derives from the construction of the Horner polynomials.

Property 6.6.1 *Horner polynomials satisfy the following recursive relation which is the basis for the method:*

$$H_j(X + c) = XH_{j+1}(X + c) + cH_{j+1}(X + c) + a_j. \qquad (*)$$

Let us provide a notation for the coefficients of the $H_j(X + c)$ polynomials before explaining the algorithm. For any given $j \in [\![0, n]\!]$ we denote by $(h_k^j)_{0 \le k \le n-j}$ the list of coefficients of $H_j(X + c)$ in the basis $(1, X, \ldots, X^{n-j})$ such that:

$$H_j(X + c) = \sum_{k=0}^{n-j} a_{k+j}(X + c)^k = \sum_{k=0}^{n-j} h_k^j X^k$$

With this notation, we have $h_k^0 = h_k$ for any $k \in [\![0, n]\!]$ since $H_0(X+c) = T_c P(X)$. Let us substitute this notation in the property $(*)$:

$$\begin{aligned} & H_j(X + c) = XH_{j+1}(X + c) + cH_{j+1}(X + c) + a_j \\ \Leftrightarrow \quad & \sum_{k=0}^{n-j} h_k^j X^k = X \cdot \sum_{k=0}^{n-j-1} h_k^{j+1} X^k + c \cdot \sum_{k=0}^{n-j-1} h_k^{j+1} X^k + a_j \\ \Leftrightarrow \quad & \sum_{k=0}^{n-j} h_k^j X^k = (a_j + c \cdot h_0^{j+1}) + \sum_{k=1}^{n-j-1} \left(h_{k-1}^{j+1} + c \cdot h_k^{j+1} \right) X^k + h_{n-j-1}^{j+1} X^{n-j}. \end{aligned}$$

From the uniqueness of the coefficients of $H_j(X + c)$ in the power basis $(1, X, \ldots, X^{n-j})$ we obtain:

$$\forall j \in [\![0, n]\!], \begin{cases} h_0^j = a_j + c \cdot h_0^{j+1} \\ h_k^j = h_{k-1}^{j+1} + c \cdot h_k^{j+1}, \forall k \in [\![1, n - j - 1]\!] \\ h_{n-j}^j = h_{n-j-1}^{j+1}. \end{cases}$$

Let us defining the operator $+_c$ such that for any $x, y \in \mathbb{R}$, $x +_c y = x + c \cdot y$. By extending the h_k^j notation such that

$$\forall j \in [\![0, n]\!], \begin{cases} h_{-1}^{j+1} = a_j \\ h_{n-j}^{j+1} = 0 \end{cases}$$

we can summarise the previous set of rules with a single equation:

$$\forall j \in [\![0, n]\!], \forall k \in [\![0, n-j]\!], h_k^j = h_{k-1}^{j+1} +_c h_k^{j+1}.$$

This formula is illustrated in figure 23. By organising the coefficients (h_k^j) in a grid, the previous equation says that a coefficient is the result of applying the $+_c$ operator to its top-left and top neighbours. The black nodes in the figure represent the coefficients of the Horner polynomials while the green and blue ones represent the nodes resulting from extending the h_k^j notation. To compute $T_cP(X)$, we only need to go down the pyramid from the node $h_0^n = a_n$ by applying the rule until we reach the $j = 0$ layer.

The algorithm 2 given in the figure takes the coefficients of the initial polynomial $P(X)$ and computes the coefficients of $T_cP(X)$ in-place. At each iteration of the variable i, the algorithm progresses down the pyramid. The variable $P[m]$ successively takes the values h_k^j such that $k+j = m$, starting at h_{-1}^{m+1} and ending at h_m^0.

Algorithm 2: $T_cP(X)$

input : $P[m] = a_m$ for all $m \leq n$
output: $P[m] = h_m$ for all $m \leq n$
for $i \leftarrow 1$ **to** n **do**
  for $m \leftarrow n - i$ **to** $n - 1$ **do**
    $P[m]$ += $cP[m + 1]$
  end
end

Example for $\boldsymbol{n = 3}$:

- $\boldsymbol{i = 1}$:
  - $P[2]$ += $cP[3]$
- $\boldsymbol{i = 2}$:
  - $P[1]$ += $cP[2]$
  - $P[2]$ += $cP[3]$
- $\boldsymbol{i = 3}$:
  - $P[0]$ += $cP[1]$
  - $P[1]$ += $cP[2]$
  - $P[2]$ += $cP[3]$

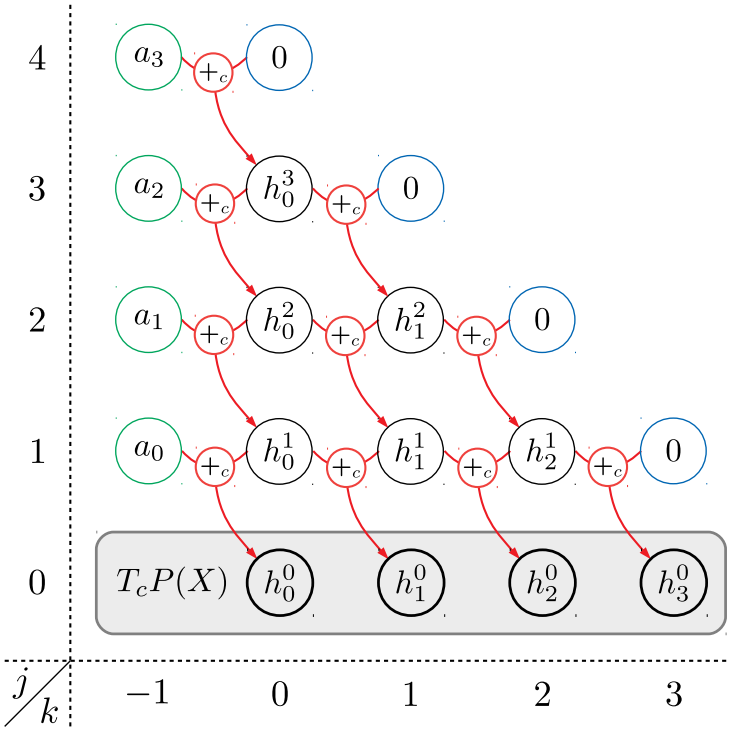


Figure 23: Illustration of the algorithm for computing the coefficients (h_k^0) of $T_cP(X)$

This algorithm uses $O(n^2)$ additions and multiplications and only $O(n^2)$ additions when $c = 1$. To compute the coefficients of $T_1RP(X)$, we apply the previous algorithm to the coefficients of $RP(X)$, i.e. to the reversed list of coefficients of $P(X)$. This can be done directly by substituting m in the inner loop of the algorithm by $m' = n - m$.

The previous method is necessary because the naïve method for computing the coefficients $(h_i)_{i \in [\![0,n]\!]}$ is too expensive. It consists in expanding each $a_i(X + c)^i$ term and then adding the resulting polynomials. From Newton's binomial expansion, we know that the coefficients of $a_i(X+c)^i$ are the $a_i\binom{i}{k}c^{i-k}$ for $0 \leq k \leq i$. If we use the multiplicative formula for binomial coefficients,

$\binom{i}{k} = \prod_{j=1}^{k} (i+1-j)/j$, computing all of the coefficients would require $O(n^3)$ multiplications, and adding the resulting polynomials would require $O(n^2)$ additions. The complexity remains the same if we set $c = 1$. The asymptotic cost of both methods is polynomial, but the number of operations required for the naïve method is much higher than for its counterpart, even for a sixth degree polynomial.

This method for the fast calculation of the coefficients of T_1RP can be used to obtain the Descartes' bound on the number of roots of P in $[0, 1]$ and implement the `DescartesBound` function mentioned in the pseudo-code of the VCA algorithm. This concludes the description of this real-root isolation algorithm, which is the one used in the current implementation of the algorithm.

6.1.2 Sturm's algorithm

Sturm's theorem provides a method of computing the exact number of distinct real roots of a univariate polynomial in an interval. Let us define what a Sturm sequence is before stating the theorem.

Definition 6.7 *Let $P \in \mathbb{R}[X]$ be a polynomial of degree n and P' its derivative. The Sturm sequence of P is the sequence of polynomials $(P_i)_{0 \leq i}$ constructed by:*

$$\begin{cases} P_0 = P \\ P_1 = P' \\ P_{i+1} = -\operatorname{rem}(P_{i-1}, P_i) \end{cases}$$

where $\operatorname{rem}(P_{i-1}, P_i)$ is the remainder of the euclidean division of P_{i-1} by P_i, such that $P_{i+1} = Q \cdot P_i - P_{i-1}$. By definition of euclidean division $\deg(P_{i+1}) < \deg(P_i)$, so the length of the Sturm sequence of P is at most $n + 1$.

