An Angular Parameterization for Manifold Connections

Oscar Dadfar

CMU-CS-22-105

May 2022

Computer Science Department

School of Computer Science

Carnegie Mellon University

Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements

for the degree of Master of Science.

Keywords: rendering, path connections, optimization, reflections, refractions

Abstract

Forming light paths connecting two points in a scene, so called direct connections, is a key building block of modern Monte Carlo rendering algorithms. For example, path tracing forms direct connections between each point on a traced path and a light source; and bidirectional path tracing forms direct connections between all pairs of vertices on two traced paths. Forming direct connections through one or multiple specular or refractive surfaces is particularly challenging: Any such connection must satisfy the laws of specular reflection or refraction at each of its vertices; as a result, valid direct connections occupy a very low-dimensional manifold within the space of all possible light paths, motivating the name manifold direct connections.

Existing works on forming manifold direct connections treat this as an optimization problem: They use gradient descent to search for points on specular reflective or refractive surfaces that, when used to form a direct connection, the resulting path satisfies the laws of specular reflection or refraction, respectively. All of these works take advantage of the fact that these laws are efficiently differentiable, and thus lend themselves to search through gradient-based optimization.

We adopt the optimization approach for forming manifold direct connections, with an important difference: Instead of optimizing over surface points such as in previous works, we optimize over the initial direction of the direct path. This reparameterized optimization problem offers a number of advantages. First, it makes it possible to form direct connections through multiple specular reflective and refractive surfaces. Furthermore, it allows forming direct connections through different surface representations, including explicit (e.g., polygonal mesh), implicit (e.g., signed distance function, neural network), and point cloud representations. Finally, it provides a continuous parameterization of the search space (space of directions), which helps accelerate the convergence of gradient-based optimization.

Contents

1 Motivation

Raytracing is an expensive graphics process where we aim to simulate physically-accurate light interactions in a scene in order to render an environment. The term ray-tracing comes from the assumption that light is a particle and follows linear paths when bouncing off surfaces. Allowing light to bounce around our scene allows for realistic color blending effects, where when a light particle hits a red surface and then a yellow surface, a little of the red surface is now blended into the intersection point on the yellow surface. We refer to light as signal, where each bounce carries a new signal of color information blended in with previous bounces before hitting the camera.

Real-life lighting involves millions of light particles bouncing around illuminating the scene, yet most of these light rays do not make it into our camera. Therefore we trace rays from the camera to the light source to minimize wasted compute resources. If when raytracing, we terminate our bounces on a diffuse surface, we can aim one last ray towards the light source to see if we get any light signal that we can send back to the camera. Yet it is not always guaranteed that a camera ray can reach a light source due to occlusion from geometry around the light source.

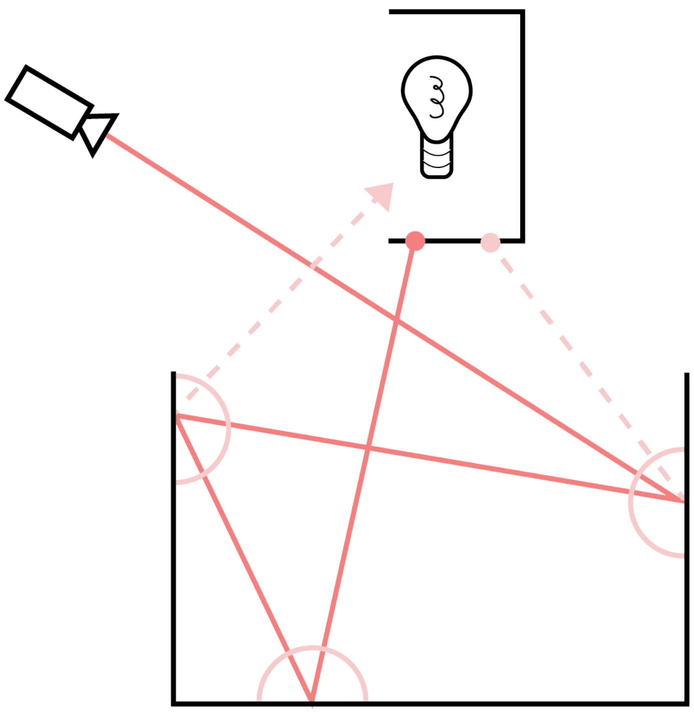


Figure 1: Raytracing with occlusion from surrounding geometry makes for a difficult process in propagating signal from the light source back to the camera. Only a select few of the rays ever reach the light.

In these cases like in Figure 1, we accumulate no light along our ray, giving us no light or signal for a given pixel and ultimately wasting a ray. Terminating on a delta-BRDF such as reflective or refractive materials complicate this even more as we are not able to pick the next path to be towards the light without violating the object's BRDF. It becomes even more increasingly difficult to render scenes with multiple mirror and glass objects, highlighting the need for an algorithm to help us locate paths from a diffuse origin to a light source through multiple refractive materials. The idea is if we terminate on a reflective or refractive material, we should be able to backtrack to the last diffuse surface interaction and choose a new ray direction that passes through the same or similar reflective and refractive materials that will reach the light source. We predict this will allow us to render interesting scenes full of specular objects with much fewer samples than regular path tracing.

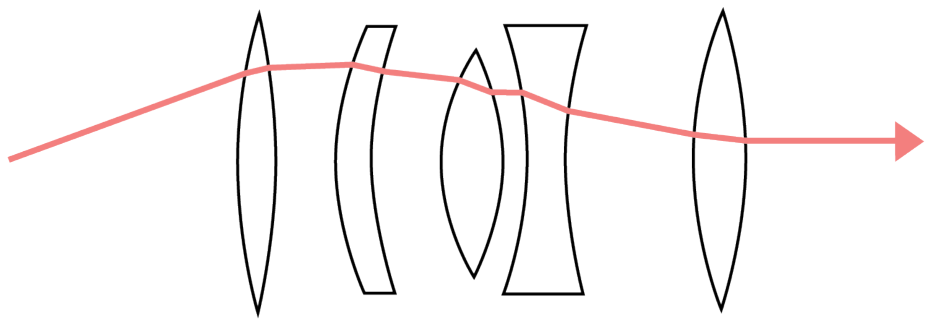


Figure 2: Light rays travelling through refractive lenses in multi-lens cameras.

Such an algorithm would be useful for rendering scenes with many delta-BRDFs, but also is useful for simulating paths into multi-lens cameras [20] like in Figure 2 and for vision-based tactile sensors, [17] where a ray needs to pass through multiple refractive lenses before hitting a point on a sensor. We could use such an approach to find a light path through the lenses that allow it to reach a given target on the camera sensor the same way we optimize a ray hitting a light source when going through multiple reflective and refractive materials. We can also use this approach for rendering Schlieren imaging [13] where we can converge paths through refractive materials towards light sources to increase signal in the images. All of these applications involve light transport through multiple specular reflective and refractive surfaces, and continuously-refractive media.

2 Background

2.1 Scene Definition

We will first formulate the raytracing problem. For the sake of simplicity, we will assume a scene file is comprised of geometry, materials, and lighting.

2.1.1 Geometry

The geometry of an object can be either implicit or explicit. Implicit geometries deal with mathematical functions that are evaluated to determine whether intersection with an object occurs. Some examples are signed distance fields [9] and constructive solid geometry [3] that are comprised of analytical functions. Points on the outside of SDFs evaluate to positive values while points on the inside of SDFs evaluate to negative values. We get an intersection when the SDF evaluates to 0. We can use ray-marching [4], where we iteratively check the SDF and push forward until we reach a field value of 0 as in Figure 3.

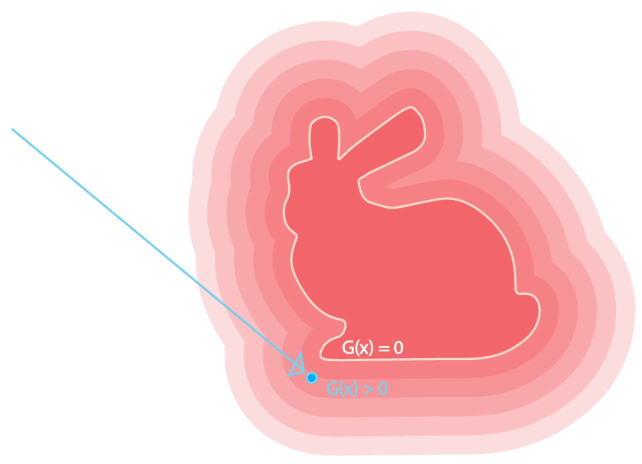


Figure 3: SDF of a bunny. We intersect the implicit geometry via ray-marching by evaluating the SDF and taking a step forward until we reach the boundary.

Explicit geometry is comprised of discrete values that do not need to be analytically solved for. This includes point clouds and, most commonly, triangulated surfaces. We often represent geometry as triangles for their flexibility in capturing curves in surfaces and how quicky we can perform intersection tests on them. This highlights the tradeoff between implicit and explicit geometry: implicit has a lower memory bandwith while requiring more computations for determining intersection, while explicit geometry is more memory-intensive for less computation.

2.1.2 Materials

There are thousands of possible materials we can use, though for the sake of simplicity we will focus on the three most popular materials: pure diffuse, reflective, and refractive. These materials are represented by their bidirectional distribution functions (BRDF) in Figure 4 that visualize the probability of an outgoing ray given the incoming ray in a material. For diffuse materials, the probability of an outgoing ray is uniform around all directions, making it independent of the incoming ray. If we consider the the hemisphere around the point of incident, then the probability of a ray is given by the inverse of the surface area of the hemisphere, or $\frac{1}{\pi}$ for short.

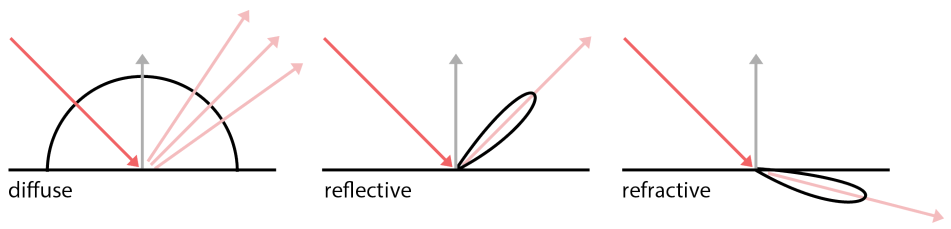


Figure 4: Common BRDFs.

Reflections and refractions are known as delta BRDFs because their distribution functions replicate the Dirac delta distribution. The BRDF is zero in all locations but one. For a reflective BRDF, the outgoing ray is a perfect reflection of the incoming ray about the surface normal of the incident surface. The equation for reflection is given as:

$$\begin{aligned} v_i &= v_{i-1}^{||} - v_{i-1}^{\perp} && (1) \\ &= (v_{i-1} - n_i cos\theta_{i-1}) - n_i cos\theta_{i-1} && (2) \\ &= v_{i-1} - 2n_i cos\theta_{i-1} && (3) \\ &= v_{i-1} - 2n_i(v_{i-1} \cdot n_i) && (4) \end{aligned}$$

where $v_{i-1}^{||}$ is the parallel component of the incoming ray to the surface, and $v_{i-1}^{\perp}$ is the perpendicular component in Figure 5, where we aim to flip the perpendicular component of the ray. n_i is the incident surface normal.

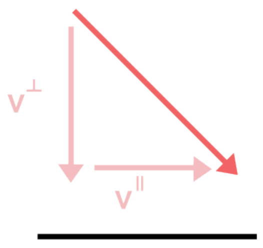


Figure 5: Ray components.

We represent refractions via Snell's law, where rays bend into the material [10]. When intersecting a refractive material, there is a special index of refraction term that tells us how much the ray bends towards or away from the surface normal. In dense materials, the index of refraction is higher, meaning the light bends more towards the normal. This index is a scientific ratio measuring how dense the material is relative to air, so naturally air will have an index of refraction of 1.

$$v_i = \frac{\eta_1}{\eta_2} v_{i-1}^{||} - n_i \sqrt{1 - ||v_i^{||}||^2} \tag{5}$$
$$= \frac{\eta_1}{\eta_2}(v_{i-1} - cos\theta_{i-1} n_i) - n_i \sqrt{1 - sin\theta_i^2} \tag{6}$$
$$= \frac{\eta_1}{\eta_2}(v_{i-1} - cos\theta_{i-1} n_i) - n_i \sqrt{1 - (\frac{\eta_1}{\eta_2})^2 (1 - cos\theta_{i-1}^2)} \tag{7}$$
$$= \frac{\eta_1}{\eta_2}(v_{i-1} - (v_{i-1} \cdot n_i) n_i) - n_i \sqrt{1 - (\frac{\eta_1}{\eta_2})^2 (1 - (v_{i-1} \cdot n_i)^2)} \tag{8}$$

In the above equation η_1 is the index of refraction of the medium that the ray currently is in, and η_2 is the index of refraction that the ray is entering. We take the ratio to measure the change of difference in the index of refraction. The parallel component of the ray bends by the index of refraction ratio given by the term $\frac{\eta_1}{\eta_2} v_{i-1}^{||}$, whereas the perpendicular component is computed by Snell's law as $-n_i \sqrt{1 - ||v_i^{||}||^2}$.

In the event that $1 - ||v_i^{||}||^2 < 0$, then we cannot compute the square root term and fail to compute the refractive outgoing ray. If we expand the terms out, we can rewrite the term under the square root as $1 - (\frac{\eta_1}{\eta_2})^2 (1 - (v_{i-1} \cdot n_i)^2)$. Both v_{i-1} and n_i are normalized, so $(v_{i-1_i} \cdot n)^2 \leq 1$, and thus $1 \geq (1 - (v_{i-1} \cdot n_i)^2) \geq 0$. So in order for $1 - (\frac{\eta_1}{\eta_2})^2 (1 - (v_{i-1} \cdot n_i)^2) < 0$, $\frac{\eta_1}{\eta_2}$ must be greater than 1. This occurs when the incoming index of refraction η_1 is much larger than the outgoing index of refraction η_2, or in other words, we are leaving a more dense for a less dense material. When this happens, the ray reflects instead of refracts, gaining the name 'total internal reflection' since reflection is the only thing we can do. This is common in water effects when the light hits the surface, but the change in density is large enough that the ray reflects back into the water.

