Computing Manifold Next-Event Estimation without Derivatives using the Nelder-Mead Method

Ana Granizo-Hidalgo, Nicolas Holzschuch

Eurographics Symposium on Rendering 2024
E Garces and E. Haines
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 43 (2024), Number 4

Computing Manifold Next-Event Estimation without Derivatives using the Nelder-Mead Method

Ana Granizo-Hidalgo and Nicolas Holzschuch

Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LJK

Figure 1: *Our algorithm, running on three different test scenes. It computes complex caustics, with the same quality as the state of the art.*

Abstract

Specular surfaces, by focusing the light that is being reflected or refracted, cause bright spots in the scene, called caustics*. These caustics are challenging to compute for global illumination algorithms. Manifold-based methods (Manifold Exploration, Manifold Next-Event Estimation, Specular Next Event Estimation) compute these caustics as the zeros of an objective function, using the Newton-Raphson method. They are efficient, but require computing the derivatives of the objective function, which in turn requires local surface derivatives around the reflection point, which can be challenging to implement. In this paper, we leverage the Nelder-Mead method to compute caustics using Manifold Next-Event Estimation* without *having to compute local derivatives. Our method only requires local evaluations of the objective function, making it an easy addition to any path-tracing algorithm.*

CCS Concepts

• ***Computing methodologies*** → ***Ray tracing;***

1. Introduction

By focusing light, specular surfaces in a scene cause bright spots, called *caustics*. These caustics play an important role in our perception of the scene and in photorealistic simulation of light transport, but they are challenging to compute. They correspond to high-intensity radiance transfer over a small surface area, where a specific path connects to the light source through specular reflections or refractions.

This makes caustics difficult to compute: they are caused by a small set of light paths, which have a null or small measure in path space; randomly sampling path space, for example using naive Monte-Carlo methods, will likely miss the paths responsible for the caustics, or under-sample them.

Several global illumination algorithms have been developed to efficiently compute caustics; Manifold Next-Event Estimation algorithms [HDF15] are efficient methods to compute caustics caused by direct reflection or refraction on a specular surface. They compute the light paths connecting a receiver point to the light source through this specular interaction by computing the zeroes of an objective function. This computation uses Newton-Raphson method and requires the derivatives of the objective function. To compute these, we need the local surface derivatives which can be difficult to implement, especially in an existing code base.

In this paper, we present a new approach for Manifold Next-Event Estimation, computing the zeros of the objective function using the Nelder-Mead method [NM65] instead of Newton-Raphson. Our method only requires local computation of the objective function, not of the derivatives, making it easier to combine with existing global illumination implementations. Nelder-Mead method is less efficient than Newton-Raphson; it requires more computation steps to converge. As a result, our algorithm is two times slower than Specular Manifold Sampling [ZGJ20]. However, it has potential in situations where the simplicity and compactness of the code base would be more important than raw speed.

In the next section, we review previous algorithms for computing caustics in global illumination. We then detail the basics of the Nelder-Mead method in Section 3. In Section 4, we present our algorithm for solving Manifold Next-Event Estimation. In Section 5, we present results and compare our algorithm with previous works, then we discuss limitations in Section 6. We conclude and present potential avenues for future work in Section 7.

2. Previous Works

Most recent works on global illumination solve the rendering equation [Kaj86] using random sampling of the set of light paths connecting the camera to the light source, via Monte-Carlo sampling. Within this framework, caustics are a difficult case: they represent a small subspace of the space of light paths, so randomly sampling the space is very likely to miss this subset.

Jensen [Jen96] introduced Photon Mapping, a two-step generic method to compute global illumination, including caustics: in a first step, photons are scattered through the scene. In the second step, they are gathered to reconstruct illumination effects. Photon mapping is very versatile: it can render almost all illumination effects. Photon mapping separates caustics from other illumination effects, allocating a separate set of photons for them. The quality of caustics reconstruction depends on the number of caustics photons.

Metropolis Light Transport [VG97] explores the path of light spaces using random mutations. The algorithm is good at finding difficult paths connecting the camera and the light source, but still has difficulties for reflections on pure specular surfaces.

Mitchell and Hanranhan [MH92] computed reflective caustics on curved surfaces using the surface derivatives. They identify reflected paths as paths with extremal length, based on Fermat's principle, and use Newton-Raphson's method to find these paths.