We denote the number of sign changes in the Sturm sequence of P evaluated at $a \in \mathbb{R}$ by $V_P(a) = V(P_0(a), P_1(a), \ldots, P_n(a))$.

Sturm's theorem states the following:

Theorem 6.8 *For a square-free polynomial P, the number of distinct real roots of P in $]a, b]$ is equal to $V_P(a) - V_P(b)$.*

This theorem provides a method for computing the exact number of roots of a polynomial in any interval. It can be used as the basis for a bisection algorithm similar to the VCA algorithm. In our case, we compute the Sturm sequence corresponding to the chord polynomial and we apply the algorithm to the initial interval $[0, 1]$. The intervals are bisected until every subinterval returns contains either no roots or a single one.

The Sturm bisection algorithm is originally described in [8]. The best current bound for its complexity is $\widetilde{O}(n^4\tau^2)$ where n is the degree of the polynomial and τ is the bit size of its coefficients, the proof can be found in [11]. The most efficient implementations rely on fast polynomial evaluation and fast Sturm sequence computation methods.

6.2 Newton's method

Let us briefly recall that the goal of the algorithm up to this point was to reformulate the problem in order to use a 1D Newton's method instead of a 2D Newton-Raphson method as the previous

algorithms did. Our goal is to improve the stability of the algorithm and reduce the execution time by tackling the problem differently by explicitly using the coplanarity condition of Snell's law.

The real-root isolation algorithm provides a set of disjoint isolating intervals, each containing a single root of the chord polynomial. This provides an estimation of the position of the solutions to our problem on the chord. This estimate can be used to provide a good initial guess to Newton's method, hopefully accelerating its convergence. Once the roots over the chord have been found, their projection on the conic constitute the solutions to our problem. In this section we describe the one-dimensional Newton's method and how it interfaces with the rest of the algorithm.

Newton's method is an iterative root finding algorithm that produces increasingly precise approximations of the roots of a real-valued function. It is a popular algorithm due to its simplicity and fast convergence under reasonable assumptions.

6.2.1 Principle

This description of the one-dimensional Newton's method is based on [6]. This method takes a function f, its derivative f', an initial guess x_0, a tolerance ϵ and a maximum number of iterations N as inputs. It then constructs a sequence (x_n) of approximations of a root x of the function f. This sequence is defined recursively by

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Geometrically, x_{n+1} is the intersection of the x-axis and the tangent to the graph of f at x_n as illustrated in figure 24. The algorithm stops when the approximation is sufficiently accurate. This is usually checked with one of the following stopping conditions: $|x_n - x_{n-1}| < \epsilon, |x_n - x_{n-1}| / |x_n| < \epsilon$ or simply $\left|f(x_n)\right| < \epsilon$. For polynomials with a large degree, the last stopping condition is usually a poor choice, since the value of the polynomial tends to increase quickly in the neighbourhood of the root.

6.2.2 Convergence

It can be shown that if f is twice continuously differentiable over an open interval $]a, b[$, and $x \in]a, b[$ is such that $f(x) = 0$ and $f'(x) \neq 0$, there exists a $\delta > 0$ such that Newton's method converges at least quadratically for any initial guess $x_0 \in [x - \delta, x + \delta]$. This property stresses the importance of providing a good initial guess to the method.

6.2.3 Polynomial evaluation

Since we only apply this method to polynomials, we can use their properties to make the method more efficient. The construction of the sequence rely on the evaluation of the polynomial and its derivative. Polynomials can be evaluated efficiently using Horner's method with only $O(n)$ floating point operations, where n is the degree of the polynomial. Horner's method can be generalised to evaluate not only the polynomial, but also its derivatives. The algorithm 3 taken from [18] evaluates

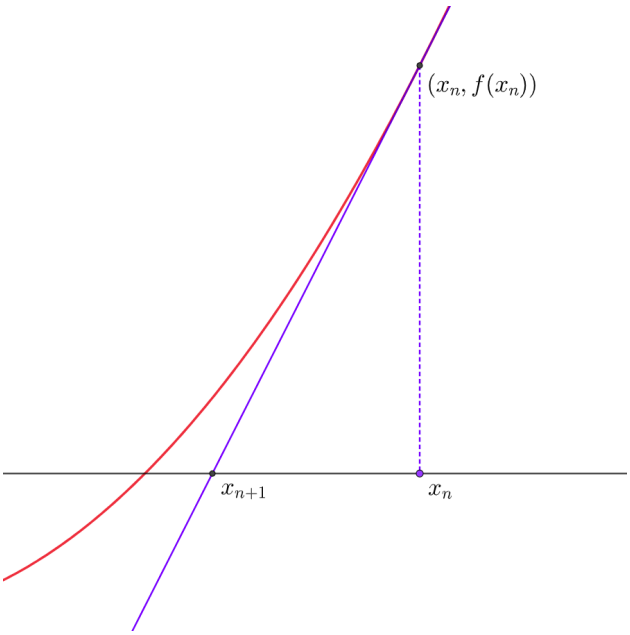


Figure 24: Geometric construction of Newton's approximation

the polynomial and its first derivative at the same time with $O(n)$ floating point operations.

Algorithm 3: Horner's method for evaluating a polynomial and its first derivative

input : A polynomial P represented as a list of coefficients (a_i)
$t \in \mathbb{R}$ the point where P and P' should be evaluated
output: evalP $= P(t)$ and evalP' $= P'(t)$ the values of P and P' at t
$n \leftarrow \deg P$;
evalP $\leftarrow a_n$;
evalP' $\leftarrow 0$;
for $i \leftarrow n$ **to** 0 **do**
  evalP' $\leftarrow t \cdot$ evalP' $+$ evalP;
  evalP $\leftarrow t \cdot$ evalP $+ a_i$;
end
return evalP, evalP';

6.2.4 Initial guess

For each root, the root-isolation algorithm returns an interval $[a, b]$ around the root. In our implementation, we choose to use the x-intercept of the straight line passing through the points $(a, f(a))$ and $(b, f(b))$. Linearising the function over the interval is a classical technique to obtain an initial estimate of the root for Newton's method.

6.3 Conic projection

The roots of the chord polynomials obtained with Newton's method lie on their corresponding chords, not on the conic itself. Theoretically, the solutions should all lie on the conic, since copla-

narity is a necessary condition for refraction. The conic subdivision algorithm guarantees that the distance between any chord and the conic lies below a user-specified threshold and thus provides a bound on the error. Unfortunately, since the coplanarity function is a sixth degree polynomial, small errors in the input are easily amplified in the output.

The simplest option is to directly use the roots on the chords, without projecting them onto the conic. Since the chord approximation can be regulated by the user, choosing a very accurate approximation guarantees that the roots of the chord polynomial are close to the roots of the constraint function over the conic. The advantage of the method is that it does not require any computation. This is the method used in the current implementation. Unfortunately, the impact of this parameter on the complexity of the algorithm is unclear. Relying on this method could potentially lead to heavily increased render times. For this reason, implementing a projection method to decrease the error without increasing the precision of the chord approximation is the ideal choice.

6.3.1 Closest projection

Let us first describe the problem. Given a root on a chord, we want to find the closest point to it on the conic arc subtended by that chord. This point is the orthogonal projection onto the conic. Let $\mathbf{Q} = (x_Q, y_Q)$ be the point on the chord and $\mathbf{P} = (x_P, y_P)$ its closest projection on the conic. Then $\overrightarrow{\mathbf{PQ}}$ must be parallel to the vector $\nabla Q(x_P, y_P)$ normal to the conic at P.

Given the implicit form of the conic and the vector normal to the conic

$$\begin{aligned} Q(x,y) &= Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0 \\ \nabla Q(x,y) &= (2Ax + By + D, Bx + 2Cy + E)^T, \end{aligned}$$

$\mathbf{P}$ is an orthogonal projection of $\mathbf{Q}$ if and only if

$$Q(x_P, y_P) = 0 \text{ and, for some } t \in \mathbb{R}, \begin{cases} x_Q - x_P = t \cdot (2Ax_P + By_P + D) \\ y_Q - y_P = t \cdot (Bx_P + 2Cy_P + E) \end{cases}.$$

By solving this linear system for x_P, y_P and substituting the solutions in the conic equation, we obtain a quartic equation in t. Solving this problem by radicals is costly and numerically unstable, and thus is not suitable for our algorithm. Figure 25 illustrates the case of a point inside an ellipse that has four orthogonal projections.