2.1.3 Lighting

There are several possible lights available to us: point, area, directional, and environment lighting. For simplicity, we will focus on supporting point and area lights in Figure 6 for they have discrete physical locations in space with intersectable geometry, whereas directional and environment lighting are represented at infinity.

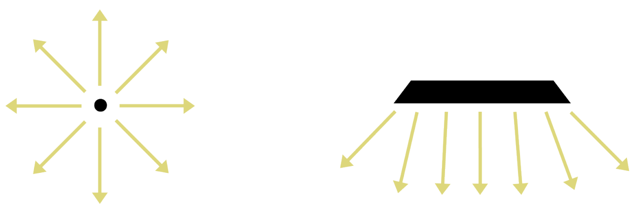


Figure 6: Point light and area light.

2.2 Forward Ray-tracing

In order to simulate global illumination effects such as color blending and delta-based BRDFs, we employ the standard ray-tracing equation given as [1]:

$$L_o(x, v_o) = L_e(x, v_o) + \int_{\mathcal{H}^2} f_r(x, v_i, v_o)\; L_i(x, v_i)\; cos\theta\; dv_i \tag{9}$$

where $L_o(x, v_o)$ represents the radiance at point x in the outgoing direction v_o. $L_e(x, v_o)$ represents the emitted radiance emitted at point x in the outgoing direction v_o. $f_r(x, v_i, v_o)$ represents the material's BRDF function at location x with incoming direction v_i and outgoing direction v_o. $L_i(x, v_i)$ represents the incoming radiance at point x in the incoming direction v_i. $cos\theta$ represents the cosine of the angle between the outgoing direction v_o and surface normal n at point x. Finally, H^2 represents the hemisphere of valid directions at point x. We choose to integrate over the hemisphere, effectively considering all possible directions in the diffuse case. For delta BRDFs such as glass and mirror, it is sufficient to consider the single direction the ray is valid in.

The rendering equation is recursive in nature [2]given that the radiance L_o along bounce i is a function of future radiance values L_o along bounce $i + 1$. Yet we cannot infinitely recurse on this function due to memory and time limitations. We define ray-tracing as an n-bounce system, where, after n bounces, we terminate the system and back-propagate the radiance values to previous bounces. We can further use Russian Roulette to help terminate in instances when the future light contributions are predicted to be low.

It is important to note that most objects in our scene do not emit radiance given by the term $L_e(x, v_o)$. In fact, the only things that would do so are the lights in our scene. If we trace rays for n bounces, but are not able to hit any lights during that time, then we back-propagate no outgoing radiance through the points, thus producing zero signal.

To fix this, we force the last bounce towards the light source. Recall with diffuse materials, our outgoing rays can go in any direction along the hemisphere centered around the surface normal. In the general case, we would sample a random ray for our next recursive bounce, but if we exceeded the max number of bounces, then our final ray we can pick to be the direction straight to the light source. This helps us reduce the number of incidents where light paths gather no radiance by forcing the last ray to the light source so as to attempt to get some signal from the last bounce.

2.3 Next-Event Estimation

In order to gather even more radiance from the light source along rays, we can use next-event estimation [7] to force more rays towards the light source. Every time we hit a diffuse surface in Figure 7, we can generate the next ray bounce according to the current material's BRDF, while generating a second ray bounce towards the light source. This ensures that every diffuse bounce samples radiance from the light source in the instance that no occlusion between the bounce and light source exists.

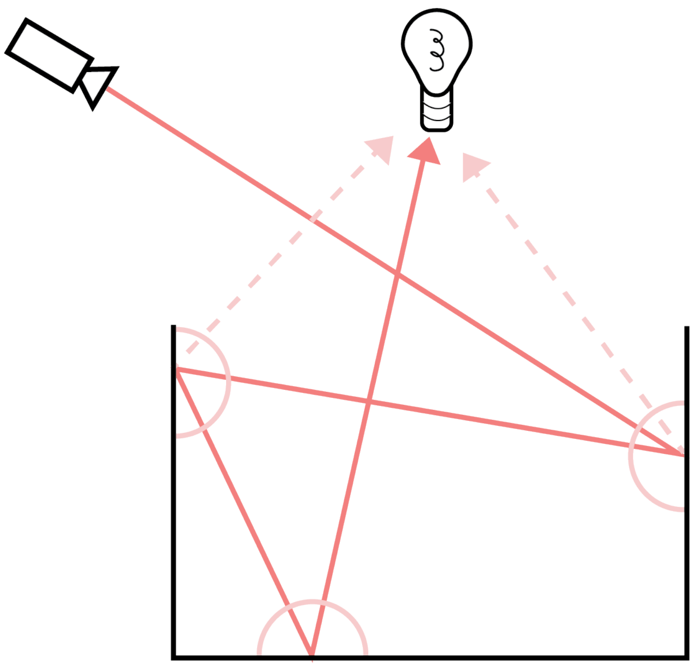


Figure 7: NEE with diffuse surface interactions. Each bounce we can trace a ray as we normally would while tracing a second ray to the light source.

2.3.1 NEE With Delta BRDFs

The issue with NEE arises when we have delta BRDFs with reflective and refractive surfaces. Recall with diffuse BRDFs, we can choose any direction for our next bounce. We no longer have that flexibility with delta BRDFs, and thus cannot force rays towards the light source each time we intersect a delta BRDF surface. To make matters worse, if we reach our last bounce in our n-bounce system and we intersect a reflective or refractive surface, we no longer can force our last ray towards the light source since that would invaldate the BRDF at that surface. Up until now, we have been optimistic that the last surface we hit is a diffuse surface, but the reality is that delta BRDFs make it more difficult to sample light effectively.

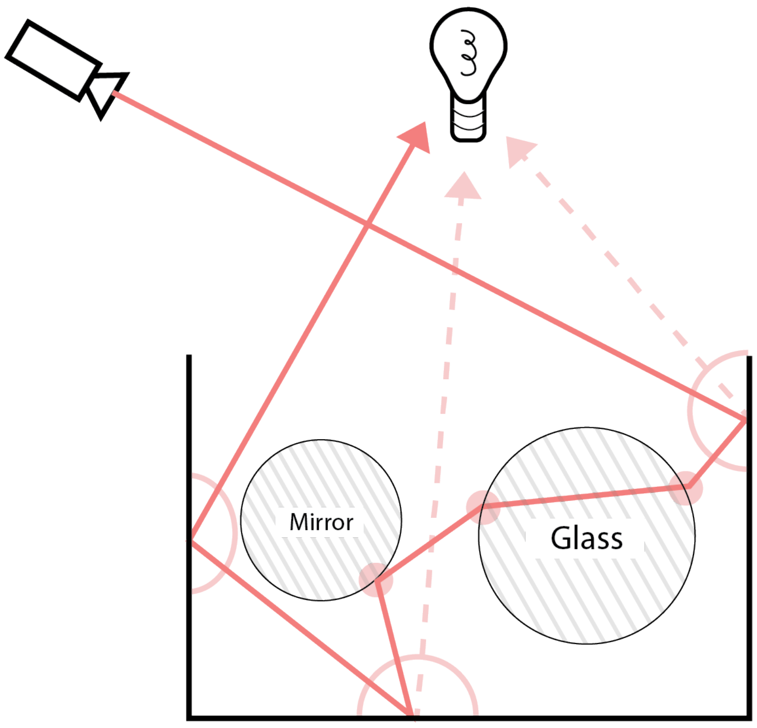


Figure 8: NEE with specular surface interactions. Intersections with delta BRDFs do not allow us to sample paths towards the light without violating the BRDF.

This can be seen with Figure 8, where out of our 6 bounces, 3 interact with delta BRDFs, preventing us from doing NEE in half of the bounces. The more reflective and refractive bounces we have, the less NEE samples we can take. This combined with occlusion from geometry obstructing the light source beckons the need for a better path finding algorithm that can use reflective and refractive surfaces to guide light paths more towards the light.

3 Contributions

The goal of this project is to develop algorithms for efficiently computing light paths connecting two points (e.g., a source and a sensor) through a sequence of specular and refractive surfaces. This problem is a generalization of Alhazen's problem, which aims to find a connective path between two points through a third intermediate point that lies on a convex reflective material, such as a sphere. Our work underlies modern Monte Carlo rendering algorithms for simulating light transport in scenes containing mirror-like and transparent objects. To realize this approach, we work on the following aspects of the problem:

1. We derive a differentiable formulation for specular ray tracing, in terms of only basic geometric queries (intersection, normal, and curvature queries) supported by most commonplace geometric representations.

2. We implement our approach in an existing physics-based renderer, Mitsuba 2 [16], and incorporate it into its implementation of path tracing and bidirectional path tracing rendering algorithms.

3. We use our implementation to render images of scenes full of specular and refractive objects, specified through a variety of geometric representations, including point clouds and neural implicit representations.

4 Related Works

4.1 Manifold Connections

The task of finding manifold connective paths between surface points through various intermediate materials is a long-sought after problem in rendering. One of the early works on differential ray tracing for sequences of specular surfaces by Mitchell and Hanrahan [5] provide an analytical solution for computing the derivative of the normal, also known as the curvature. They are also one of the earlier papers to discuss automatic differentiation in the context of path optimization. Chen and Arvo [8] introduced perturbations in specular paths along curved surfaces in order to compute nearby optical paths for fast approximations of specular surfaces, building on the field of differential rendering for path optimization. They compute a second-order Taylor expansion of the hit point x_i as a linear system in order to compute the next-step hit point in order to optimize a path between two points with x_i in the middle.

For longer paths more than a point long, Jakob and Marschner [12] explores manifold specular paths as a Monte Carlo problem. They define paths as a set of points and compute the contribution per pixel as:

$$f(x_1 ... x_n) = L * W \prod_{i=1} f(x_k, x_{k+1}) G(x_k, x_{k+1})$$

where L is the emitted radiance, W is the pixel's importance, f is the BRDF at x_k and G is the geometric factor. They generalize this geometric factor to be the derivative of projected solid angle at one vertex with respect to area at the other vertex. We can use chain rule to create a geometric factor for multiple bounces as the solid angle at one vertex with respect to the area at its successive vertex, and the area of the successive vertex with respect to the area of the next vertex, and so forth.

They define a specular manifold set S as all the specular components in the path and add the constraint that all the half-vectors of the incoming and outgoing rays to this point must be colinear with the normal. The Implicit Function theorem allows them to reparametrize a continuous set of specular points in terms of any two points and gives the derivative of all these points with respect to these two points. For simplicity, we use the endpoints of a specular path. This way, we can update one of the two endpoints while keeping the other fixed. We denote the gradient as C, which is the derivative of the constraints. We can use this data to help compute the solid angle, which in turn helps us compute the generalized geometric factor of our points needed for computing the pixel contribution.

The algorithm for this is as follows: assuming the initial and end points are diffuse, offset the initial position by some amount. Using C, we can identify how other specular points are expected to change when we change one of the endpoints. This step in each point is along the path perpendicular to the surface normal, which we correct for by projecting it back to the surface. We walk over the specular chain, updating the points until we reach our first diffuse point, which is now also offset. We can then run our algorithm again on the successive chain to reconnect the path back to the next available diffuse point. Thus, we need knowledge of the specular chains around the chain we are trying to update.

They formulate this algorithm as a Markov chain problem, where a step is taken proportional to the contribution function above where we try to maximize the pixel's signal. This proceeds as an iterative Newton method that can be backtracked in case of error. They extend their work to also account for glossy materials by modifying the constraints to be equal to some offset to capture the offset from ideal specular transport.

Further work by Zeltner et al. [19] builds on this by citing that different points of specular contact converged to different local optima, and that the contribution of the optima specular path should be weighted by the area of the search space that, when manifold-walked, produces the optima. Between any 3 points (x_1, x_2, x_3) where x_2 is specular, their algorithm is as follows: pick an initial guess for x_2 and solve using manifold-walks, then continue to sample for new values of x_2 and solve via manifold-walk until the same point is converged upon. At each step, update the probability of each solution following a Bernoulli distribution. While this produces an unbiased solution, it is slow to compute. A biased but lower variance solution would instead allocate M different values for x_2 and solve via manifold-walk, counting how many times each local optima is converged to estimate the probability of selecting from each region.

To render specular geometry with normal maps which may introduce a lot of noise into the convergence space, they alter their algorithm slightly to sample a normal offset from the normal map and keep the offset constant during the manifold walk of an iteration. This does not affect convergence while taking into account various normal when searching for local optima. To render glints, they similarly solve for a scene without glints, and refine their walk by adding glints back in, confining their search space to a small parallelogram based on ray differentiation.

4.2 Refractive Sampling

Walter et al. [11] extends this work by solving for the refractive paths from a point L outside a material to a point V inside a refractive material by computing the intersection point P that lie on the surface. Their results are targeted for triangular meshes by assuming the half vectors of PV and PL are colinear with the surface normal at P and use a 1D

Newton-Raphson iterative solver to find the target point x.

Pediredla et al. also found that refractive maifold paths can also be used to solve paths for refractive materials in heterogeneous index-of-refraction materials [18]. These heterogeneous refractive indexes cause light to bend due to the eikonal equations. These equations cannot be analytically solved for and must be solved for numerically using gradient descent. They case on two scenes: when one point is in the material and another is on the edge, and when both points are inside the material. The former is easier to solve for, allowing for a simpler computing pass. Given initial points x_0 and y, and a varying refractive index $n(x)$, the goal is to determine an initial velocity v_0 such that $x^* = y$ where x^* is the result of solving the initial boundary problem for arc-length s^*. This allows us to optimize the loss $L = 1/2||x^* - y||^2$ by taking the derivative of L with respect to v_0. This requires us to know $\frac{ds^*}{dv_0}$, which is possible since x^* lies on the boundary, so s^* depends on v_0 for all possible values of v_0. For the second case, we let $s = argmin_{x_s}||x_s - y||$, which translates to optimizing the implicit surface $(x_s - y)^T v_s = 0$, giving us a way to relate s^* to v_0. We can combine these equations to perform symplectic integration on the ray's position and velocity.

Computing an unbiased estimate also requires summing the contributions from every path, which in turn requires knowing the probability of a path for MC estimation. Zeltner et al. take a similar approach to specular manifold sampling [19] by solving for an initial path and continuing to solve random initializations until the same path is found, in which case a probability based off a Bernoulli distribution can be estimated. They also propose taking multiple samples to estimate multiple path probabilities to combine with MC, terminating after some Russian roulette probability.