Walter et al. [WZHB09] compute refracted paths inside surfaces defined by triangles with interpolated normals. They identify refracted paths as paths through points where the half-vector is equal to the opposite of the normal, and search for these points using Newton-Raphson's method.

Jakob and Marschner [JM12] extend Metropolis Light Transport [VG97] for reflection on specular surfaces, by restricting the space of perturbations to a manifold, which can be explored systematically. It is very efficient to find specular paths into the scene, but not so efficient when we need to connect the last point of a path to the light source.

Practical path guiding [MGN17] gradually learns the incoming radiance distribution using an adaptive spatio-directional hybrid data structure, resulting in stable performance in scenes with difficult geometry, including caustics. This performance stability is a strong advantage of the method, making it compatible with production environments.

Classical Path-Tracing usually relies on Next-Event Estimation: computing direct illumination by connecting the current point to the light source using a shadow ray. Next-Event Estimation dramatically improves convergence speed, but it is not available when there is a refractive surface on the way. Hanika et al. [HDF15] combine Next-Event Estimation and Manifold exploration into *Manifold Next-Event Estimation*: they use manifold exploration to find the refracted path through the surface, connecting the light source with the last point on the path.

Zeltner et al. [ZGJ20] improve this technique with *Specular Manifold Sampling*, using a different objective function and systematic search for multiple solutions. Loubet et al. [LZHJ20] used slope-space integrals for fast computation of specular next-event estimation.

Our work uses a similar approach as [HDF15, ZGJ20], but we do not compute derivatives in our search for solutions, using instead Nelder-Mead algorithm [NM65].

3. The Nelder-Mead algorithm

3.1. Principles

The Nelder-Mead algorithm [NM65] is an algorithm to find a local minimum of a function, like gradient descent. Unlike gradient descent, it does not compute the local derivatives, but only the value of the function at certain sample points. At each step, it maintains a *simplex* of sample points, and either extends or contracts the simplex based on computations.

For a function defined over a parameter space of dimension d, the Nelder-Mead algorithm maintains a simplex of $d+1$ sample points. In our case, we are interested in a function defined on a surface, of dimension 2, so we maintain a triangle of sample points.

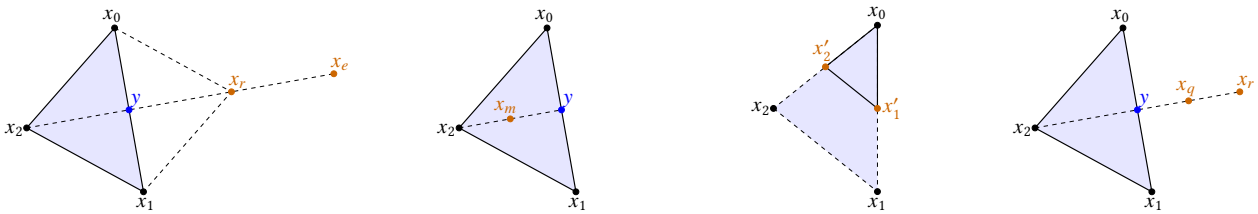
At each step, the sample point with the largest value is replaced by another point, with a smaller value, ensuring that the average value for all points on the simplex is decreasing.

3.2. Algorithm

In this section, we are working with an objective function f defined over a space of dimension d. We maintain a set of sample points $\{x_0, x_1, ..., x_d\}$, ordered by the value of the function:

$$f(x_0) \leq f(x_1) \leq ... \leq f(x_d) \tag{1}$$

At each iteration, our goal is to replace x_d (the point with the highest value) with another sample point, with a lower value for f. We start by computing y, the centroid of the remaining points, $\{x_0, x_1, ..., x_{d-1}\}$. We then compute x_r, the symmetric of x_d around y, and the value of the function at x_r, $f(x_r)$ (see Figure 2a). The algorithm then depends on the value of $f(x_r)$ compared to the other values of f already computed:



(a) *Reflect and Expand* **(b)** *Contract* **(c)** *Shrink* **(d)** *Middle expand*