In [7], the authors review different methods for projecting points onto conics. Their goal is to find the minimal distance from a point to a conic in order to fit a conic section to data points. Even though their goal is unrelated to our purpose, the methods they consider could be used in our algorithm. In particular, they introduce a modified version of *Eberly's projection method*, originally presented in [13] and based in an idea from [17]. Eberly's method only applies to ellipses. It first transforms the ellipse into its centered, axis-aligned counterpart. Then it computes and solves the quartic polynomial with Newton's method and applies the inverse transformation to obtain the projected point for the original ellipse. In [7], the authors generalise Eberly's method to all types of conic and prove that the method is numerically well-behaved. Since our current algorithm already implements Newton's method and transformations on conic sections, this method would fit in perfectly.

The roots of the chord polynomials projected on the conic section are the points $\mathbf{P}$ such that $\mathbf{LPV}$ is a valid refracted path according to Snell's law of refraction. The last part of the algorithm computes the contribution of these paths to the radiance at $\mathbf{V}$.

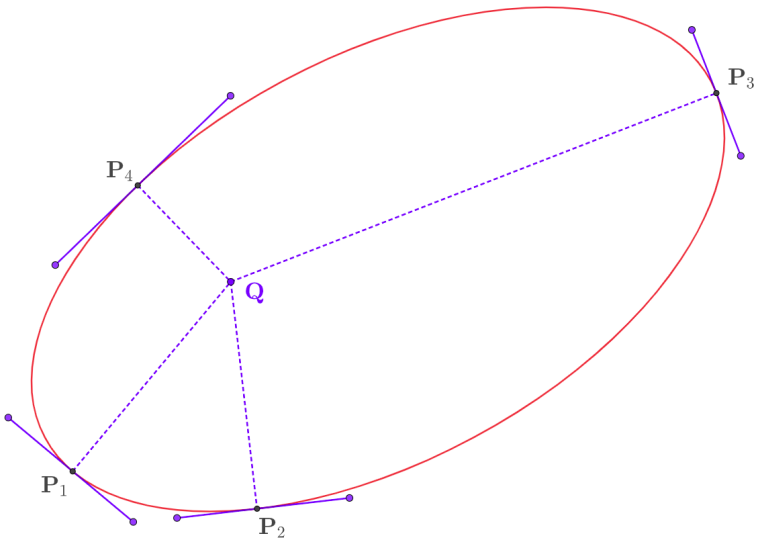


Figure 25: Orthogonal projection onto an ellipse

7 Contribution of refracted paths

Given a triangle and two points **L** and **V**, our algorithm computes all of the points **P** on the triangle such that **LPV** is a valid light path according to Snell's law of refraction. We first reformulated the problem by using the coplanarity condition to restrict the search space. Then, by approximating the coplanarity conic with chords, we successfully defined the constraint function over the new search space. Finally, we used a real-root isolation algorithm and Newton's method to find all of the solutions to our problem, which were then projected onto the search space. Every point **P** found by the algorithm corresponds to a different light path contributing to the radiance at the medium point **V**. The last part of the algorithm consists in computing the contribution of these light paths to the lighting of the scene.

7.1 Contribution

Let us define $d_V = ||\mathbf{V} - \mathbf{P}||, d_L = ||\mathbf{L} - \mathbf{P}||$, and $\hat{\omega}_L$, $\hat{\omega}_V$ the normalised directions from **P** to **L** and **V** respectively. Walter *et al.* [36] define the contribution of a refracted light path at **V** as:

$$\text{contribution} = \frac{I_e F A \rho}{D}$$

where I_e is the intensity of the point light source, F is the Fresnel transmittance at the specular interface, $A = T_r(\mathbf{V} \to \mathbf{P})$ is the beam transmittance between **V** and **P**, ρ is the value of the phase function at **V**, and D is the *distance correction factor* (DCF).

In the absence of participating media and refractive interface the DCF is just the square of the distance between **L** and **V**. If the shading normal $\mathbf{n}_s$ over the triangle is constant and equal to the normalised geometric normal $\hat{\mathbf{n}}_g$, then:

$$D = (d_V + \eta d_L)\left[\frac{\langle\hat{\omega}_L, \hat{\mathbf{n}}_g\rangle}{\langle\hat{\omega}_V, \hat{\mathbf{n}}_g\rangle} d_V + \eta \frac{\langle\hat{\omega}_L, \hat{\mathbf{n}}_g\rangle}{\langle\hat{\omega}_V, \hat{\mathbf{n}}_g\rangle} d_L\right].$$

In the general case, the DCF lacks a simple form. The article uses the ray differentials introduced by Igehy in [21] to compute the DCF. We denote by a' the derivative of a with respect to a small perturbation in $\hat{\omega}_V$ and perpendicular to it.

$$\begin{aligned}
\mathbf{P}' &= d_V \left[\hat{\omega}'_V - \frac{\langle \hat{\omega}'_V, \hat{\mathbf{n}}_g \rangle}{\langle \hat{\omega}_V, \hat{\mathbf{n}}_g \rangle} \hat{\omega}_V \right] \\
\mu &= \langle \hat{\omega}_L, \hat{\mathbf{n}}_s \rangle + \eta \langle \hat{\omega}_V, \hat{\mathbf{n}}_s \rangle \\
\mu' &= \left[\eta^2 \frac{\langle \hat{\omega}_V, \hat{\mathbf{n}}_s \rangle}{\langle \hat{\omega}_L, \hat{\mathbf{n}}_s \rangle} + \eta \right] \left(\langle \hat{\omega}'_V, \hat{\mathbf{n}}_s \rangle + \langle \hat{\omega}_V, \hat{\mathbf{n}}'_s \rangle \right) \\
\hat{\omega}'_L &= \eta \hat{\omega}'_V + \mu' \hat{\mathbf{n}}_s + \mu \hat{\mathbf{n}}'_s \\
\mathbf{L}' &= \mathbf{P}' - \langle \mathbf{P}', \hat{\omega}_L \rangle \hat{\omega}_L + d_L \hat{\omega}'_L
\end{aligned}$$

These derivatives are computed for two different directions perpendicular to $\hat{\omega}_V$ and to each other to obtain $\mathbf{L}'_\perp$ and $\mathbf{L}'_{||}$. The DCF is then $D = ||\mathbf{L}'_\perp \wedge \mathbf{L}'_{||}||$. The derivation for these formulas with the ray differentials is provided in appendix C. The original article only includes the previous set of equations.

7.2 Distance correction factor

The original article [36] does not explain the presence of the DCF in the formula for the contribution. It only mentions that it corresponds to the ratio of the differential area at $\mathbf{L}$ created by a differential solid angle around ω_V. This represents the action of the interface on a pencil of rays, i.e. whether it focuses or disperses light. The only other article that mentions this name is [20] which improves on the previous next-event estimation method and uses the same formula for the contribution.

The article on ray differentials [21] mentions the fact that they can be used to compute caustics in ray tracing, and refers the reader to an article on wavefront tracing [26] describing a technique similar to ray differentials. Its authors use similar information to compute how the energy carried by light wavefronts varies with propagation, reflection and refraction. This approach considers the geometry of a differential area of a light wavefront. Indeed, the shape of these wavefronts and their intensity vary with the interaction of light with the scene. Wavefront tracing implicitly accounts for the dispersion of light depending on its interaction with the scene.

In the manifold exploration article [23] on which the manifold next-event estimation technique relies, the authors explore the properties of the specular manifold defined in section 3.2. When computing the contribution of the paths in the specular manifold, the authors introduce a *generalised geometric factor* analogous to the one appearing when changing from the solid angle measure to the area measure in the lighting integrals. For a sequence of vertices $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2)$, where $\mathbf{x}_0, \mathbf{x}_2$ are diffuse and $\mathbf{x}_1$ is specular, the geometric factor for two vertices and the generalised geometric factor for the whole chain are:

$$G(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1) = \left| \frac{\mathrm{d}\omega^\perp(\mathbf{x}_0)}{\mathrm{d}A(\mathbf{x}_1)} \right|, \qquad G(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1 \leftrightarrow \mathbf{x}_2) = \left| \frac{\mathrm{d}\omega^\perp(\mathbf{x}_0)}{\mathrm{d}A(\mathbf{x}_2)} \right|.$$

This corresponds exactly to the interpretation of the DCF as the impact of a small perturbation to the direction of the path at one end of the chain on the differential area at the other end of the chain. The article introducing this notation does not provide a precise explanation for the presence

of this factor. One of the few articles that mentions the generalised geometric factor is [25]. In this paper, this factor arises when changing integration measures in integrals. More specifically, it arises in the definition of integration over a different path space with a new measure. This provides an intuition for the origin of the DCF.