5 Forward Path Rendering

Rendering scenes with multiple delta BRDFs can lead to problems where rays terminating on these same delta BRDFs cannot construct paths from the final hit point to the light source. Recall that with diffuse BRDFs, we can essentially pick any path to the light source. We extend this idea by terminating rays earlier at a diffuse object, and instead seek to search for a final path to the light source, noting that the final path is not always a linear connection between the object and light due to occlusion from objects that lie between the two. It may instead be the case that there exists a path from the sensor to the light that hits a mirror surface along the way. We aim to develop algorithms for efficiently finding these light paths in Figure 9 through reflective and refractive materials.

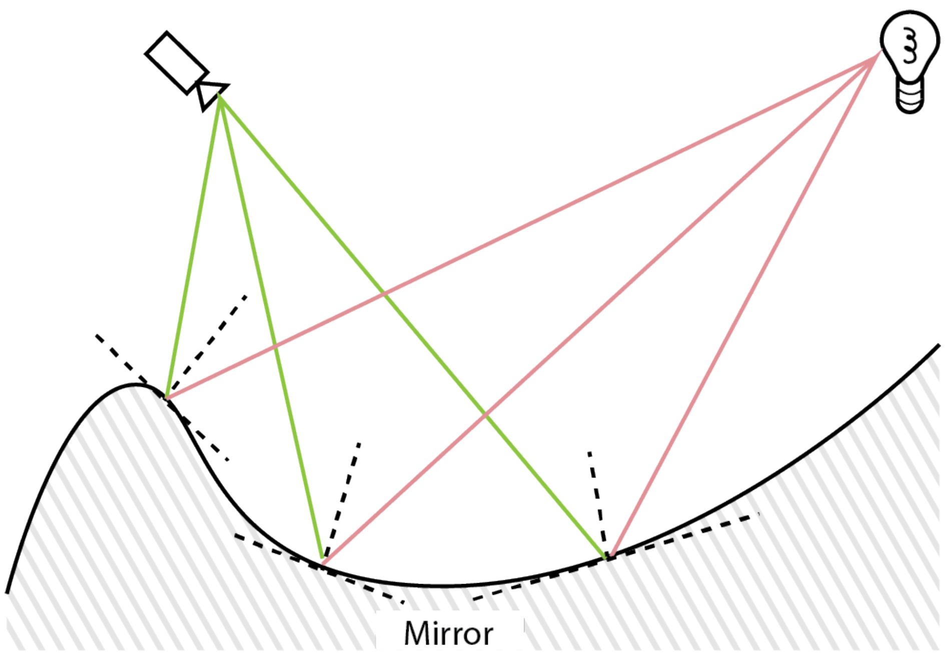


Figure 9: Multiple reflective paths between a source and target point.

5.1 Path Definition

We formulate the rendering problem as a series of linear functions defined by an origin x, unit direction v and time t along the unit direction. Together they comprise a set of line segments illustrated below, where our goal is to minimize the distance between the target light source and any point along the line segments.

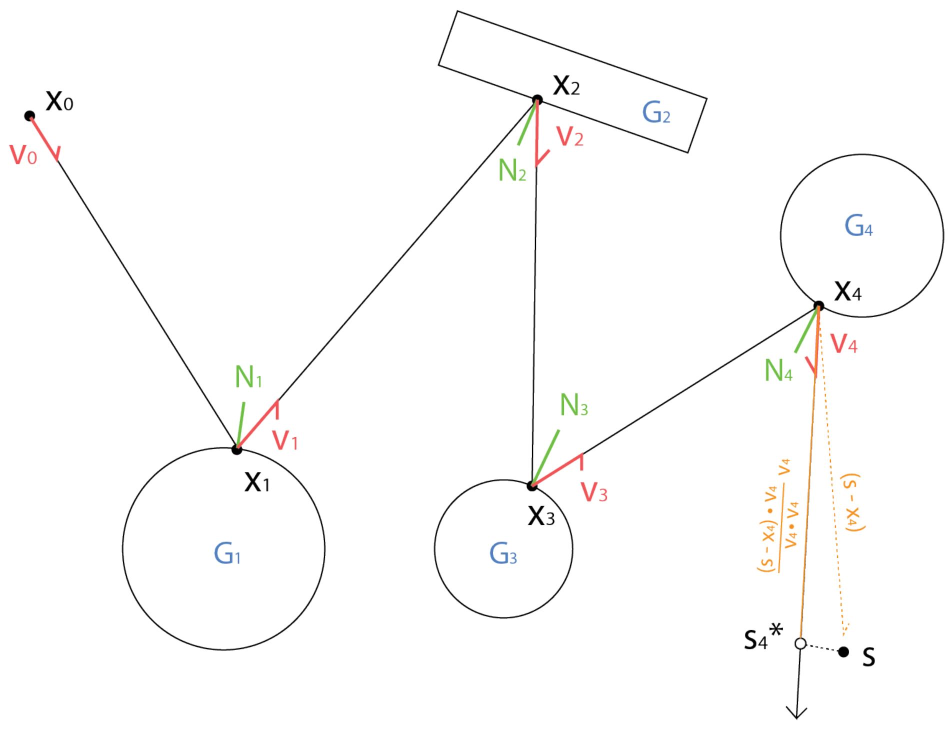


Figure 10: Forward path tracing formulation.

- k [1x1] total number of simulated bounces
- i [1x1] current bounce in range $[0, k]$
- x_0 [1x3] initial point
- v_0 [1x3] initial direction
- x_i [1x3] point of contact on surface i
- v_i [1x3] outgoing direction on surface i
- $G_i(x_i)$ [1x1] implicit geometry function of geometry i at surface point x_i
- $n_i(x_i)$ [1x3] normal for geometry i at point x_i (shorthand n_i)
- s [1x3] sensor point (target)
- s_i^* [1x3] closet point to s between points x_i and x_{i+1} (x_{i+1} may not exist)

We intersect a geometry at point x_i if $G_i(x_i) = 0$. This allows us to take normally explicit geometric representations such as triangles and points and rewrite them as implicit representations.

$$G_{sphere}(x, (o, r)) = ||x - o||^2 - r^2 \tag{10}$$

$$G_{plane}(x, (o, n)) = (x - o) \cdot n \tag{11}$$

Where (o, r) are the sphere's origin and radius respectively, and (o, n) are the plane's origin and normal respectively. We can think of a point as a special case of a sphere with radius $r = 0$. Furthermore, we can think of a triangle as a special case of a plane with the added condition that the intersection point x lies within the range of the triangle's three vertices (v_0, v_1, v_2) which we can compute by checking the barycentric coordinates (α, β, γ).

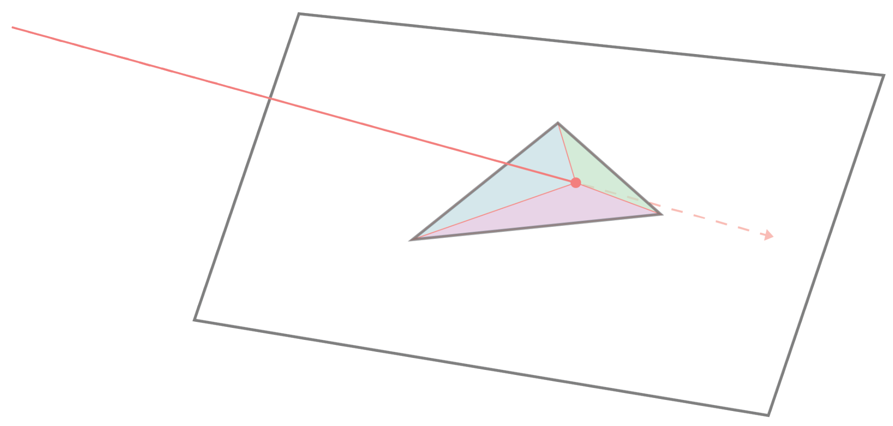


Figure 11: Planar assumption for triangles. We intersect a triangle using the plane formula while also checking the barycentric coordinates to check if inside triangle.

To get the normals for the above shape in Figure 11, we can take the derivative of the implicit geometry functions with respect to the intersection point x. For the sphere, we have to normalize the result, which adds an extra division term to the derivative.

$$\frac{dG_{sphere}(x,(o,r))}{dx} = 2 * (x - o) \tag{12}$$
$$N_{sphere}(x,(o,r)) = (x - o)/||x - o|| \tag{13}$$

$$\frac{dG_{plane}(x,(o,r))}{dx} = n \tag{14}$$
$$N_{plane}(x,(o,n)) = n \tag{15}$$

This gives us the following equation for a ray:

$$x_i = x_{i-1} + v_{i-1} * t_{i-1} \tag{16}$$

At each step i of our render, we generate a ray at origin x_i and direction v_i into our scene and find the closest intersection time t_i by checking intersections with the geometric functions of all primitives given by Equations 10 and 11.

We can use Equation 16 to get the new ray origin x_{i+1}. We check the material type of the intersected geometry, which tells us how to compute the next direction v_{i+1}. For reflective and refractive materials, we use Equation 4 and Equation 8 respectively. Both equations require the geometry's normal, which we can get from Equations 13 and 15. This gives us a recursive way of computing the sequence (x_i, v_i, t_i) for k bounces.

Our goal is to minimize the distance between our best s_i^* for some i and the target s, which in this case is our light source.

$$L = min_{i \in [0,k]} ||s_i^* - s||^2 \tag{17}$$

This ends up as a search problem, where we consider the L2 norm squared of every single ray i we trace up until termination. To compute s_i^* for a given ray, we solve for the following equation:

$$s_i^* = x_i + v_i t_i^* \tag{18}$$

Where t_i^* is computed as:

$$\begin{aligned} t_i^* &= (s - x_i) \cdot v_i / ||v_i||^2 \qquad (19) \\ &= (s - x_i) \cdot v_i / (v_i \cdot v_i) \qquad (20) \end{aligned}$$

It is important that t_i^* is clipped between $[0, t_i]$ so that it falls within the bounds of the the line segment that is the ray at bounce i. When i is the final bounce, we set $t_i = \infty$. Once we compute t_i for the ray i, we can the compute the loss L_i from Equation 17 associated with that ray, taking the min of all L_i's as we traverse each ray.

6 Backwards Path Rendering

Our goal is to maximize the radiance per pixel, which we can do by finding paths that reach the light source. For diffuse surfaces, we can use NEE to sample from the lights when no occlusions exist between the current hit point and the light source. Yet for delta BRDFs, we have to carefully pick paths that allow us to reach the light source. This is possible by setting the target point s as the light source, and optimizing for the initial ray's direction v_0 that minimizes the loss function. In other words, we aim to compute $\frac{dL}{dv_0}$.

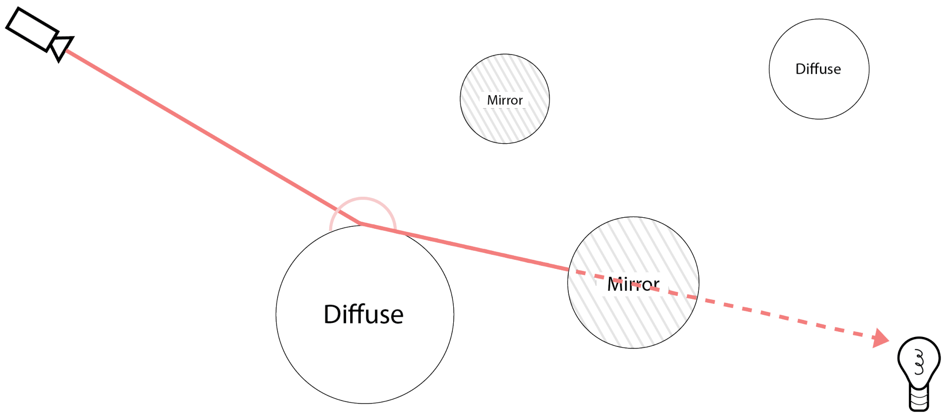


Figure 12: NEE failure case. Rays from the first diffuse bounce cannot reach the light source due to occlusion from other scene geometry.

Rather than having s be a point on the light like in Figure 12, we can also have it be another point on a diffuse surface. If we look at the above scenario, we can use NEE after our first diffuse bounce to measure any emittance from the light, but quickly find that a mirror in our way occludes any sort of light from reaching us. The question now is whether there is another reachable objective for us to optimize.

The solution is to find our way to another point on a diffuse surface, whether it be the current surface or another surface, so that we can try NEE this time from that point. This idea extends further into bidirectional path-tracing [6] where we aim to connect two rays with each other, but there may be delta-BRDF geometry that stands in the way of their connection, so we aim to find a connecting path between the two ends of the rays.

We can set our initial position x_0 to be the intersection point on the current diffuse surface, and the target point s to be a random point on another diffuse surface. We then initialize a random direction v_0 and attempt to minimize the loss L so that we reach the target point s as close as possible. Sometimes it may be a direct connection, other times it may be several bounces to the point. It all depends on how we initialize v_0.