Figure 2: *The different cases in the Nelder-Mead method.*

- If $f(x_r)$ is the best value so far ($f(x_r) < f(x_0)$), then we try going even further, computing x_e as an expanded version of x_r (see Figure 2a):

$$x_e = y + \overrightarrow{yx_r} \tag{2}$$

  We then replace x_d by the point with the smallest value between x_r and x_e: If $f(x_e) \leq f(x_r)$, then we replace x_d with x_e. Otherwise, we replace x_d with x_r.
- If $f(x_r)$ is the worst value so far ($f(x_r) > f(x_d)$), then we try contracting the triangle by computing x_m, the middle of $[yx_d]$ (see Figure 2b). If that point is better than x_r ($f(x_m) < f(x_r)$), then we use it as a replacement for x_d. Otherwise, we shrink the entire simplex around x_0, replacing all points x_i except x_0 by the middle of the segment $[x_i x_0]$ (see Figure 2c).
- If x_r is better than the second worst point, x_{d-1}, but not better than the best point x_0 ($f(x_0) \leq f(x_r) \leq f(x_{d-1})$), then we replace x_d with x_r.
- If x_r is between the second worst point and the worst point ($f(x_{d-1}) \leq f(x_r) \leq f(x_d)$), then try expanding the triangle a little less, by computing x_q, the middle of $[yx_r]$ (see Figure 2d). If x_q is better than x_r then we replace x_d by x_q, otherwise we shrink the entire simplex around x_0, replacing all points x_i except x_0 by the middle of the segment $[x_i x_0]$ (see Figure 2c).

At the end of an iteration, we have either replaced x_d with a new sample point that has a lower value for the objective function or shrunk the entire simplex around the point with the lowest value. We then re-order our $d+1$ sample points and compute the next iteration.

3.3. Analysis

Most of the time, Nelder-Mead replaces the sample point with the largest value with one sample point with a lower value, so the average error over the simplex decreases. In some cases, the simplex contracts towards the sample point with the lowest value. The latter is not guaranteed to improve the objective, but heuristically it does.

Each iteration requires between 1 and $d+2$ computations of f. The best-case scenario (1 computation of f) happens when we replace the worst point x_d with the reflected point x_r. The worst-case scenario happens when we have to shrink the simplex and recompute values of f at all the new points.

In low dimensions, for example in our case with $d = 2$, the worst case is more expensive than computing the gradient of f. The main interest of Nelder-Mead is the ease of implementation, as it does not require implementing the data structures required to compute the local derivatives of the path.

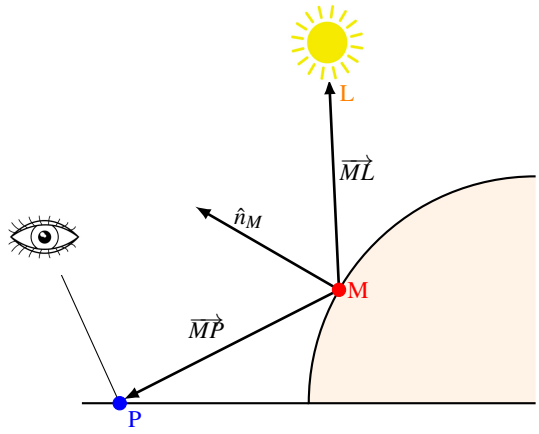
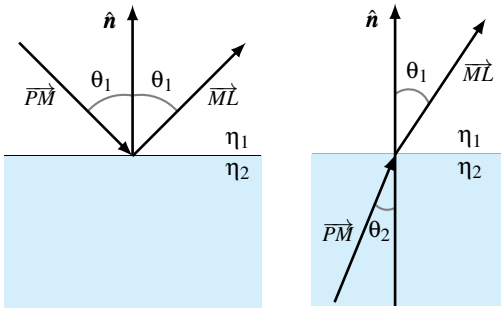


Figure 3: *A specular path through a reflective interface.*



(a) *Reflection* **(b)** *Refraction*

Figure 4: *Snell-Descartes law.*

4. Our algorithm

4.1. Problem position

We place ourselves with the same setting as Manifold Next-Event Estimation [HDF15] and Specular Manifold Sampling [ZGJ20]: we have computed a path from the camera to a diffuse or glossy receiver, and we want to connect the last point on this light path to the light source (see Figure 3). Direct connections to the light source

are usually called *Next-Event Estimation*. Connecting the last point to the light source through one interaction on a specular surface, either reflection or refraction, is *Manifold Next-Event Estimation*.

We assume that our scene contains surfaces with varying normals, either triangles with interpolated normals or surfaces with normal maps.