The BSDF describing perfect specular scattering involves Dirac distributions depending on the incident and outgoing directions. The Dirac distribution can be integrated away when integrating with respect to solid angle. The factors resulting from changing from area measure to solid angle measure simplify neatly and result in the generalised geometric factor. The derivation is technical and requires a background in light transport theory. The details are provided in appendix D.

This concludes the theoretical contribution of this internship. By separately handling the coplanarity condition in Snell's law, we have successfully reformulated the refraction constraint over a one-dimensional search space. The coplanarity condition defines a conic section over the triangle plane. Thanks to well-known geometrical properties of conic sections, this allows us to only look for solutions on the conic arcs inside of the triangle. By defining the refraction constraint over straight lines, and approximating the relevant conic arcs by their chords, we use numerical methods to find the solutions to the problem on these arcs. This results in valid refracted light paths whose contribution to the lighting can be computed in the framework of light transport theory.

8 Implementation

This section explains in detail the current state of the implementation of the algorithm and the assumptions it relies on. It then discusses what the next course of action is to obtain a robust implementation.

8.1 Current state

The current version of the algorithm is implemented in the 0.6 version of the Mitsuba renderer [22]. The algorithm is implemented as a single-file `subsurface` plugin and interfaces with the renderer as specified in the documentation. The implementation is based on the code of the next-event estimation algorithm in [20], which is included in the renderer in the file `singlescatter.cpp`. It is intended to work with Mitsuba's `path` integrator. The code interfacing with the rest of the path tracer has been left as is, and as many fragments of the original code have been recycled to fit in the new algorithm. The implementation uses the linear algebra functions and structures provided by Mitsuba and the Eigen library [14] as much as possible, only implementing new methods when strictly necessary.

The implementation heavily relies on the following data structures:

- `ConicSection` stores the coefficients of the conic, its type, and the affine transformation for the parametrisation and its inverse;
- `ConicPoint` stores the position of the point in barycentric space and the corresponding conic parameter;
- `ConicParam` stores the value of the parameter and the branch for the hyperbolic case;
- `ConicChord` stores the `ConicPoint`s defining it, the point for the next chord subdivision, the distance to the conic and the chord polynomial;
- `Polynomial` stores as a vector of coefficients.

The current version includes:

- the entire implementation of the *dimensionality reduction* section;
- a partial implementation of the *finding refracted paths* section:
  - an implementation of the methods computing both Descartes and Sturm's bounds
  - an implementation of Newton's method
- the entire implementation of the *contribution of specular paths* section;
- the interface between the next-event estimation algorithm and the rest of the path tracer.

The implementation of the real-root isolation methods is currently incomplete. The bisection stage of the algorithm is not implemented yet. The algorithm computing the closest projection of a point on a conic is not implemented either. Once the implementation is finished, we will be able to compare our algorithm to the previous methods.

8.2 Assumptions and improvements

The implementation of the algorithm relies on a number of assumptions that have been pointed out throughout the report. Here is a summary of the cases that the current version of the algorithm does not handle:

- the coplanarity conic is a parabola;
- the coplanarity conic is degenerate;
- the chord polynomial is not square-free;
- the coplanarity conic is tangent to any side of the triangle.

The first three cases are simple to handle. If the conic is a parabola, we only need to specifically compute an affine transformation that maps the parabola to its unit counterpart. The parametrisation would also need to be adapted to work with the rest of the algorithm. If the conic is degenerate, then it is either a single point or a set of straight lines. Since we can define the constraint function over straight lines, degenerate conics do not rely on chord approximation. If the chord polynomial is not square-free, Yun's algorithm can be used to obtain a square-free decomposition of the polynomial. The last case is more of a problem. The non-tangency assumption allows us to construct a complete classification of the types of triangle-conic intersections. The algorithm that generates the pairs of intersection points that delimit the conic arcs inside of the triangle fully depends on this classification. Accepting tangent intersection points would considerably enlarge the classification, and the algorithm would have to be adapted to include all new cases. It would be interesting to gather information on the frequency of these supposedly rare cases when rendering multiple scenes. If these cases happen more often than we expected, they should be dealt with properly.

Floating-point accuracy is also a problem in this algorithm, to which real-root isolation algorithms are particularly sensitive. The coefficients of the chord polynomials span multiple orders of magnitude. The real-root isolation algorithm operates on these coefficients, which can cause accuracy problems. The same can be said for Newton's method. The current implementation does not make any particular effort to minimise floating-point error. If these errors become a problem in the final implementation, some parts of the code will have to be rewritten with that concern in mind.

Finally, the whole algorithm can be adapted to the reflection case, since the theoretical framework is almost identical. The main difference is that for refraction, every surface in the scene must be tested, whereas for refraction we only need to test the interface separating our two points $\mathbf{L}$ and $\mathbf{V}$. This requires more sophisticated culling tests to be able to discard most of the triangles in the scene that cannot contain a solution. Without these tests, the method would not scale well to scenes containing a large number of triangles.

9 Conclusion

In this internship report, we have explained the importance of next-event estimation techniques in modern path tracing by providing the theoretical framework of light transport theory. Regular NEE techniques fail when there is no direct line of sight to the light sources. We have presented two state-of-the-art algorithms that extend NEE to work through refractive interfaces and with reflective surfaces defined as triangle meshes with interpolated normals. These algorithms encode Snell's law as a constraint function over each triangle and find its roots with two-dimensional numerical methods, often struggling to converge for complicated scenes. Our contribution is an algorithm that uses the coplanarity condition in Snell's law separately to reduce the dimensionality of the problem. This condition can be represented as a conic section over every triangle. Since coplanarity is a necessary condition for refraction, all solutions to the problem lie on the conic section. By then defining the constraint function over this conic section, the problem can be solved with a one-dimensional Newton's method. Unfortunately, the constraint function cannot be directly defined over the conic section. Instead, we define it as a function of a point on a straight line, and we closely approximate the conic section by its chords. This allows us to define the constraint function over the approximation of the coplanarity conic. We then compute the roots of the constraint function over the chords and we project them onto the conic, thus obtaining the light paths refracted through the triangle connecting to a light source. Finally, the contribution of these paths is computed and taken into account by the path tracing algorithm. The contribution also includes a partial implementation of the algorithm in the Mitsuba renderer.

Future work Despite there being room for improvement, next-event estimation techniques are well known and well optimised for typical cases. Making NEE more versatile by extending it to notoriously hard to handle situations, such as paths involving specular vertices, is indubitably useful. It can considerably improve the convergence of path tracing in difficult lighting situations. This requires adapting the method to every new case we want to handle, which can be quite labour-intensive.

Recently, *path guiding* [27, 34] has gained in popularity. Path guiding is a family of importance sampling techniques that guide light paths towards interesting regions of the scene. In difficult lighting situations, sampling paths that contribute significantly to the final image is hard. By learning from previous sampling choices, path guiding allows for an increasingly efficient sampling of path space. This technique considerably improves the convergence of the algorithm in scenes where finding paths reaching a light source is complicated.

NEE is a method that deterministically connects a sample to a light source, even through refractive interfaces or after a reflection on a specular surface. By feeding these light paths to the path guiding algorithm, we could direct it towards interesting regions of the scene. This combination of NEE and path guiding could improve rendering times even further than each one of the methods independently.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Nicolas Holzschuch for his guidance and availability, and for all of the engaging discussions about physically-based rendering and computer graphics we had during my internship. Besides my supervisor, I would like to thank the rest of the

Maverick team, who made me feel welcome since the very first day of my internship. I also wish to thank Clément Legrand for his detailed answers to my many mathematical inquiries, his endless list of amazing textbook recommendations and his willingess to watch conic sections do the charleston. Last but not least, I would like to thank Juliette Veuillez, whose emotional support helped me get through these strange times.

A Conic section parametrisation

The implicit equation $Q(u, v) = 0$ defining the conic section is a poor formulation for choosing points on the conic. Since the conic section is a one-dimensional curve, we can indicate the position of a point on the conic with a single parameter t.