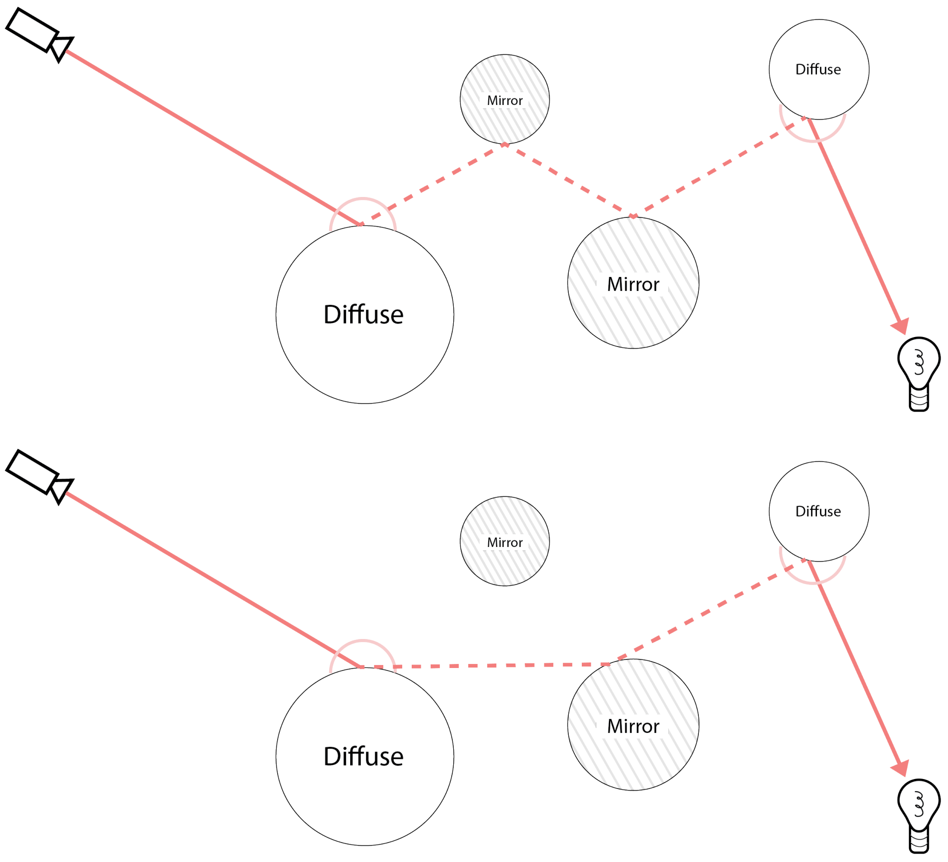


Figure 13: Retargeting paths to other diffuse objects allow us to continue NEE without occlusion. Manifold connections do not need to be between a point and a light source, but may also be between two surface points.

Figure 13 shows two examples of possible paths connecting to a target point on the rightmost diffuse surface. From here, we attempt NEE again and find we are able to sample the light.

6.1 Intersection Derivatives

$$\frac{dL}{dv_0} = 2(s^*_{i^*} - s) \cdot \frac{ds^*}{dv_0} \tag{21}$$

We start by computing the derivative of L with respect to v_0 from Equation 17. Here we are only interested in the rays in range $[0, i^*]$, where $i^* < k$ is the ray bounce with the closest point to s that lies on it. All other rays after that do not contribute to the computational graph that is used to compute the loss L, so we can ignore them.

$$\frac{ds^*}{dv_0} = \frac{dx_n}{dv_0} + \frac{dv_n}{dv_0}t^* + v_n\frac{dt^*}{dv_0} \tag{22}$$

We then use chain rule to compute the derivative of Equation 18. This requires us to know several derivatives at this point.

$$\frac{dt^*}{dv_0} = [((s - x_n) \cdot \frac{dv_n}{dv_0} - \frac{dx_n}{dv_0} \cdot v_n) * (v_n \cdot v_n) - (2 * v_n \cdot \frac{dv_n}{dv_0}) * ((s - x_n) \cdot v_n)]/(v_n \cdot v_n)^2 \tag{23}$$

Derivative of Equation 20. This is a special case since the final time t^* is computed differently than other time values t_i that come before.

$$\frac{dx_i}{dv_0} = \frac{dx_{i-1}}{dv_0} + \frac{dv_{i-1}}{dv_0} * t_{i-1} + v_{i-1} \cdot \frac{dt_{i-1}}{dv_0} \tag{24}$$

Derivative of Equation 16. This is a similar format to Equation 22 because the original function is recursive. x_{i-1} is used to compute x_i and so forth.

Computing the derivative $\frac{dv_n}{dv_0}$ is a little more involved as it depends on the BRDF of the intersection geometry G_i. Due to the simplification of our algorithm, we only consider reflective and refractive surfaces in our derivatives. Any diffuse surfaces are neglected, because if we stumbled on a diffuse surface, we can reset our algorithm's origin x_0 to the diffuse intersection point and continue our search from there.

$$\frac{dv_i}{dv_0} = \frac{dv_{i-1}}{dv_0} - 2 * n \cdot (\frac{dv_{i-1}}{dv_0} \cdot n + v_i \cdot \frac{dn_i}{dv_0})) - 2 * \frac{dn_i}{dv_0} * (n \cdot v_i) \tag{25}$$

The derivative of the reflective BRDF in Equation 4. Computing the derivative for the refractive case is more complicated, so we will rewrite our refractive case in simpler terms.

$$cos_i = (v_i \cdot n) \tag{26}$$

$$k_i = 1 - (\frac{\eta_1}{\eta_2})^2 * (1 - cos_{i-1}^2) \tag{27}$$

$$sqrtk_i = \sqrt{k_i} \tag{28}$$

$$v_i = \frac{\eta_1}{\eta_2} * v_{i-1} - (\frac{\eta_1}{\eta_2} * cos_{i-1} + sqrtk_i) * n \tag{29}$$

Thus, we can compute the derivative for refraction by stitching together the derivatives of its inputs.

$$\frac{dcos_i}{dv_0} = \frac{dv_i}{dv_0} \cdot n + v_i \cdot \frac{dn}{dv_0} \tag{30}$$

$$\frac{dk_i}{dv_0} = 2 * \frac{\eta_1}{\eta_2} * \frac{\eta_1}{\eta_2} * cos_{i-1} * \frac{dcos_{i-1}}{dv_0} \tag{31}$$

$$\frac{dsqrtk_i}{dv_0} = \frac{\frac{dk_i}{dv_0}}{2 * \sqrt{k}} \tag{32}$$

$$\frac{dv_i}{dv_0} = \frac{\eta_1}{\eta_2} * \frac{dv_{i-1}}{dv_0} - (\frac{\eta_1}{\eta_2} * \frac{dcos_{i-1}}{dv_0} + \frac{dsqrtk_i}{dv_0}) * n - (\frac{\eta_1}{\eta_2} * cos_{i-1} + sqrtk_i) * \frac{dn}{dv_0} \tag{33}$$

The derivative of the refractive BRDF in Equation 8.

Getting the derivative for the time t with respect to the initial velocity v_0 is a bit different since we did not have any formal way to compute the intersection time t. Rather, we said that the implicit surface function G_i must meet the constraint that $G_i(x_i) = 0$. If we rewrite x_i in terms of other variables we know from Equation 16, we get the following:

$$0 = G_i(x_{i-1} + v_{i-t} * t_{i-1}) \tag{34}$$

We know that $G_i(x_i) = 0$, and so any changes to x_i must still ensure that $G_i(x_i) = 0$. By rewriting x_i in terms of its components $(x_{i-1}, v_{i-1}, t_{i-1})$, we suddenly have a t_{i-1} value to work with, so we take the derivative of the implicit functions and its arguments while still ensuring that their derivative equals 0 due to the previously mentioned constraint. We can then isolate $\frac{dt_{i-1}}{dv_0}$ as shown below.

$$0 = \frac{dG_i}{dv_0} \cdot (\frac{dx_{i-1}}{dv_0} + t_{i-1} * \frac{dv_{i-1}}{dv_0} + v_{i-1} * \frac{dt_{i-1}}{dv_0}) \tag{35}$$

$$\frac{dG_i}{dv_0} \cdot (v_{i-1} * \frac{dt_{i-1}}{dv_0}) = -\frac{dG_i}{dv_0} \cdot (\frac{dx_{i-1}}{dv_0} + t_{i-1} * \frac{dv_{i-1}}{dv_0}) \tag{36}$$

$$\frac{dt_{i-1}}{dv_0} = -\frac{dG_i}{dv_0} \cdot (\frac{dx_{i-1}}{dv_0} + t_{i-1} * \frac{dv_{i-1}}{dv_0})/(\frac{dG_i}{dv_0} \cdot v_{i-1}) \tag{37}$$

6.2 Geometric Derivatives

We need to know the geometric derivatives for the implicit surfaces in order to compute the time derivatives. These values are similar to the geometric normals without the normalization factor

$$\frac{dG_i}{dv_0}_{sphere} = 2 * (x_i - o_i) \cdot \frac{dx_i}{dv_0} \tag{38}$$

$$\frac{dG_i}{dv_0}_{plane} = n \cdot \frac{dx_i}{dv_0} \tag{39}$$

We also need the derivatives of the normals with respect to the input direction v_0.

$$\frac{dn_i}{dv_0}_{sphere} = \frac{2 * ((x_i - o) \cdot (x_i - o)) * \frac{dx_i}{dv_0} - (x_i - o) \cdot (2 * ((x_i - o) \cdot \frac{dx_i}{dv_0}))}{2 * ((x_i - o) \cdot (x_i - o))^{\frac{3}{2}}} \tag{40}$$

$$\frac{dn_i}{dv_0}_{plane} = 0 \tag{41}$$

What about intersecting triangulated surfaces like meshes? This is where our logic for treating a triangle as a plane comes in. When we do our forward pass, we check if a point intersects a triangle by checking if a point lies within the barycentric coordinates of the triangle. Yet during our backwards pass, we only care about the implicit geometry derivative $\frac{dG_i}{dv_0}$, which for a triangle, is the same as a plane since we only care about the normal. This leads us to the following assumption:

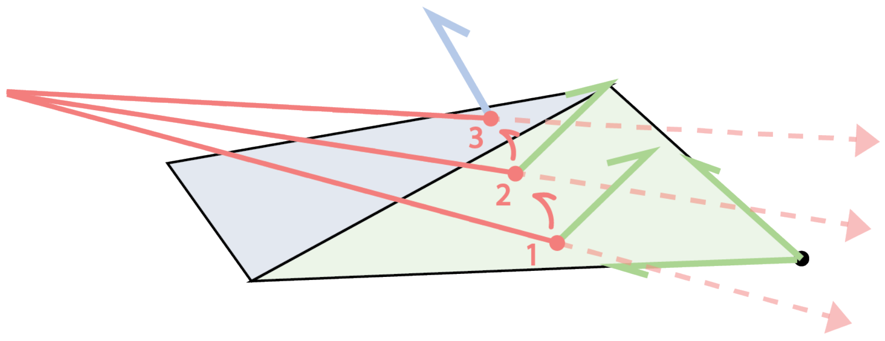


Figure 14: Planar assumption for normals on a triangle. Intersections on the same triangle share the same normals, leading to discontinuities in both the normals and the resulting gradients when switching between triangles.

For each triangle in Figure 14, we compute the normal by taking the cross products of two edges. While this may work in theory, it means that our derivatives for the normals are effectively 0 for all triangles, since a small change in position x_i when computing the surface normal $N(x_i)$ still gives us the same normal as seen by Equation 39. Thus, we make the assumption that

$$\frac{dn_i}{dv_0}_{triangle} = \frac{dn_i}{dv_0}_{plane} = 0 \tag{42}$$

This gives us the nice property that a lot of values in Equation 25 and Equation 33 cancel out when we compute the derivative, thus reducing the complexity of the computational graph.

Yet this can also be an issue because it creates a non-continuous set of normals along a mesh of triangles, which as a result creates a non-continuous set of derivatives $\frac{dL}{dv_0}$ as we can experience sudden jumps in the derivatives when switching between surfaces of the same mesh.

To fix this, we can use a mesh with per-vertex normals that we can interpolate. Recall when checking for the intersection of a triangle, we compute the barycentric coordinates (α, β, γ). Then, when we want to compute the normal at a point, we can use these coordinates that we get for free from the intersection check to interpolate their values to the current intersection point. We still need to normalize the output, as a weighted average of normalized values does not ascertain that the output will also be normalized.

$$n' = \alpha(x) * N(p_0) + \beta(x) * N(p_1) + \gamma(x) * N(p_2) \tag{43}$$

$$N_{triangle}(x, (p_0, p_1, p_2),) = \frac{n'}{||n'||} \tag{44}$$

Now we have an equation for the normal that relies on the hit point x, which in turn relies on the input direction v_0. Taking the derivative, we get:

$$\frac{dn'}{dv_0} = \frac{\alpha(x)}{dv_0} * N(p_0) + \frac{\beta(x)}{dv_0} * N(p_1) + \frac{\gamma(x)}{dv_0} * N(p_2) \tag{45}$$

$$\frac{dN_{triangle}}{dv_0} = \frac{\frac{dn'}{dv_0}}{(n' \cdot n')^{\frac{1}{2}}} - \frac{(\frac{dn'}{dv_0} \cdot n') * n'}{(n' \cdot n')^{\frac{3}{2}}} \tag{46}$$

The vertex normals $(N(p_0), N(p_1), N(p_2))$ are constant features of the mesh, so we do not differentiate them. Now when we go to process normals in Figure 15, we get smoother results per step, leading to smoother derivatives of the loss, and an overall smoother search space.

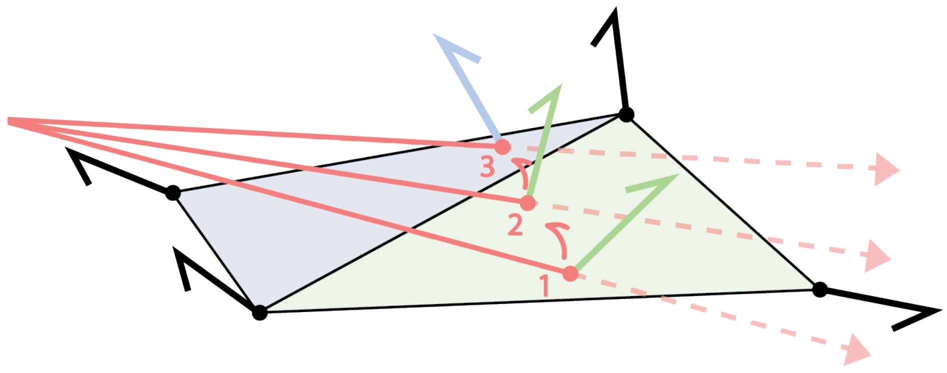


Figure 15: Barycentric interpolation of per-vertex normals on a triangle. This approach allows for smoother normals and smoother gradients between triangles on the same mesh.

6.3 Computational Graph

It is important to notice how we take a derivative of the normal with respect to the input ray direction $\frac{dn_i}{dv_0}$. This is because the input point from the previous iteration x_{i-1} is used to compute the normal for the current step n_i that is then used to compute the outgoing ray direction v_i in Figure 16.