A specular path connecting the last point on the light path P to the light source L, through a point on the specular surface M, must obey the law of Snell-Descartes:

- The two vectors $\overrightarrow{MP}$, $\overrightarrow{ML}$ and the normal at M, $\hat{\boldsymbol{n}}_M$ must be coplanar.
- For reflection, $\overrightarrow{MP}$ and $\overrightarrow{ML}$ must be symmetric around the normal $\boldsymbol{n}_M$ (see Figure 4a).
- For refraction, the sines of the angles between $\overrightarrow{MP}$ or $\overrightarrow{ML}$ and the normal $\boldsymbol{n}_M$ are connected with the index of refraction in each material η_i (see Figure 4b):

$$\eta_1 \sin\theta_1 = \eta_2 \sin\theta_2 \tag{3}$$

We express these conditions using an objective function f, based on the half-vector $\hat{\boldsymbol{h}}$, as in [WZHB09, HDF15]. For specular reflection, the half-vector is:

$$\hat{\omega}_P = \frac{\overrightarrow{MP}}{\|\overrightarrow{MP}\|} \tag{4}$$

$$\hat{\omega}_L = \frac{\overrightarrow{ML}}{\|\overrightarrow{ML}\|} \tag{5}$$

$$\hat{\boldsymbol{h}} = \frac{\hat{\omega}_P + \hat{\omega}_L}{\|\hat{\omega}_P + \hat{\omega}_L\|} \tag{6}$$

For specular refraction:

$$\hat{\boldsymbol{h}} = -\frac{\eta_2\hat{\omega}_P + \eta_1\hat{\omega}_L}{\|\eta_2\hat{\omega}_P + \eta_1\hat{\omega}_L\|} \tag{7}$$

The objective function in both cases is:

$$f(M) = norm(\hat{\boldsymbol{h}} - \hat{\boldsymbol{n}}_M) \tag{8}$$

In both cases, we used the convention that the normal $\hat{\boldsymbol{n}}_M$ is pointing towards outside the object. The specular paths we are looking for correspond to zeroes of these objective functions f.

4.2. Algorithm description

Our algorithm follows the Nelder-Mead method:

- First we select a starting simplex, a set of three points where we compute the value of the objective function (see Section 4.3).
- Then we recursively refine this simplex using Nelder-Mead method (see Section 4.4).
- We stop the search when the value of the objective function falls below a certain threshold, when the size of the triangle is too small or after a preset number of iterations (see Section 4.5).
- For faster convergence, we try to reject early paths where the search will not converge (see Section 4.7).
- Following Zeltner et al. [ZGJ20], we do this search with multiple starting points on the surface, as there can be several specular paths connecting the receiving point to the light source through a specular interaction.

The main differences with the original Nelder-Mead method described come from the fact that our simplexes are mapped onto a curved surface instead of a flat plane, so we need to reproject each point back to the surface, using ray-casting.

4.3. Selection of the starting simplex

For the Nelder-Mead method, we need three starting points instead of a single starting point. We select three random points on the surface and form our starting simplex with these. From our experiments, we determined that using three random points on the surface is the most efficient initialization method; even if the points are too far apart, the method converges to a solution. We tried other methods, such as using the vertices of a random triangle in the surface tesselation, but found this to be less efficient.

4.4. Searching for the minimum

We apply the basic Nelder-Mead algorithm, as detailed in Section 3, with one key difference: Nelder-Mead assumes we have a global parametric space, and our surfaces are not parameterized. Whenever we compute a new point for the method, we first compute this point in the plane defined by the current simplex, then reproject it on the surface using ray-casting.

After each refinement step, we have a simplex of three points lying on the surface.

4.5. Stopping criteria

We stop the refinement when either of the following criteria is met:

- the value of the objective function at the vertices of the simplex is below a certain threshold,
- or the length of the largest edge of the simplex is smaller than another threshold,
- or if we have reached a predefined maximum number of iterations.

In each case, we compute the center of gravity of the simplex, then reproject it on the surface and compute the value of the objective function. If it is smaller than our threshold, we consider that this point makes a valid specular path. We then check its visibility from the light source using a shadow ray and connect it to the light source if it is visible, creating a complete light path from the camera to the light source.

4.6. Contribution of a path

Once we have found the path connecting point P to the light, we need to calculate the contribution of this path. For this, we use the same approach as Specular Manifold Sampling [ZGJ20] in their unbiased version. First, we evaluate the path contribution by computing the reflectance at the specular point $fs(M)$, the generalized geometry factor $G(P \leftrightarrow M \leftrightarrow L)$, and the emitter weight w_L.