In the case where the conic is a unit circle or a unit hyperbola, a very simple parametrisation can be used. For a circle, the points are defined by $(\cos t, \sin t)$; for a hyperbola, the points are $(\pm \cosh t, \sinh t)$. The coplanarity conic defined by Q can be transformed into its unit counterpart by applying a translation, a rotation and anisotropic scaling, i.e. an affine transformation. To explicitly compute these transformations, we need to determine the center $\mathbf{x}_c = (x_c, y_c)$ of the conic, the angle θ between the major axis and the x-axis, and the semi-major axis a and semi-minor axis b of the conic.

These computations are based on [5] and [30]. We will consider ellipses and hyperbolas defined by the following equation and matrices:

$$Q(x, y) = Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

$$A_Q = \begin{bmatrix} A & B/2 & D/2 \\ B/2 & C & E/2 \\ D/2 & E/2 & F \end{bmatrix}; \qquad A_{33} = \begin{bmatrix} A & B/2 \\ B/2 & C \end{bmatrix}.$$

A.1 Centre

Central conics have a centre by definition, which is defined as the midpoint of any diameter of the conic. Let us denote it by (x_c, y_c). The diameter parallel to the x-axis has the equation $y = y_c$. Substituting in the conic equation gives:

$$Ax^2 + (By_c + D)x + Cy_c^2 + Ey_c + F = 0.$$

If the roots of this equation are x_1 and x_2 then their midpoint x_c is:

$$x_c = \frac{1}{2}(x_1 + x_2) = -\frac{1}{2}\frac{By_c + D}{A}.$$

Similarly, substituting with $x = x_c$ gives:

$$y_c = \frac{1}{2}(y_1 + y_2) = -\frac{1}{2}\frac{By_c + E}{C}.$$

By rearranging the previous equations, we obtain the centre as the solution of the following system of equations:

$$\begin{cases} Ax_c + \frac{B}{2}y_c + \frac{D}{2} = 0 \\ \frac{B}{2}x_c + Cy_c + \frac{E}{2} = 0 \end{cases}.$$

The explicit solution can be found with Cramer's rule:

$$x_c = \frac{\begin{vmatrix} B/2 & D/2 \\ C & E/2 \end{vmatrix}}{|A_{33}|} = \frac{C_{31}(A_Q)}{|A_{33}|}; \qquad y_c = \frac{\begin{vmatrix} A & -D/2 \\ B/2 & -E/2 \end{vmatrix}}{|A_{33}|} = \frac{C_{32}(A_Q)}{|A_{33}|}.$$

Note that $|A_Q| = \frac{D}{2}C_{31}(A_Q) + \frac{E}{2}C_{32}(A_Q) + F|A_{33}|$, i.e. $\frac{D}{2}x_c + \frac{E}{2}y_c + F = \frac{|A_Q|}{|A_{33}|}$.

A.2 Angle

Let us change variables in the equation of the conic so that the centre corresponds to the origin of the plane. The change of variable is the following: $\bar{x} = x - x_c$ and $\bar{y} = y - y_c$.

$$\begin{aligned}
& A(\bar{x}+x_c)^2 + B(\bar{x}+x_c)(\bar{y}+y_c) + C(\bar{y}+y_c)^2 + D(\bar{x}+x_c) + E(\bar{y}+y_c) + F = 0 \\
\Leftrightarrow\quad & A\bar{x}^2 + B\bar{x}\bar{y} + C\bar{y}^2 + (2Ax_c + By_c + D)\bar{x} + (Bx_c + 2Cy_c + E)\bar{y} + \\
& Ax_c^2 + Bx_cy_c + Cy_c^2 + Dx_c + Ey_c + F = 0 \\
\Leftrightarrow\quad & A\bar{x}^2 + B\bar{x}\bar{y} + C\bar{y}^2 + (Ax_c + \frac{B}{2}y_c + \frac{D}{2})x_c + (\frac{B}{2}x_c + Cy_c + \frac{E}{2})y_c + \frac{D}{2}x_c + \frac{E}{2}y_c + F = 0 \\
\Leftrightarrow\quad & A\bar{x}^2 + B\bar{x}\bar{y} + C\bar{y}^2 + \frac{|A_Q|}{|A_{33}|} = 0
\end{aligned}$$

In polar coordinates, this becomes $r^2(A\cos^2(\theta) + B\cos(\theta)\sin(\theta) + C\sin^2(\theta)) + \frac{|A_Q|}{|A_{33}|} = 0$. r is maximal (resp. minimal) when the multiplicative term is minimal (resp. maximal). This term is extremal whenever its derivative is null:

$$\begin{aligned}
& -2A\cos(\theta)\sin(\theta) + B(\cos^2(\theta) - \sin^2(\theta)) + 2C\cos(\theta)\sin(\theta) = 0 \\
\Leftrightarrow\quad & B(\cos^2(\theta) - \sin^2(\theta)) + 2(C-A)\cos(\theta)\sin(\theta) = 0 \\
\Leftrightarrow\quad & B\frac{1-\tan^2(\theta)}{1+\tan^2(\theta)} + 2(C-A)\frac{\tan(\theta)}{1+\tan^2(\theta)} = 0 \\
\Leftrightarrow\quad & B\tan^2(\theta) + 2(A-C)\tan(\theta) - B = 0 \\
\Leftrightarrow\quad & Bm^2 + 2(A-C)m - B \text{ with } m = \tan(\theta) \\
\Leftrightarrow\quad & B[2\frac{\lambda - A}{B}]^2 + 2(A-C)[2\frac{\lambda - A}{B}] - B = 0 \text{ with } m = 2\frac{\lambda - A}{B} \\
\Leftrightarrow\quad & 4(\lambda - A)^2 + 4(A-C)(\lambda - A) - B^2 = 0 \\
\Leftrightarrow\quad & (\lambda - A)(\lambda - C) - (\frac{B}{2})^2 = \lambda^2 - (A+C)\lambda + |A_{33}| = 0
\end{aligned}$$

The solutions for λ are the eigenvalues of A_{33}. The canonical equation of the central conic is then $\lambda_1 x^2 + \lambda_2 y^2 + |A_Q|/\lambda_1\lambda_2 = 0$, since $\lambda_1\lambda_2 = |A_{33}|$. If the conic is an ellipse, the eigenvalue with the smallest magnitude corresponds to the major axis. If the conic is a hyperbola, the eigenvalue with a sign opposite to $K = |A_Q|/|A_{33}|$ corresponds to the transverse axis. The semi-major axis a and the semi-minor axis b are given by $|K/\lambda_i|^{\frac{1}{2}}$ with the corresponding eigenvalue.

A.3 Affine transformation

A given central conic can be written in its unit form or its barycentric form as follows:

$$\begin{cases} \bar{x}^2 \pm \bar{y}^2 = 1 \\ Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0. \end{cases}$$

We can switch from one form to the other by multiplying the homogeneous coordinates of a point by multiplying them by an affine transform matrix or its inverse:

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = T_c \cdot R_\theta \cdot S_{a,b} \begin{pmatrix} \bar{x} \\ \bar{y} \\ 1 \end{pmatrix},$$

where

$$T_c = \begin{bmatrix} 1 & 0 & x_c \\ 0 & 1 & y_c \\ 0 & 0 & 1 \end{bmatrix}; \qquad R_\theta = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}; \qquad S_{a,b} = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

In the case of an ellipse, $t = \operatorname{atan2}(\bar{y}, \bar{x})$; in the case of a hyperbola, $t = \operatorname{asinh} \bar{y}$ and the sign of $\bar{x}$ indicates the branch the point lies on. We could have used a different parametrisation for the hyperbola, where a single value of the parameter corresponds to exactly a single point on the conic. Instead, we use this parametrisation for its similarity to the ellipse's, which allows us to consider very similar approaches for both cases in our algorithm.

B Triangle-hyperbola intersection

This is a sketch of the proof of why a hyperbola cannot intersect a triangle four times with one branch and twice with the other. This is not a full proof, but it conveys the main idea behind it.

We will consider a square hyperbola and a triangle of any shape. Since any hyperbola can be transformed into the square hyperbola with an affine transformation as explained in appendix A, this incurs no loss of generality. Given that one branch intersects four times the triangle, there is at least one of the sides of the triangle intersecting the branch twice.

Let us take this side of the triangle and place $\mathbf{P}_1$ and $\mathbf{P}_2$ on it. By moving the point $\mathbf{P}_0$ over the triangle plane, we can count the number of intersections of the triangle with the hyperbola depending on the region of the plane $\mathbf{P}_0$ lies in. These regions are illustrated in figure 26. Each region contains a label of the form $X+, Y-$, where X represents the number of intersections between the triangle and the positive branch of the hyperbola, and Y the number of intersections with its negative branch. No one of these regions are of the type $4+, 2-$, and these regions depend continuously on the position of the initial side of the triangle, $\mathbf{P}_1$ and $\mathbf{P}_2$. No specific configuration of these variables can make a new region appear. As such, the $4+, 2-$ case is impossible.