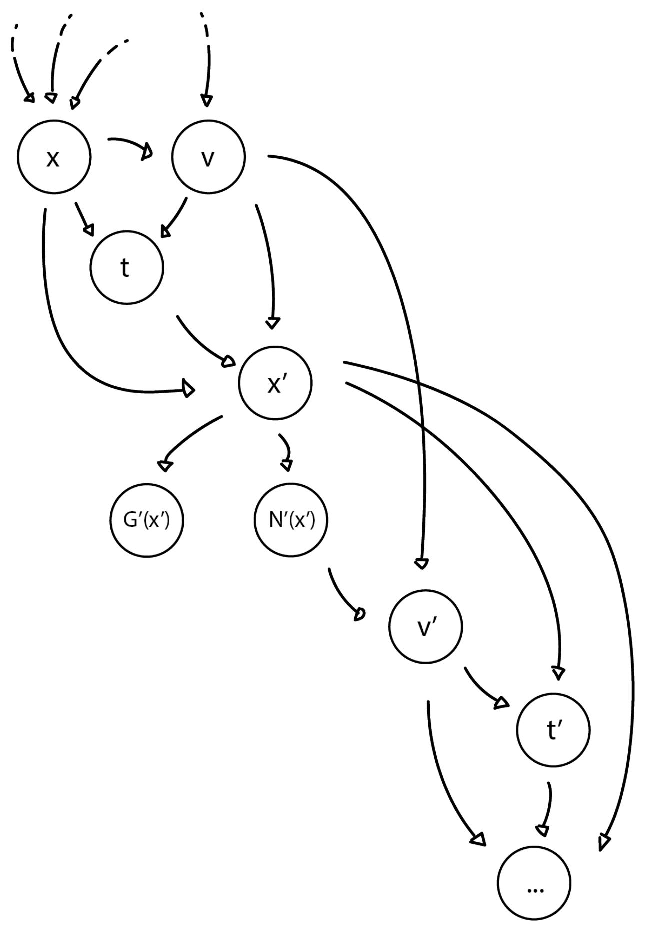


Figure 16: Computational graph of forward raytracing pass.

To walk through it in more steps, consider the computation graph for a current ray iteration (x, v, t) and future ray iteration (x', v', t'). Either v is given to us from initialization, or it is computed from the inductive case. We then intersect the scene geometry given the ray constructed from (x, v) to get the time t. From there we can use Equation 16 to compute the new hit point x', which is also the same point we used to evaluate the implicit geometry function $G(x') = 0$ for a hit. This allows us to compute the normal $N(x')$ at the hit point location using the appropriate geometric function. Then we compute the next iteration's direction v' using either Equation 4 or 8 depending on the geometry's BRDF. Now we have a new ray (x', v') that we can perform intersection on to get the new time t', and the process repeats.

6.4 Polar Coordinates

Since we are optimizing for the direction v_0, and since v_0 is a unit vector, we can rewrite it as a set of polar coordinates (θ, ϕ). We can convert from polar to cartesian coordinates using the following formula:

$$x = sin(\theta) * cos(\phi) \tag{47}$$

$$y = sin(\theta) * sin(\phi) \tag{48}$$

$$z = cos(\theta) \tag{49}$$

And from cartesian to polar:

$$\theta = acos(z) \tag{50}$$

$$\phi = atan2(y, x) \tag{51}$$

$$\phi = \phi < 0 \ \ ? \ \ \phi + 2 * \pi \ \ : \ \ \phi \tag{52}$$

Because Equation 53 is only of the two equations used during the forward pass, we only need the derivative from polar to cartesian.

$$\frac{dv_0^{cart}}{dv_0^{polar}} = \begin{bmatrix} cos(\theta) * cos(\phi) & sin(\theta) * -sin(\phi) \\ cos(\theta) * sin(\phi) & sin(\theta) * cos(\phi) \\ -sin(\theta) & 0 \end{bmatrix} \tag{53}$$

From here, we will refer to $\frac{dL}{dv_0}$ as the [1x2] gradient of the loss L in terms of the polar coordinates $v_0^{polar} = (\theta, \phi)$

6.5 Optimization

Once we have a way to compute $\frac{dL}{dv_0}$ we can optimize our initial direction using gradient descent. We can use several types of optimizers, such as linear, adagrad [15], and adam [14], while noting that the latter optimizers require more hyperparameters to configure. To reduce the number of hyperparameters, we consider a simpler case of gradient descent [19]:

$$\beta = max(\beta * \beta_{scale}, \beta_{max}) \tag{54}$$

$$v_0 = v_0 - \beta * \frac{dL}{dv_0} \tag{55}$$

In each step, we increase β by some small factor $\beta_{scale} = 1 + 1\text{e}{-5}$ while ensuring it does not go over some value $\beta_{max} = 0.1$, although these values change depending on the scale of the scene. In the event that our loss L suddenly increases, either due to a change in surfaces or too large a step, we half the value β to slow down the step process and allow itself to comfortably search the small space around it in order to get back on track.

Our convergence criteria is simple: i) our loss L is less than some threshold L_{thres}, or ii) we spend too many iterations searching and terminate with failure. The value of L_{thres} again is dependent on the scale of the scene, as a scene with geometry twice as far apart requires an L_{thres} value twice as large to get the same results. We find that in many of our trials that 100 iterations of search is sufficient before terminating.

6.6 Sampling Probability

Once we converge to a good enough solution, we then need to sample the ray's contribution to the incoming emittance in Equation 9. This requires us to know the probability of selecting a ray, which we can rewrite in sampling terms as the following.

$$L_o(x, v_o) = L_e(x, v_o) + \frac{1}{N} \sum_{j=0}^{N} \frac{f_r(x, v_j, v_o) Li(x, v_j) cos\theta}{p(v_j)} \tag{56}$$

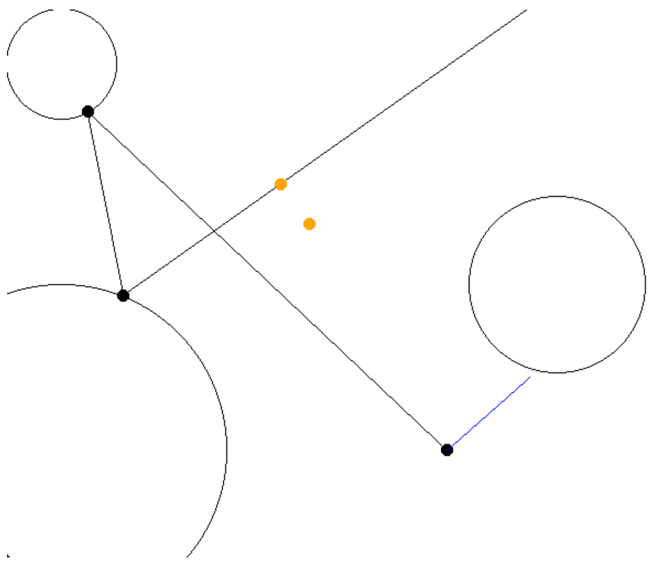
Recall that there may be several possible paths to the goal depending on how we initialize v_0. To correctly account for the contribution, we need to compute $p(v_0)$ which we can do by Bernoulli trials [19].

Rather than taking hundreds of random samples of initial values v_0, optimizing them, and counting how many have similar paths to the original path, we instead run several random samples of v_0 that we optimize until we get a path similar to our first path. Our probability is then the inverse number of trials.

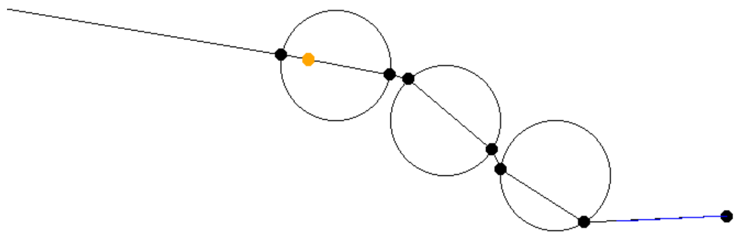
7 Trial Systems

7.1 2D Interactive Interface

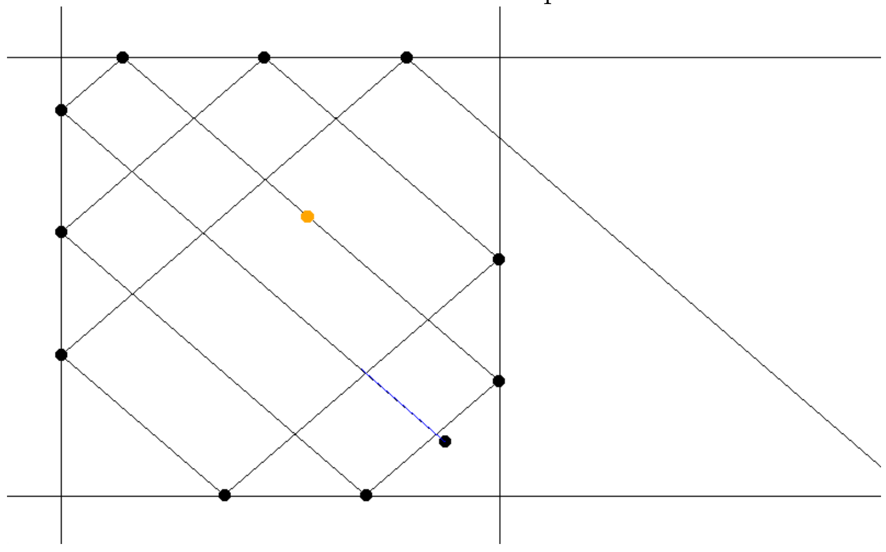
To test our system out, we begin by implementing our derivations in a simple 2D render environment. For simplicity, our environment supports planes and spheres, as well as reflective and refractive materials.



Scene 1: Reflective spheres



Scene 2: Refractive spheres



Scene 3: Reflective box

Incident bounce points are shown as black dots, with a max depth of 10 bounces. Following that, the 11th bounce will not intersect any scene geometry but rather extend to infinity. The orange dots represent the target and the closest point on any availble ray to the target. In scenes where there is only one orange dot, the target and the closest point overlap, signaling success. We also visualize a green and blue line from the ray's initial origin, where the green line visualizes the numeric gradient and the blue line visualizes the analytic gradient. The numeric gradient is computed as:

$$\frac{dL}{d\theta} = \frac{L(\theta + \epsilon) - L(\theta)}{\epsilon} \tag{57}$$

Keeping in mind we only consider the θ coordinate since we are in 2D. The numeric gradient is computed by tracing two rays while the analytic gradient traces one ray and uses the computation graph to produce the gradient.

In the above scenes, the blue line overlaps the green line, verifying that the gradients in both cases match. In the first scene, the gradient is perpendicular to the ray's initial direction, showing that the system has not finished converging. In the two scenes following it, the gradient overlaps the current ray direction, showing that any further steps will not change the initial direction of the ray. It is in these later two scenes that we also see the ray properly reach the target, even if the target lies inside a refractive material.

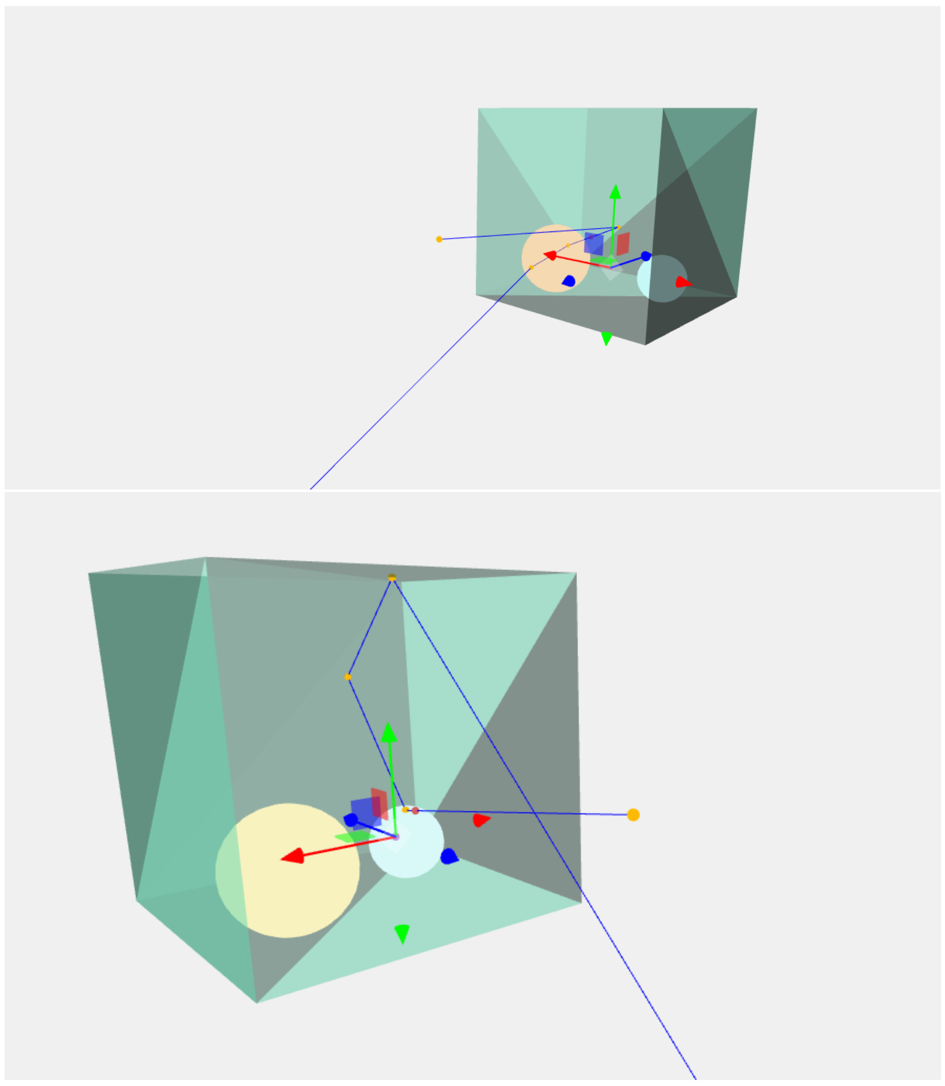
7.2 3D Interactive Interface

To verify our system further, we wrote a 3D test environment to test both (θ, ϕ) polar coordinates. Again we take the numeric gradient as:

$$\frac{dL}{d\theta} = \frac{L(\theta + \epsilon, \phi) - L(\theta, \phi)}{\epsilon} \tag{58}$$

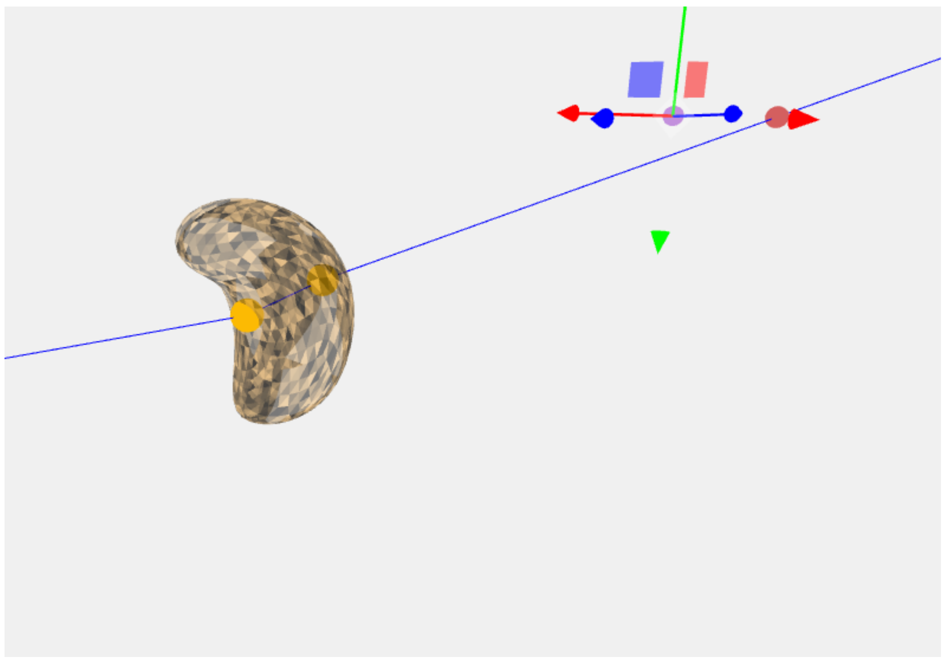
$$\frac{dL}{d\phi} = \frac{L(\theta, \phi + \epsilon) - L(\theta, \phi)}{\epsilon} \tag{59}$$

And compare it to our analytical gradient to verify them. Keep in mind this approach requires three ray traces to compute the numeric solution.