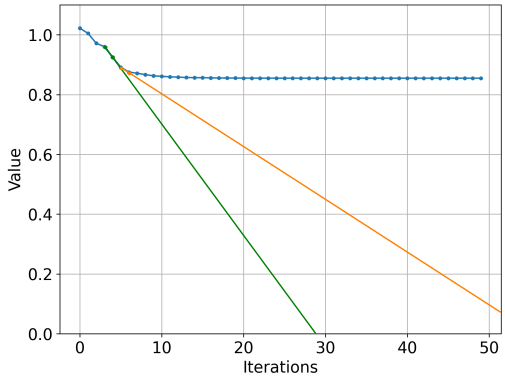


Figure 5: *Average values of the objective function at each iteration, in a case when it does not converge to 0. In green: extrapolation between the average at iterations 3 and 4; at this rate, the algorithm could get to 0 before the 30th iteration. In orange: extrapolation between the average at iterations 5 and 6, at this rate we need more than 50 iterations to get to 0, so we cancel the search.*

$$S = fs(M) \cdot G(P \leftrightarrow M \leftrightarrow L) \cdot w_L \tag{9}$$

where $fs(M)$ is computed using the Fresnel term and the solid angle compression. The generalized geometry factor [JM12] uses the derivatives at each point of the path to relate different solid angles at both ends of the path.

Once we have the specular contribution, we need to account for the probability of sampling that path. This is estimated separately by repeatedly calling the function to find a specular path and counting the number of trials required to sample the same path again. The final contribution is given by:

$$contribution = S \cdot \langle 1/p_k \rangle \tag{10}$$

where $\langle 1/p_k \rangle$ is the estimated inverse probability of sampling point M.

4.7. Preliminary rejection

We are searching for zeros of the objective function. The Nelder-Mead algorithm converges to a local minimum, which might not be a zero of the function. We detect whether the algorithm is converging to a non-zero local minimum by looking at the slope of the error function, and stop the refinement in those cases.

After each refinement step, we compute the average error for the current simplex by averaging the values of the objective function at the vertices. This average error is a monotonous decreasing function by design of the Nelder-Mead algorithm. At each step, we extrapolate the error curve using the newly computed average error and the average error computed at the previous step (see Figure 5). We stop the refinement early if the extrapolated error does not reach zero within the maximum number of iterations.

As the error function can behave erratically over the first iterations, we only apply this criterion after 20 iterations.

5. Results

We implemented our algorithm on the CPU inside Mitsuba 2 [NDVZJ19]. Unless otherwise specified, all the numbers and figures in this paper are computed on an Intel Xeon with 2.40 GHz, 32 cores.

We ran our algorithm on three different test scenes (see Figure 1): a specular bumpy sphere, a specular Stanford bunny, and a scene with two specular vases. We use the following hyperparameters, that are the same for all our test scenes: 100k trials to compute the probability of each path, 50 iterations for the Nelder-Mead method, 1e-4 as the threshold for the objective function, and 0.1 as the threshold for the simplex edges.

5.1. Algorithm behaviour

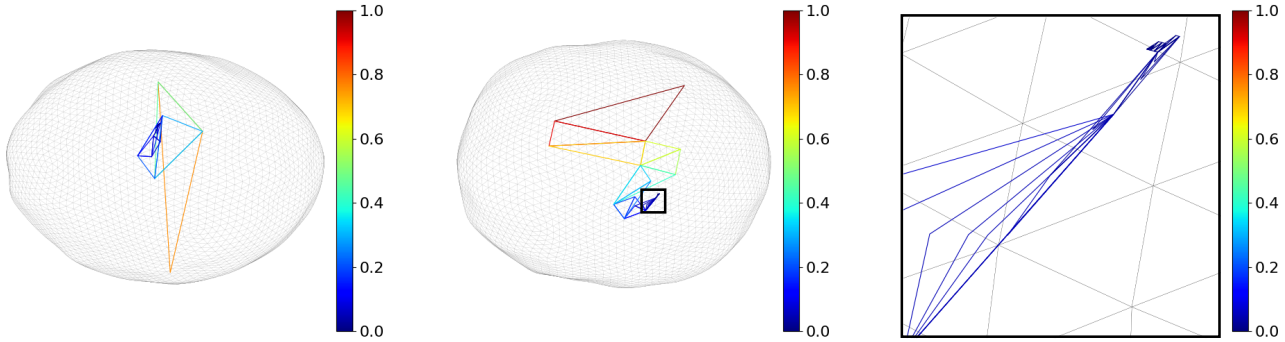
Figure 6 shows the successive simplex computed by the method during two specific searches. The different steps in the search appear clearly: first the search area moves across the surface, then the simplex contracts around the solution.