C Ray differentials

Let us compute the ray differentials for transfer (light propagation), refraction for our scene. To compute D, we apply a perturbation to $\hat{\omega}_V$ along along two unit vectors $\hat{u}_x$ and $\hat{u}_y$ such that

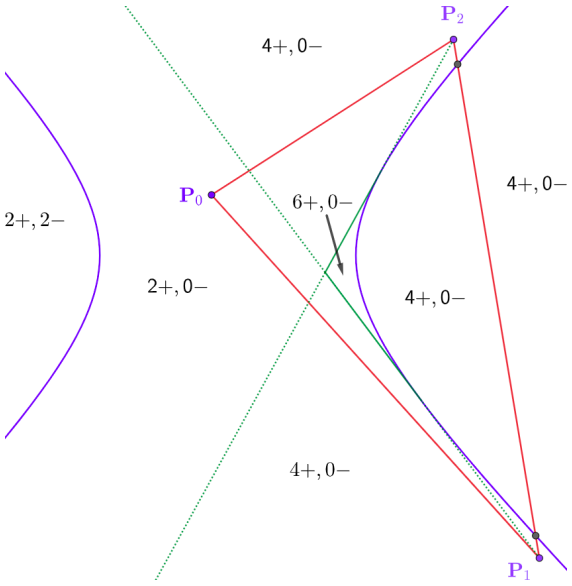


Figure 26: Illustration for the proof

$(u_x, u_y, \hat{\omega}_V)$ is a direct orthogonal basis: $\hat{u}_y = \hat{\omega}_V \wedge \hat{\mathbf{n}}_g$ and $\hat{u}_x = \hat{u}_y \wedge \hat{\omega}_V$. We then define the offset vector $\omega = \hat{\omega}_V - x\hat{u}_x - y\hat{u}_y$ and its normalized counterpart $\hat{\omega}$. We denote by d_V and d_L the distances between points $\mathbf{V}$ and $\mathbf{P}$, and points $\mathbf{L}$ and $\mathbf{P}$ respectively.

Since we work with differential offsets, we will consider that the derivatives of ω and its normalised counterpart $\hat{\omega}$ are the same. We only show the derivative with respect to x, but the same calculations apply to the derivatives with respect to y. For an explanation of the ray differential formulas for transfer and refraction, please refer to [21].

Initial state The starting point is $\mathbf{V}$ and the starting direction is $-\omega$. Their corresponding derivatives are:

$$\begin{aligned}\frac{\partial \mathbf{V}}{\partial x} &= 0\\ \frac{\partial \omega}{\partial x} &= -\hat{u}_x\\ \frac{\partial \hat{\omega}}{\partial x} &= \frac{\partial \omega}{\partial x}\,.\end{aligned}$$

Transfer from V to P First, light propagates from $\mathbf{V}$ towards the interface along the direction of $\hat{\omega}$, intersecting it at a point $\mathbf{P}$.

$$\begin{aligned}\mathbf{P} &= \mathbf{V} - d_V\hat{\omega}\\ \frac{\partial \mathbf{P}}{\partial x} &= d_V\left[\frac{\langle\frac{\partial \hat{\omega}}{\partial x}, \hat{\mathbf{n}}_g\rangle}{\langle\hat{\omega}, \hat{\mathbf{n}}_g\rangle} - \frac{\partial \hat{\omega}}{\partial x}\right]\hat{\omega} = d_V\left[\hat{u}_x - \frac{\langle\hat{u}_x, \hat{\mathbf{n}}_g\rangle}{\langle\hat{\omega}, \hat{\mathbf{n}}_g\rangle}\right]\hat{\omega}\end{aligned}$$

Refraction at P

$$\begin{aligned}
\hat{\omega}_L &= -\eta\hat{\omega} + \mu\hat{\mathbf{n}}_s \\
\mu &= \langle\hat{\omega}_L, \hat{\mathbf{n}}_s\rangle + \eta\langle\hat{\omega}, \hat{\mathbf{n}}_s\rangle \\
\frac{\partial\hat{\omega}_L}{\partial x} &= \eta\hat{u}_x + \frac{\partial\mu}{\partial x}\,\hat{\mathbf{n}}_s + \mu\,\frac{\partial\hat{\mathbf{n}}_s}{\partial x} \\
\frac{\partial\mu}{\partial x} &= \frac{\eta\mu}{\langle\hat{\omega}_L, \hat{\mathbf{n}}_s\rangle}\left[\langle -\hat{u}_x, \hat{\mathbf{n}}_s\rangle + \langle\hat{\omega}, \frac{\partial\hat{\mathbf{n}}_s}{\partial x}\rangle\right]
\end{aligned}$$

Shading normal at P To compute the derivatives of the shading normal at P, we first need to compute the derivatives of the barycentric coordinates. This method was inspired by [15].

$$\begin{aligned}
\mathbf{P} &= \mathbf{P}_0 + u\mathbf{p}_{10} + v\mathbf{p}_{20} \\
\frac{\partial\mathbf{P}}{\partial x} &= \frac{\partial u}{\partial x}\,\mathbf{p}_{10} + \frac{\partial v}{\partial x}\,\mathbf{p}_{20}
\end{aligned}$$

$$\begin{cases}
\langle\frac{\partial\mathbf{P}}{\partial x}\,, \mathbf{p}_{10}\rangle = \frac{\partial u}{\partial x}\|\mathbf{p}_{10}\|^2 + \frac{\partial v}{\partial x}\,\langle\mathbf{p}_{10}, \mathbf{p}_{20}\rangle \\
\langle\frac{\partial\mathbf{P}}{\partial x}\,, \mathbf{p}_{20}\rangle = \frac{\partial u}{\partial x}\,\langle\mathbf{p}_{10}, \mathbf{p}_{20}\rangle + \frac{\partial v}{\partial x}\|\mathbf{p}_{20}\|^2
\end{cases}$$

$$\begin{cases}
\left[\|\mathbf{p}_{10}\|^2\|\mathbf{p}_{20}\|^2 - \langle\mathbf{p}_{10}, \mathbf{p}_{20}\rangle^2\right]\frac{\partial u}{\partial x} = \|\mathbf{p}_{20}\|^2\,\langle\frac{\partial\mathbf{P}}{\partial x}\,, \mathbf{p}_{10}\rangle - \langle\mathbf{p}_{10}, \mathbf{p}_{20}\rangle\langle\frac{\partial\mathbf{P}}{\partial x}\,, \mathbf{p}_{10}\rangle \\
\left[\|\mathbf{p}_{10}\|^2\|\mathbf{p}_{20}\|^2 - \langle\mathbf{p}_{10}, \mathbf{p}_{20}\rangle^2\right]\frac{\partial v}{\partial x} = -\langle\mathbf{p}_{10}, \mathbf{p}_{20}\rangle\langle\frac{\partial\mathbf{P}}{\partial x}\,, \mathbf{p}_{10}\rangle + \|\mathbf{p}_{10}\|^2\,\langle\frac{\partial\mathbf{P}}{\partial x}\,, \mathbf{p}_{10}\rangle
\end{cases}$$

Now we can compute the derivatives of $\hat{\mathbf{n}}_s$.

$$\begin{aligned}
\mathbf{n}_s &= w\hat{\mathbf{n}}_0 + u\hat{\mathbf{n}}_1 + v\hat{\mathbf{n}}_2 \\
\frac{\partial\mathbf{n}_s}{\partial x} &= \frac{\partial w}{\partial x}\,\hat{\mathbf{n}}_0 + \frac{\partial u}{\partial x}\,\hat{\mathbf{n}}_1 + \frac{\partial v}{\partial x}\,\hat{\mathbf{n}}_2 \\
w &= 1 - u - v \\
\frac{\partial w}{\partial x} &= -\frac{\partial u}{\partial x} - \frac{\partial v}{\partial x} \\
\frac{\partial\hat{\mathbf{n}}_s}{\partial x} &= \frac{1}{\|\mathbf{n}_s\|}\,\frac{\partial\mathbf{n}_s}{\partial x} - \langle\hat{\mathbf{n}}_s, \frac{1}{\|\mathbf{n}_s\|}\,\frac{\partial\mathbf{n}_s}{\partial x}\rangle\hat{\mathbf{n}}_s
\end{aligned}$$

Transfer from P to L The light now propagates from $\mathbf{P}$ to $\mathbf{L} = \mathbf{P} + d_L\hat{\omega}_L$ along the direction $\hat{\omega}_L$.

$$\mathbf{L}'_{||} = \frac{\partial\mathbf{L}}{\partial x} = \left[\frac{\partial\mathbf{P}}{\partial x} + d_L\,\frac{\partial\hat{\omega}_L}{\partial x}\right] - \langle\frac{\partial\mathbf{P}}{\partial x}\,, \hat{\omega}_L\rangle\hat{\omega}_L$$

D Origin of the distance correction factor

Here is the derivation for a diffuse-specular-diffuse path consisting of surface points and without any participating media. This is not intended as a fully rigorous, mathematically accurate derivation.