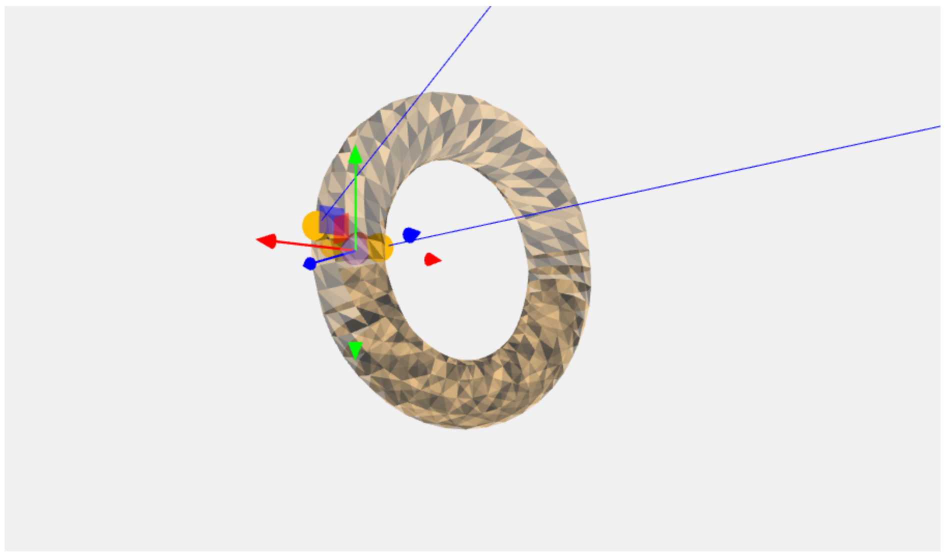


Scene 1: Cornell box

We attach our target point to an object with drag controls so that we can move the target around and watch our ray convergence adapt. In our simple Cornell box scene above, we test its ability to converge when interacting with both reflective (green) and refractive (orange) objects. The shades of green do not signify anything other than the scene is comprised of multiple triangles rather than planes.



Scene 2: Bean



Scene 3: Torus

Our scene can also support any .obj files comprised of hundreds of primitives. The above were files taken from 15-462's media directory.

8 Rendering

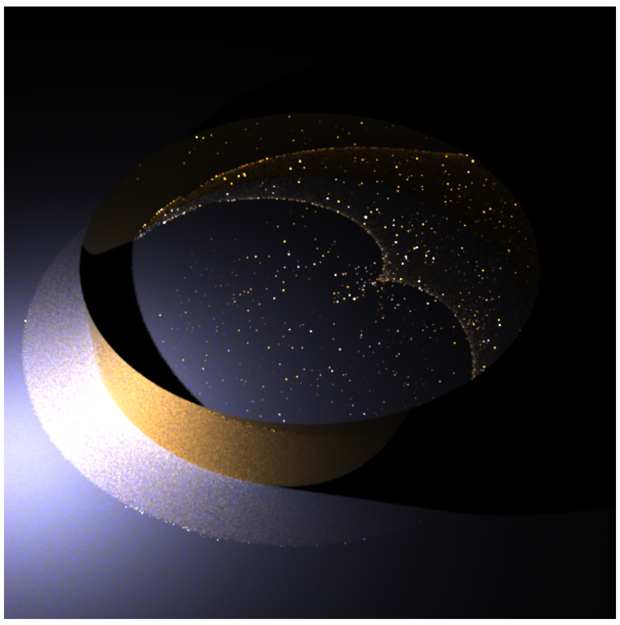
We implemented our manifold path search algorithm in the research rendering codebase Mitsuba 2 [16]. Mitsuba provides code for building accelerated geometric structures for primitive querying, intersection tests for many explicit and implicic primitives, as well as various material types with support for normal and displacement maps.

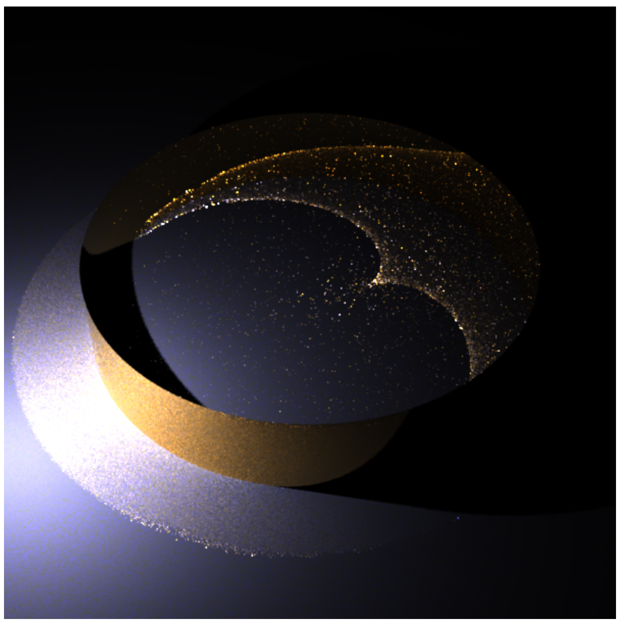
We extended the work of Zeltner et al. [19] by adding in the ray's derivatives for the backwards pass. Given a start point x_0 and a destination light source s, we randomly generate a direction v_0 by sampling a random point on another object x_1 and setting $v_0 = norm(x_1 - x_0)$. Normally we would generate a random (θ, ϕ), but that results in many of the rays not intersecting any surface, so instead we pick a direction v_0 that is guaranteed to hit some point x_1 and continue tracing for k bounces. We then compute the gradient $\frac{dL}{dv_0}$ and update v_0 using the optimizations in Equation 55.

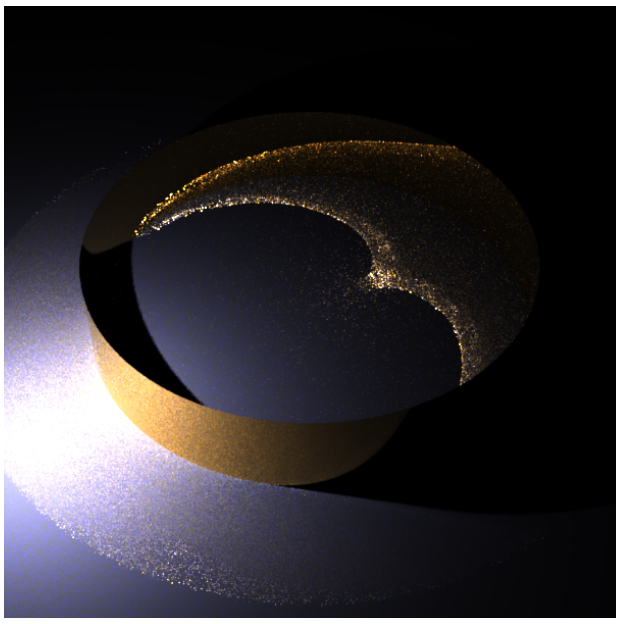
If the current loss L is better than some threshold L_{thres}, we consider the ray a success and the compute the incoming radiance from it using Equation 9, approximating the probability using Bernoulli trials.

8.1 Single-Bounce System

Below is a gallery of scenes showing the rendering results for different configurations.







Scene 1: Reflective Ring

$L_{thres}^{left} = 0.05$, $L_{thres}^{right} = 0.5$, $L_{thres}^{bottom} = 5.0$

We perform single-bounce raytracing on a reflective ring above. One interesting feature is that the ring geometry is actually a cylinder with an implicit geometric function, yet despite that, we treat it as a discretized set of planes, where at each intersection, we set the normal derivative $\frac{dn}{dv_0} = 0$, a similar property we said would be true for triangles under the plane assumption. Despite this generalization, we still get good convergence results, although we predict better results in fewer iterations had we taken the time to compute the normal derivative for this implicit geometry.

Another interesting point is that, when we increase L_{thres}, we get more samples, as can be seen by a larger ring of samples around the ring itself. This does not mean our algorithm necessarily performs better, but instead it makes it easier to converge. A good analogy is if we are trying to score a goal, instead of kicking the ball more accurately, we instead just make the goal bigger. The reason such an approach works is when we consider the type of light source we are trying to reach in Figure 17.

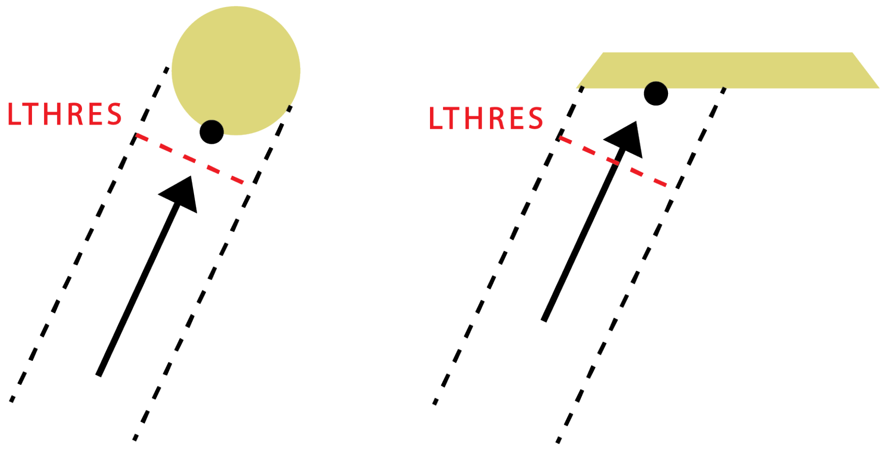
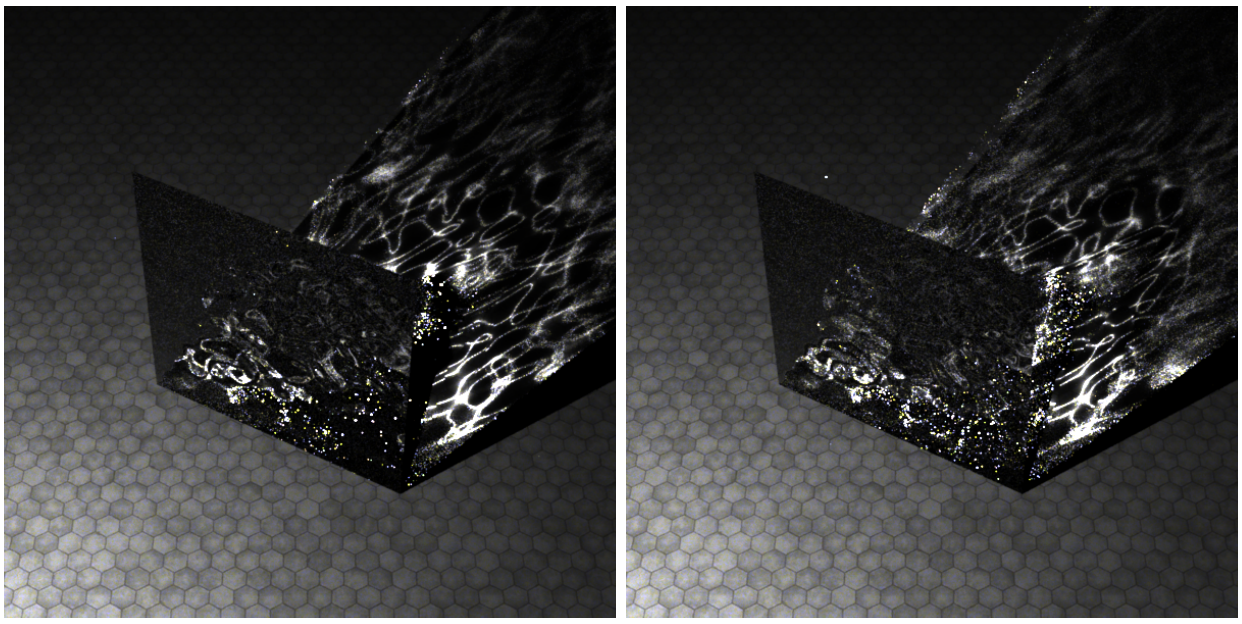


Figure 17: Setting a the loss threshold. A larger threshold allows for more possible rays to reach the light, whereas too large a threshold may cause rays to miss the light source.