The goal is to provide an intuition for the origin of the DCF in the framework of light transport theory.

The theoretical concepts behind this derivation are explained in more detail in section 5.A of E. Veach's thesis [33] on *general Dirac distributions*. The reader can also refer to Veach's thesis for an explanation of the formula for the specular BSDF. We will use the $\mathrm{d}\sigma$ notation for solid angle measures instead of $\mathrm{d}\omega$ to avoid confusion between directions and integration measures. We will use the following properties of general Dirac distributions:

$$\int_\Omega f(x)\delta_\mu(y-x)\,\mathrm{d}\mu(x) = f(y),$$
$$\int_\Omega f(x)\,\mathrm{d}\mu(x) = \int_\Omega f(x)\,\frac{\mathrm{d}\mu}{\mathrm{d}\nu}\,(x)\,\mathrm{d}\nu(x).$$

Let x_2 be a surface point and x_0 a sample on a light source separated by a specular interface. We define $\mathcal{S}$ as the set of paths of length 3 between x_2 and x_0 passing through a specular vertex x_1 on the refractive interface. The direction between vertices x_i and x_{i+1} will be denoted by $\hat{\omega}_{i,i+1}$.

The contribution of these paths to the illumination at V can be computed as follows:

$$\int_{\mathcal{S}} L_e(x_0 \to x_1)G(x_0 \leftrightarrow x_1)\bar{f}_{s,\hat{\mathbf{n}}_s}(x_0 \to x_1 \to x_2)G(x_1 \leftrightarrow x_2)\,\mathrm{d}A(x_0)\,\mathrm{d}A(x_1)\,\mathrm{d}A(x_2).$$

where

$$G(x_i \leftrightarrow x_{i+1}) = \frac{\mathrm{d}\sigma^\perp_{x_{i+1},\hat{\mathbf{n}}_g}(\hat{\omega}_{i+1,i})}{\mathrm{d}A(x_i)},$$
$$\bar{f}_{s,\hat{\mathbf{n}}_s}(x_0 \to x_1 \to x_2) = R\cdot\delta_{\sigma^\perp_{x_1,\hat{\mathbf{n}}_s}}(\omega_i-\omega_{10})\frac{\langle\omega_i,\hat{\mathbf{n}}_s\rangle}{\langle\omega_i,\hat{\mathbf{n}}_g\rangle}$$
$$\mathrm{d}\sigma^\perp_{x,\hat{\mathbf{n}}}(\omega) = \langle\omega,\hat{\mathbf{n}}(x)\rangle\,\mathrm{d}\sigma(\omega)$$

By changing integration measures from $\mathrm{d}A(x_1)$ to $\mathrm{d}\sigma^\perp_{x_1,\hat{\mathbf{n}}_s}(\omega_i)$, we can integrate away the delta function in the BSDF. Unless specified otherwise, $\hat{\mathbf{n}}_s$ and $\hat{\mathbf{n}}_g$ are the normals at the specular vertex x_1.

$$\begin{aligned}
&\int_{\mathcal{S}} L_e(x_0 \to x_1)G(x_0 \leftrightarrow x_1)\bar{f}_{s,\hat{\mathbf{n}}_s}(x_0 \to x_1 \to x_2)G(x_1 \leftrightarrow x_2)\,\mathrm{d}A(x_0)\,\mathrm{d}A(x_1)\,\mathrm{d}A(x_2)\\
&= \iint_{A^2}\left[\int_\Omega K(\omega_i)\delta_{\sigma^\perp_{x_1,\hat{\mathbf{n}}_s}}(\omega_i-\omega_{10})\,\mathrm{d}\sigma^\perp_{x_1,\hat{\mathbf{n}}_s}(\omega_i)\right]\mathrm{d}A(x_0)\,\mathrm{d}A(x_2)\\
&\quad\text{where } K(\omega_i) = R\frac{\langle\omega_i,\hat{\mathbf{n}}_s\rangle}{\langle\omega_i,\hat{\mathbf{n}}_g\rangle}L_e(x_0\to x_1)\,\frac{\mathrm{d}\sigma^\perp_{x_1,\hat{\mathbf{n}}_g}(\omega_i)}{\mathrm{d}A(x_0)}\,\frac{\mathrm{d}\sigma^\perp_{x_2,\hat{\mathbf{n}}_g}(\omega_{21})}{\mathrm{d}A(x_1)}\,\frac{\mathrm{d}A(x_1)}{\mathrm{d}\sigma^\perp_{x_1,\hat{\mathbf{n}}_s}(\omega_i)}\\
&= \iint_{A^2} K(\omega_{10})\,\mathrm{d}A(x_0)\,\mathrm{d}A(x_2)\\
&= \iint_{A^2} R\frac{\langle\omega_{10},\hat{\mathbf{n}}_s\rangle}{\langle\omega_{10},\hat{\mathbf{n}}_g\rangle}L_e(x_0\to x_1)\,\frac{\mathrm{d}\sigma^\perp_{x_1,\hat{\mathbf{n}}_g}(\omega_{10})}{\mathrm{d}\sigma^\perp_{x_1,\hat{\mathbf{n}}_s}(\omega_{10})}\,\frac{\mathrm{d}\sigma^\perp_{x_2,\hat{\mathbf{n}}_g}(\omega_{21})}{\mathrm{d}A(x_0)}\,\mathrm{d}A(x_0)\,\mathrm{d}A(x_2)\\
&= \iint_{A^2} R\frac{\langle\omega_{10},\hat{\mathbf{n}}_s\rangle}{\langle\omega_{10},\hat{\mathbf{n}}_g\rangle}\frac{\langle\omega_{10},\hat{\mathbf{n}}_g\rangle}{\langle\omega_{10},\hat{\mathbf{n}}_s\rangle}L_e(x_0\to x_1)\langle\omega_{21},\hat{\mathbf{n}}_g(x_2)\rangle\,\frac{\mathrm{d}\sigma(\omega_{21})}{\mathrm{d}A(x_0)}\,\mathrm{d}A(x_0)\,\mathrm{d}A(x_2)\\
&= \iint_{A^2} RL_e(x_0\to x_1)\langle\omega_{21},\hat{\mathbf{n}}_g(x_2)\rangle\,\frac{\mathrm{d}\sigma(\omega_{21})}{\mathrm{d}A(x_0)}\,\mathrm{d}A(x_0)\,\mathrm{d}A(x_2)
\end{aligned}$$

The term $\frac{\mathrm{d}\sigma(\omega_{21})}{\mathrm{d}A(x_0)}$ corresponds to the inverse of the distance correction factor and can be computed using ray differentials as shown in C. This approach can be generalised to longer specular chains by changing measures accordingly to integrate away the delta functions introduced by the specular BSDFs. This can probably be generalised to include participating media and deal with the case where x_2 is a volume scattering event. When dealing with area lights, the probability of choosing a light sample must also be taken into account when computing the contribution of the refracted light paths.

References

[1] N.H. Abel. *Mémoire sur les équations algébriques où on démontre l'impossibilité de la résolution de l'équation générale du cinquième degré*. Christiania - Groendahl, 1824.

[2] Alkiviadis Akritas. "Vincent's theorem of 1836: Overview and future research". In: *Journal of Mathematical Sciences* 168 (July 2010), pp. 309–325. DOI: 10.1007/s10958-010-9982-1.

[3] Alberto Claudio Alesina and Massimo Galuzzi. "Vincent's theorem from a modern point of view". In: *Rendiconti del Circolo Matematico di Palermo* Serie II. Suppl 64 (Jan. 2000), pp. 179-191.

[4] Arthur Appel. "Some Techniques for Shading Machine Renderings of Solids". In: *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*. AFIPS '68 (Spring). Atlantic City, New Jersey: Association for Computing Machinery, 1968, pp. 37–45. ISBN: 9781450378970. DOI: 10.1145/1468075.1468082.

[5] Ayoub B. Ayoub. "The Central Conic Sections Revisited". In: *Mathematics Magazine* 66.5 (1993), pp. 322–325. ISSN: 0025570X, 19300980.

[6] R.L. Burden and J.D. Faires. *Numerical Analysis*. Cengage Learning, 2010. ISBN: 9780538733519.

[7] N. Chernov and S. Wijewickrema. "Algorithms for projecting points onto conics". In: *Journal of Computational and Applied Mathematics* 251 (2013), pp. 8–21. ISSN: 0377-0427. DOI: https://doi.org/10.1016/j.cam.2013.03.031.

[8] G. E. Collins and R. Loos. "Real Zeros of Polynomials". In: *Computer Algebra: Symbolic and Algebraic Computation*. Ed. by Bruno Buchberger, George Edwin Collins, and Rüdiger Loos. Vienna: Springer Vienna, 1982, pp. 83–94. ISBN: 978-3-7091-3406-1. DOI: 10.1007/978-3-7091-3406-1_7.

[9] Robert L. Cook, Thomas Porter, and Loren Carpenter. "Distributed Ray Tracing". In: *SIGGRAPH Comput. Graph.* 18.3 (Jan. 1984), pp. 137–145. ISSN: 0097-8930. DOI: 10.1145/964965.808590.

[10] René Descartes. *Discours de la Méthode – La Géométrie*. 1637.

[11] Zilin Du, Vikram Sharma, and Chee K. Yap. "Amortized Bound for Root Isolation via Sturm Sequences". In: *Symbolic-Numeric Computation*. Ed. by Dongming Wang and Lihong Zhi. Basel: Birkhäuser Basel, 2007, pp. 113–129. ISBN: 978-3-7643-7984-1.

[12] Philip Dutre, Kavita Bala, Philippe Bekaert, and Peter Shirley. *Advanced Global Illumination*. AK Peters Ltd, 2006. ISBN: 1568813074.

[13] David H. Eberly. "Chapter 14 - Distance Methods". In: *3D Game Engine Design (Second Edition)*. Ed. by David H. Eberly. Second Edition. The Morgan Kaufmann Series in Interactive 3D Technology. San Francisco: Morgan Kaufmann, 2007, pp. 639–679. ISBN: 978-0-12-229063-3. DOI: https://doi.org/10.1016/B978-0-12-229063-3.50018-2.

[14] Gaël Guennebaud, Benoît Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010.

[15] Eric Haines and Tomas Akenine-Möller, eds. *Ray Tracing Gems*. http://raytracinggems.com. Apress, 2019, pp. 326–328.

[16] Johannes Hanika, Marc Droske, and Luca Fascione. "Manifold Next Event Estimation". In: *Comput. Graph. Forum* 34.4 (July 2015), pp. 87–97. ISSN: 0167-7055.

[17] John C. Hart. "II.1. - Distance to an Ellipsoid". In: *Graphics Gems*. Ed. by Paul S. Heckbert. Academic Press, 1994, pp. 113–119. ISBN: 978-0-12-336156-1. DOI: https://doi.org/10.1016/B978-0-12-336156-1.50019-7.

[18] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. 2nd. USA: Society for Industrial and Applied Mathematics, 2002. ISBN: 0898715210.

[19] Markus Hohenwarter. "GeoGebra: Ein Softwaresystem für dynamische Geometrie und Algebra der Ebene". (In German.) MA thesis. Paris Lodron University, Salzburg, Austria, Feb. 2002.

[20] N. Holzschuch. "Accurate Computation of Single Scattering in Participating Media with Refractive Boundaries". In: *Comput. Graph. Forum* 34.6 (Sept. 2015), pp. 48–59. ISSN: 0167-7055. DOI: 10.1111/cgf.12517.

[21] Homan Igehy. "Tracing Ray Differentials". In: *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '99. USA: ACM Press/Addison-Wesley Publishing Co., 1999, pp. 179–186. ISBN: 0201485605. DOI: 10.1145/311535.311555.

[22] Wenzel Jakob. *Mitsuba renderer*. http://www.mitsuba-renderer.org. 2010.

[23] Wenzel Jakob and Steve Marschner. "Manifold Exploration: A Markov Chain Monte Carlo Technique for Rendering Scenes with Difficult Specular Transport". In: *ACM Trans. Graph.* 31.4 (July 2012). ISSN: 0730-0301. DOI: 10.1145/2185520.2185554.

[24] James T. Kajiya. "The Rendering Equation". In: *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '86. New York, NY, USA: ACM, 1986, pp. 143–150. ISBN: 0-89791-196-2. DOI: 10.1145/15922.15902.

[25] Anton S. Kaplanyan, Johannes Hanika, and Carsten Dachsbacher. "The Natural-Constraint Representation of the Path Space for Efficient Light Transport Simulation". In: *ACM Trans. Graph.* 33.4 (July 2014). ISSN: 0730-0301. DOI: 10.1145/2601097.2601108.

[26] Don Mitchell and Pat Hanrahan. "Illumination from Curved Reflectors". In: *SIGGRAPH Comput. Graph.* 26.2 (July 1992), pp. 283–291. ISSN: 0097-8930. DOI: 10.1145/142920.134082.

[27] Thomas Müller, Markus Gross, and Jan Novák. "Practical Path Guiding for Efficient Light-Transport Simulation". In: *Computer Graphics Forum* 36.4 (June 2017), pp. 91–100. ISSN: 1467-8659. DOI: 10.1111/cgf.13227.

[28] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. 3rd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2016. ISBN: 9780128006450.

[29] Fabrice Rouillier and Paul Zimmermann. "Efficient isolation of polynomial's real roots". In: *Journal of Computational and Applied Mathematics* 162.1 (Jan. 2004). Article dans revue scientifique avec comité de lecture. internationale., pp. 33–50. DOI: 10.1016/j.cam.2003.08.015.

[30] J.W. Rutter. *Geometry of Curves*. Chapman Hall/CRC Mathematics Series. Taylor & Francis, 2000. ISBN: 9781584881667.

[31] Vikram Sharma. "Complexity Analysis of Algorithms in Algebraic Computation". PhD thesis. USA, 2007.

[32] Jacques Charles François Sturm. "Mémoire sur la résolution des équations numériques". In: *Bulletin des Sciences de Férussac* (1829).

[33] Eric Veach. "Robust Monte Carlo Methods for Light Transport Simulation". AAI9837162. PhD thesis. Stanford, CA, USA, 1998. ISBN: 0591907801.

[34] Jiří Vorba, Johannes Hanika, Sebastian Herholz, Thomas Müller, Jaroslav Křivánek, and Alexander Keller. "Path Guiding in Production". In: *ACM SIGGRAPH 2019 Courses*. SIGGRAPH '19. Los Angeles, California: ACM, 2019, 18:1–18:77. ISBN: 978-1-4503-6307-5. DOI: 10.1145/3305366.3328091.

[35] Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. "Microfacet Models for Refraction Through Rough Surfaces". In: *Proceedings of the 18th Eurographics Conference on Rendering Techniques*. EGSR'07. Grenoble, France: Eurographics Association, 2007, pp. 195–206. ISBN: 978-3-905673-52-4. DOI: 10.2312/EGWR/EGSR07/195-206.

[36] Bruce Walter, Shuang Zhao, Nicolas Holzschuch, and Kavita Bala. "Single Scattering in Refractive Media with Triangle Mesh Boundaries". In: *ACM Trans. Graph.* 28.3 (July 2009). ISSN: 0730-0301. DOI: 10.1145/1531326.1531398.

[37] Turner Whitted. "An Improved Illumination Model for Shaded Display". In: *Commun. ACM* 23.6 (June 1980), pp. 343–349. ISSN: 0001-0782. DOI: 10.1145/358876.358882.

[38] Wikipedia. *Conic Sections*. URL: https://en.wikipedia.org/wiki/Conic_section.

[39] Wikipedia. *Matrix Representation of Conic Sections*. URL: https://en.wikipedia.org/wiki/Matrix_representation_of_conic_sections.

[40] David Y.Y. Yun. "On Square-Free Decomposition Algorithms". In: *Proceedings of the Third ACM Symposium on Symbolic and Algebraic Computation*. SYMSAC '76. Yorktown Heights, New York, USA: Association for Computing Machinery, 1976, pp. 26–35. ISBN: 9781450377904. DOI: 10.1145/800205.806320.