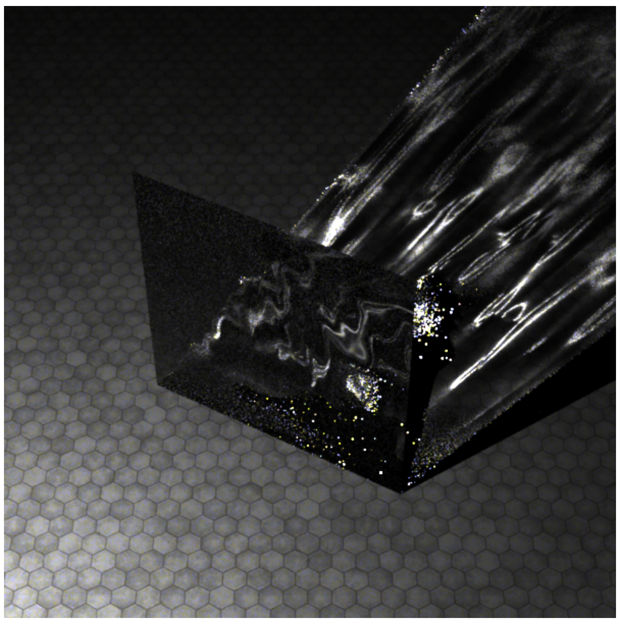
Recall with point lights, we need to reach the light perfectly in order to sample from it. Real, physically accurate lights are not points, but rather have some volume. For larger lights, we are allowed an even larger 'goal' to score into, allowing us a larger L_{thres} value. The same is even more true for area lights when we have a large surrounding area around our target. In this scene and all scenes following, we use area lights, explaining why such large L_{thres} values work well. The closer the target point is to the center of the light, the larger a L_{thres} we are allowed, which is why it is best to set the target s to be the center of the light.



Scene 2: Gaussian-Mapped Refractive Plane

$L^{left}_{thres} = 0.5$, $L^{right}_{thres} = 5.0$

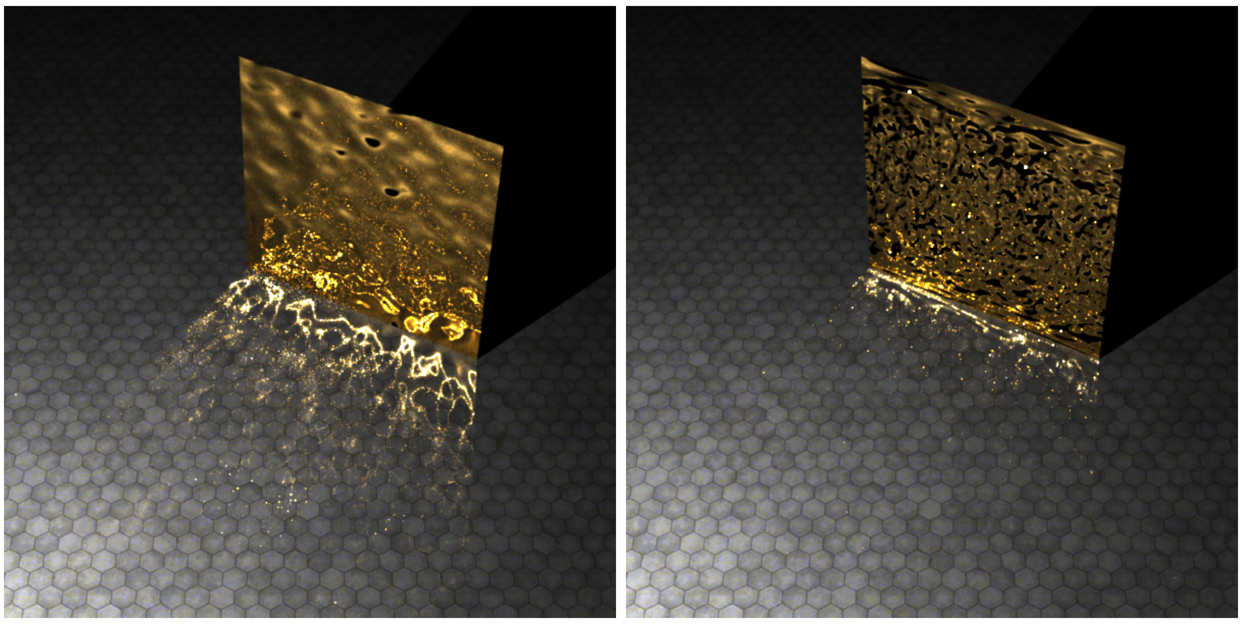
We further perform single-bounce raytracing on a refractive gaussian normal-mapped plane. Normally refraction is treated as a two-point problem of refracting into and out of the surface, yet since the plane we refract into has no depth, we never actually enter the material.



Scene 3: Fractal-Mapped Refractive Plane

$L_{thres} = 0.5$

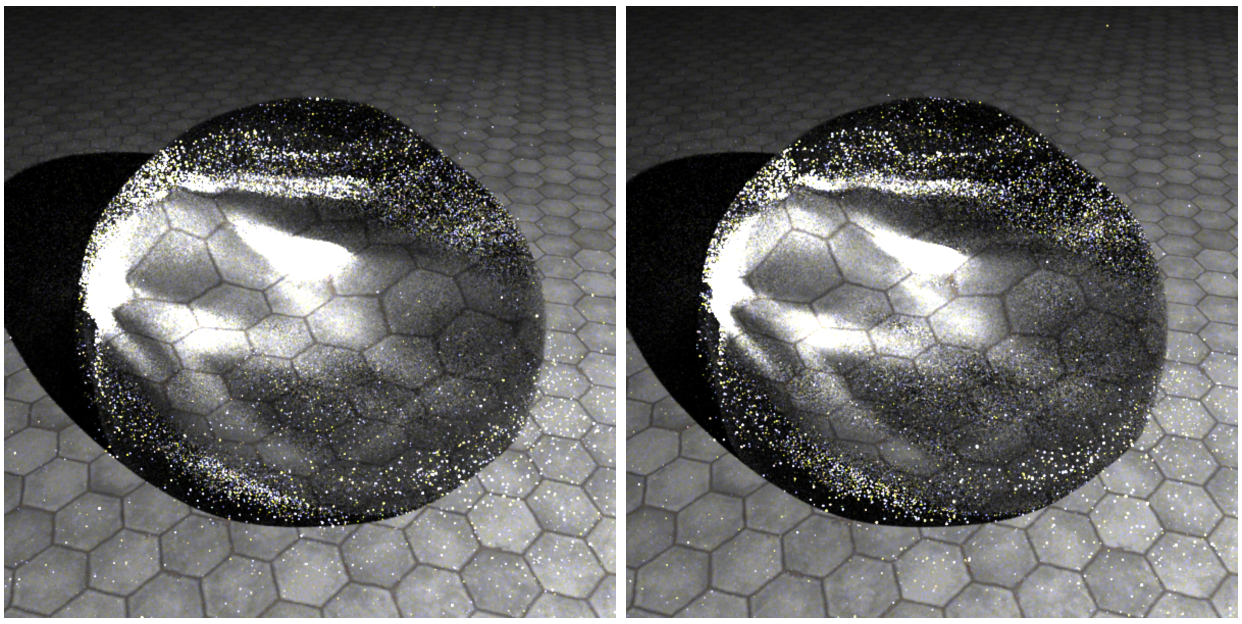
We then change the normal map and again perform single-bounce raytracing on a refractive fractal normal-mapped plane. Changing the normal map heavily changes the refractive pattern generated, as new paths to the lights must be calculated.



Scene 4: Normal-Mapped Reflective Plane

$L^{left}_{thres} = 5.0$, $L^{right}_{thres} = 5.0$

We perform single-bounce raytracing on a reflective gaussian normal-mapped plane for a [Left] gaussian normal-mapped plane and [Right] fractal normal-mapped plane. Similar to the previous refractive plane cases, changing the normal map changes the reflective pattern below. The right image took substantially longer (10x as long) than the left image due to the light paths being very challenging to find. It was in this image that the normal map was more extreme and had more normals pointing away from the light source, meaning they could not be used to find a path to the light. Even though we set a threshold of 100 tries per ray in both images, the left image on average could converge in 20 steps while the right image would exhaust most of the 100 tries with failure, extending the execution time substantially.



Scene 5: Refractive Sphere

$L^{left}_{thres} = 20.0$, $L^{right}_{thres} = 10.0$

We also perform single-bounce raytracing on a refractive sphere. The reason we are able to compute this surface in a single bounce is because we attempt to connect light paths from inside of the sphere where the diffuse region is to outside of the sphere, which only takes one refractive bounce.

The sphere is a good example of a mesh that has per-vertex normals. Thus, we can use Equation 46 to compute the derivative of the normal with respect to the barycentric coordinates we got during the first pass in order to measure the change in normal, or the curvature, $\frac{dn}{dv_0}$ of our geometry to get a more accurate gradient $\frac{dL}{dv_0}$ compared to just setting $\frac{dn}{dv_0} = 0$.

The sphere scene had the largest L_{thres} values because the scale of the scene was the largest. Recall if the geometry and lights in a scene are twice as big as another file, then its L_{thres} value scales twice as much. This is the case with Scene 5 compared to Scene 1 as shown in Figure 18.

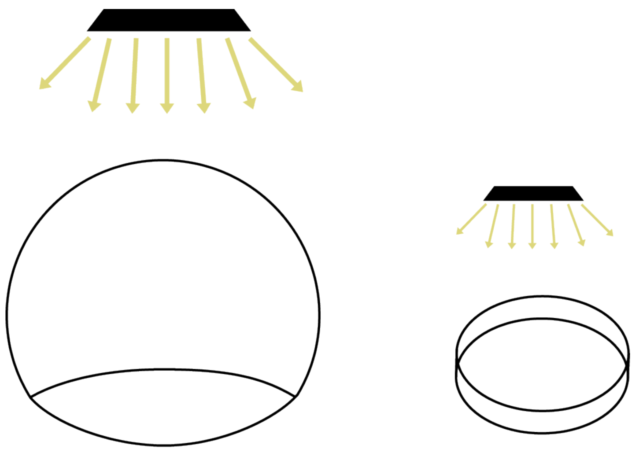


Figure 18: The loss threshold varies with the scene and light scales. larger lights correspond to larger loss thresholds.

We notice that there is a darker region around the edges of the sphere, which we assume may be due to one of two reasons. Our first assumption is that rays entering near the edges of the sphere may refract back out of the sphere before hitting the diffuse surface below, causing 2 refractive bounces which is not allowed by our single-bounce limit. We show this in the diagram below, where the left image illustrates a ray that refracts into the sphere and hits the diffuse surface, where we optimize for a single-bounce path refracting out of the sphere towards the light. The right image shows how towards the edges, rays are more likely to refract twice before hitting the diffuse surface, which means they need two refractive bounces to make it back to the light as shown in Figure 19. To fix this, we increase the maximum number of bounces, which in turn helps the samples partially along the edges of the image.

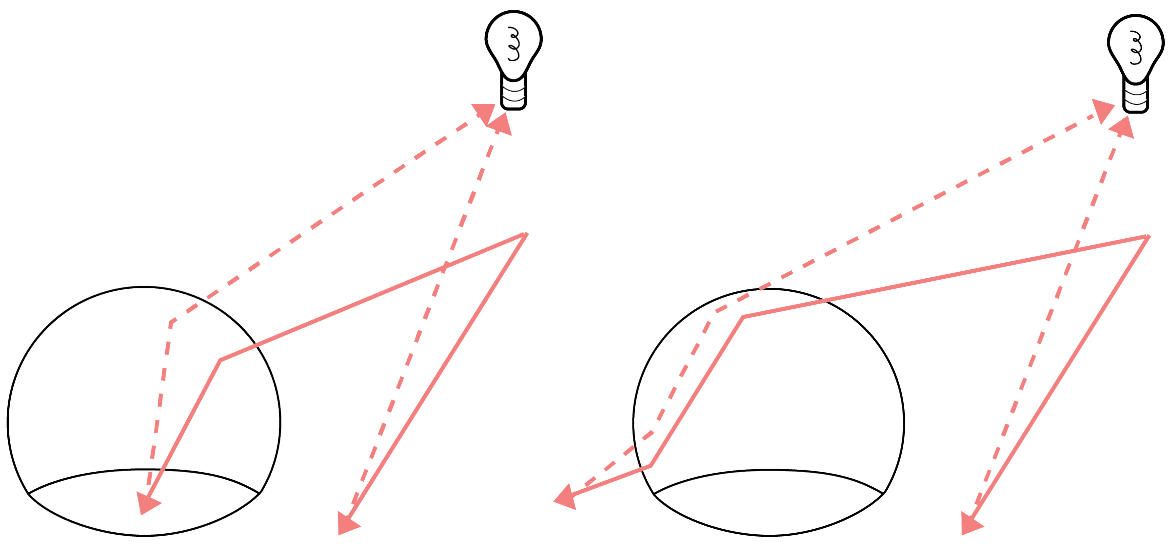


Figure 19: Rays require more than one refractive bounce along the edges, making it difficult to solve for in our single-path system.

Our second assumption is that the gradient around this area is very unstable. Recall that the value $\frac{dn}{dv_0}$ measure the curvature around some point x_i. The curvature for our sphere should be constant throughout, yet the curvature projected onto our camera is not. For example, the portions of the sphere that have normals facing the camera appear to have less curvature, than the normals perpendicular to the camera around the edges. Another way to say this is that there is more fall-off around the edges of the sphere, and so small perturbations due to our optimization are more extreme around the edges of the sphere. Thus, we need to account for the sensitivity of optimizing around regions where there is high fall-off.

A similar, but different problem is also when we change surfaces. For example, when trying to optimize the 0-bounce ray on the left, we may end up taking a step size into the refractive material that now requires 2 bounces, changing our gradient a lot more and leading to a non-continuous loss function. It becomes difficult to converge around the edges of materials for this reason as our algorithm would continue to jump back and forth between materials in Figure 20.

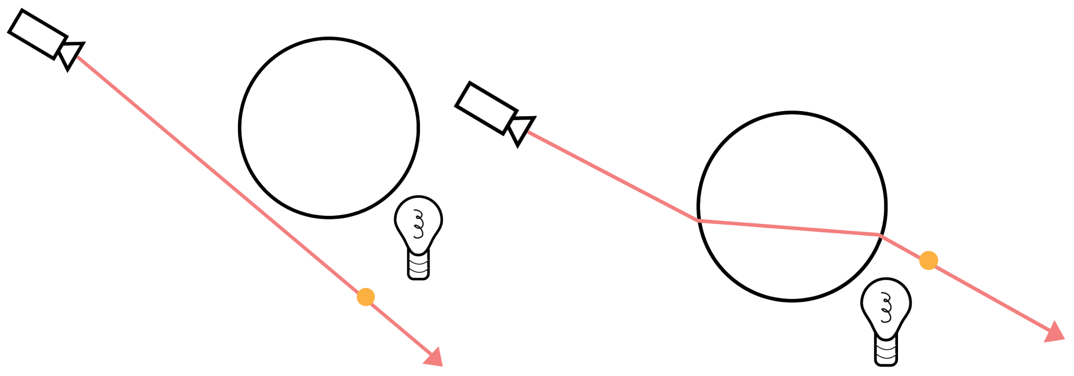
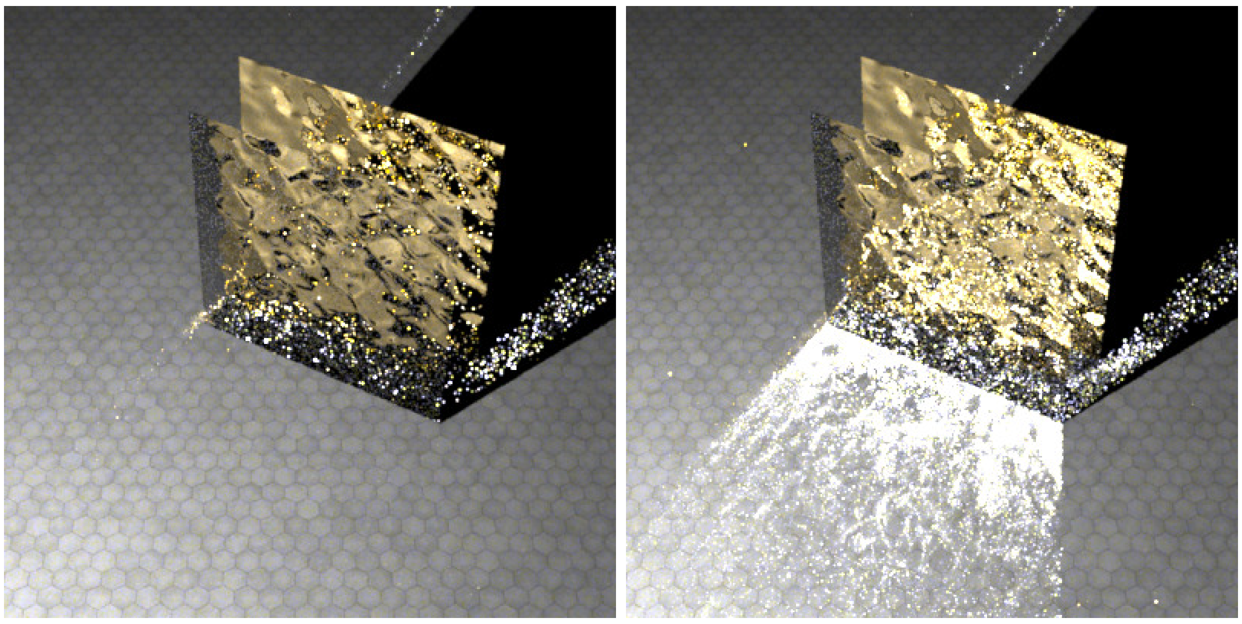


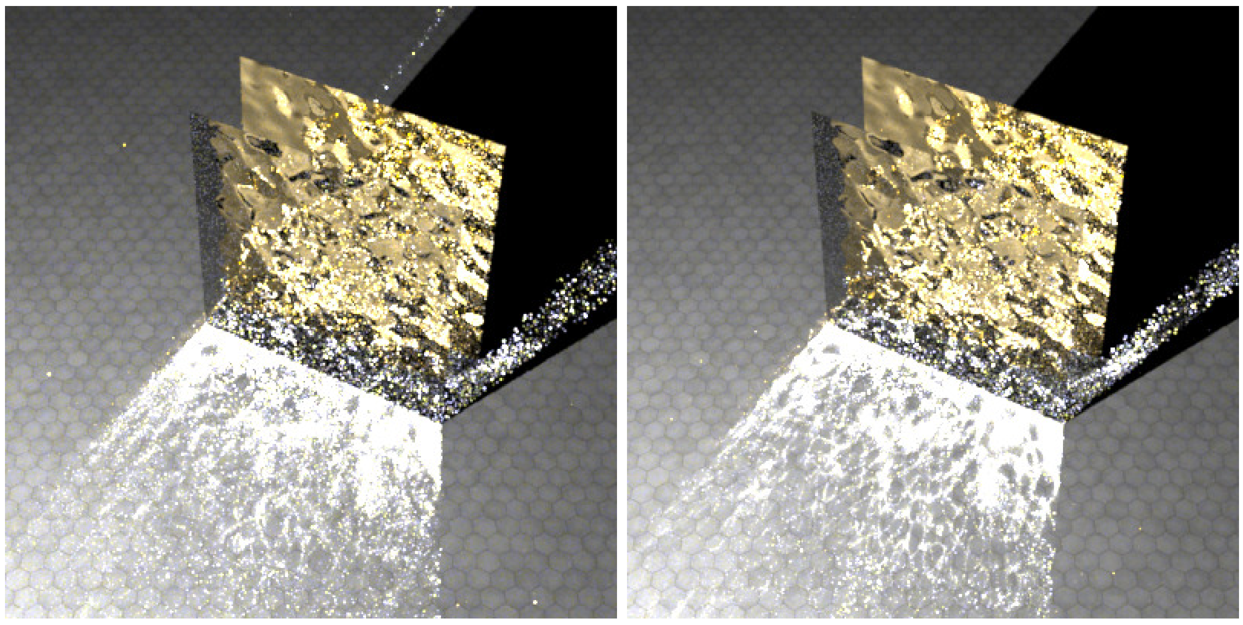
Figure 20: Changing between materials causes a large change in the loss function, leading to discontinuities in the gradient and future steps. This is a bigger problem for multi-bounce systems where longer ray paths have larger likelihoods of switching between geometry earlier in the path.

We found that a smaller step size for β works best for these scenarios, but is also leads to longer convergence times for other regions of the image where we are not changing between materials so frequently, for example the center of the sphere. This is why we half the step size β each time we change materials to help encourage smaller steps around edges. This method unfortunately does not work as well with future bounces, as even a small change in the ray direction will make for a large change in material intersections further into the ray's traversal, so we can only make this modification for the first ray bounce.

8.2 Multi-Bounce System


Scene 6: Reflective Refractive Plane

We introduce a gaussian-mapped refractive plane in front of a gaussian-mapped reflective plane and see that our single-bounce method on the left is not able to capture any light in front of the refractive surface while the 5-bounce system on the right is. This is because the paths starting from the diffuse surface in front of the refractive surface require a refractive bounce through the first plane, followed by a reflective bounce off the second plane and a final refractive bounce again through the first plane before hitting the light source, requiring at least 3 bounces. Our algorithm is easily able to adjust to multi-bounce scene files. By increasing the number of bounces during the forward pass, our computational graph automatically captures the new multi-bounce gradient $\frac{dL}{dv_0}$.



Scene 6: Reflective Refractive Plane

$L_{thres}^{left} = 5.0$, $L_{thres}^{right} = 0.5$

We can also see how adjusting the L_{thres} value adds additional light into the scene. On the left image with a higher L_{thres}, we can see an additional beam of light to the left of the reflective plane near the shadow while the right image does not have such a beam. By having a larger L_{thres}, we allow more rays to reach the light from more possible angles. This is also evident by looking at the light in front of the refractive surface, which is wider in the left image.

9 Future Work

9.1 Generalized Loss

Currently the termination criteria is whether some loss L is less than a given threshold L_{thres}. This does not generalize well to scenes of different scales, as the same scene scaled up by a factor of 2x will have a loss value L that is also 2x, making the convergence criteria even more difficult. This also allows lights to scale by a factor of 2x as well, giving more opportunity to set L_{thres} to be larger. To make the loss more scale-invariant, we can instead compare the normalized change in loss $\Delta L = \frac{L_t - L_{t-1}}{L_t}$ for iteration $t \in [0, k]$ and terminate when ΔL is less than our new termination threshold L'_{thres}. This allows us to set the same L'_{thres} value for multiple scenes without having to tune it as a hyperparameter.

We can extend this idea to also consider the scene's volume when determining other hyperparameters, such as optimization parameters β_{init} β_{scale}, and β_{max}. Larger scenes should have larger values, which we can scale by the bounding box volume of the scene to make them more adaptive to different scenes without having to hand-tune the variables each time we change scenes.

9.2 Screen-Space Curvature

Some areas of the sphere were more sensitive to gradient step updates, particularly in areas where the change in normal with respect to the pixels was large. We refer to this as screen-space curvature. To measure this value, we can take the dot product between the incoming camera ray and the normal to get the screen space curvature. We can then weigh our step size proportional to this value, where we make smaller steps in regions where there is larger screen space curvature.

9.3 Implicit Surfaces

We can attempt to extend our pipeline to support additional implicit surfaces such as cylinders and point clouds as well as neural networks. In order to be compatible with our approach, the surface would need a valid geometric function $G(x)$ that we could then twice differentiate to get the normal $N(x)$ and the curvature $C(x)$. We could then use the geometric function during ray marching to iteratively check for intersection with the surface.

The normal would be needed to compute reflective and refractive bounces during the forward pass while the curvature would be needed for the normal's derivative during the backwards pass. For more structured shapes such as cylinders we can simply take the derivative of the cylinder equation, while more advanced structures such as point clouds require PCA to extract normal and curvature information of a point relative to its neighbors.

9.4 Heterogeneous Refractive Material

It would be interesting to extend our pathtracer to handle refractive material where the index of refraction is a function of the current position in the material x_i. Previous work [18] examined and solved for these types of systems as an iterative update approach where we compute the current index of refraction and take a step forward. Extending our forward-pass pipeline to account for this would require us to modify the refraction class to support functions for the index of refraction, as well as an iterative ray tracer that steps through refractive materials rather than traces all the way through. In terms of the backwards pass, we could consider the general case that each step through the refractive material is simply a change into a new refractive material, effective treating each step as a new refractive bounce. This would extend the computation graph without us needing to extend the back-propagation or derivation stage in any way.

10 Bibliography

References

[1] 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. URL: https://doi.org/10.1145/1468075.1468082.

[2] 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. URL: https://doi.org/10.1145/358876.358882.

[3] Scott D Roth. “Ray casting for modeling solids”. In: *Computer Graphics and Image Processing* 18.2 (1982), pp. 109–144. ISSN: 0146-664X. DOI: https://doi.org/10.1016/0146-664X(82)90169-1. URL: https://www.sciencedirect.com/science/article/pii/0146664X82901691.

[4] K. Perlin and E. M. Hoffert. “Hypertexture”. In: *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH ’89. New York, NY, USA: Association for Computing Machinery, 1989, pp. 253–262. ISBN: 0897913124. DOI: 10.1145/74333.74359. URL: https://doi.org/10.1145/74333.74359.

[5] Don Mitchell and Pat Hanrahan. “Illumination from Curved Reflectors”. In: *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH ’92. New York, NY, USA: Association for Computing Machinery, 1992, pp. 283–291. ISBN: 0897914791. DOI: 10.1145/133994.134082. URL: https://doi.org/10.1145/133994.134082.

[6] Eric Lafortune. *Mathematical Models and Monte Carlo Algorithms for Physically Based Rendering.* Tech. rep. 1996.

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

[8] Min Chen and James Arvo. “Theory and Application of Specular Path Perturbation”. In: *ACM Trans. Graph.* 19.4 (Oct. 2000), pp. 246–278. ISSN: 0730-0301. DOI: 10.1145/380666.380670. URL: https://doi.org/10.1145/380666.380670.

[9] T. Chan and Wei Zhu. “Level set based shape prior segmentation”. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05).* Vol. 2. 2005, 1164–1170 vol. 2. DOI: 10.1109/CVPR.2005.212.

[10] Bram de Greve. "Reflections and Refractions in Ray Tracing". In: (2006).

[11] Bruce Walter et al. "Single Scattering in Refractive Media with Triangle Mesh Boundaries". In: *ACM SIGGRAPH 2009 Papers*. SIGGRAPH '09. New Orleans, Louisiana: Association for Computing Machinery, 2009. ISBN: 9781605587264. DOI: 10.1145/1576246.1531398. URL: https://doi.org/10.1145/1576246.1531398.

[12] 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. URL: https://doi.org/10.1145/2185520.2185554.

[13] Amrita Mazumdar. "Principles and Techniques of Schlieren Imaging Systems". In: 2013.

[14] Diederik Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations* (Dec. 2014).

[15] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *ArXiv* abs/1609.04747 (2016).

[16] Merlin Nimier-David et al. "Mitsuba 2: A Retargetable Forward and Inverse Renderer". In: *ACM Trans. Graph.* 38.6 (Nov. 2019). ISSN: 0730-0301. DOI: 10.1145/3355089.3356498. URL: https://doi.org/10.1145/3355089.3356498.

[17] Arpit Agarwal, Tim Man, and Wenzhen Yuan. *Simulation of Vision-based Tactile Sensors using Physics based Rendering*. 2020. DOI: 10.48550/ARXIV.2012.13184. URL: https://arxiv.org/abs/2012.13184.

[18] Adithya Pediredla et al. "Path Tracing Estimators for Refractive Radiative Transfer". In: *ACM Trans. Graph.* 39.6 (Nov. 2020). ISSN: 0730-0301. DOI: 10.1145/3414685.3417793. URL: https://doi.org/10.1145/3414685.3417793.

[19] Tizian Zeltner, Iliyan Georgiev, and Wenzel Jakob. "Specular Manifold Sampling for Rendering High-Frequency Caustics and Glints". In: *ACM Trans. Graph.* 39.4 (July 2020). ISSN: 0730-0301. DOI: 10.1145/3386569.3392408. URL: https://doi.org/10.1145/3386569.3392408.

[20] Ethan Tseng et al. "Differentiable Compound Optics and Processing Pipeline Optimization for End-to-end Camera Design". In: *ACM Transactions on Graphics (TOG)* 40.2 (2021).