Figure 7 shows the average value of the objective function for the vertices of the simplex, computed at each iteration for two different searches (one that is successful, the other that is not). This average value decreases at each step: the algorithm is converging towards a local minimum.

5.2. Validation: comparison with Specular Manifold Sampling

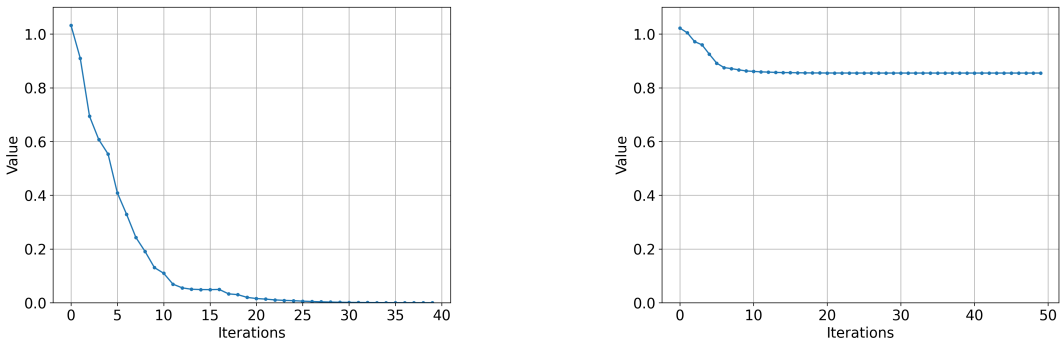
Figure 8, shows a comparison between our algorithm and Specular Manifold Sampling (SMS), the unbiased version, [ZGJ20] on all our test scenes, with the same number of samples per pixel (spp) at each scene. Our algorithm is slower for the same number of samples per pixel but reaches similar quality levels. In addition to the visual comparison, we calculate the structural similarity index measure (SSIM) between each image and a reference image. The structural similarity indicates the resemblance between two images, considering the degradation of the image as a perceived change in structural information. The result of this measurement provides us with an index that represents how similar the images are, as well as a new image on a greyscale, in which white means that those parts of the images are similar, and black means they are different. In the bumpy sphere scene in Figure 8, we can see that our method has trouble finding some of the paths that are in the border between the sphere and the ground, which is clearly represented in the structural similarity image. This happens because we always check the visibility between each new simplex point and either the light or the diffuse point. When we are close to the edge, the Nelder-Mead method can take us to a point on the other side of the floor, which prevents us from reaching that point and discarding the sample. On the other hand, our method can find more paths on the ground than SMS, which is also visible in the structural similarity image in all the test scenes. Both algorithms reach a comparable percentage of image similarity.

Table 1 shows, for each test scene, the computation time using



(a) **(b)** **(c)**

Figure 6: *Trajectory of our search function. (a) and (b) show the successive values of the simplex for two different searches, and (c) shows a zoom in of (b). The color of the triangles represents the value of the average of the objective function over the vertices.*



(a) *Case when the value converges to 0. (Found a valid path).*

(b) *Case when the value does not converge to 0.*

Figure 7: *Average value of the objective function at each iteration for two different searches.*

	Ours		SMS	
	time	%	time	%
Bumpy sphere 8spp	18.94 s	57.51	8.79 s	47.15
Vases 32spp	10.15 mn	23.56	4.25 mn	20.06
Stanford bunny 64spp	18.14 mn	29.31	8.32 mn	27.22

Table 1: *Comparison of computation time and percentage of successful searches between our method and Specular Manifold Sampling.*

our algorithm and the percentage of searches initiated that were successful. It also shows the same number for Specular Manifold Sampling, for comparison. With the same number of samples per pixel, our algorithm runs two times slower than Specular Manifold Sampling. The search algorithm converges slightly more often.

5.3. Comparison with a Newton-Raphson method that uses gradients computed using Finite Differences

Finite differences is a way of approximating the derivatives of the objective function, which is slightly simpler than actually computing the derivatives. We compared our method with an implementation that uses finite differences in the same setup. Figure 9 shows this comparison, where we can see that this method performs up to 3 times faster than ours, but does not manage to find all the solutions.

5.4. Comparison with Path Guiding and Photon Mapping

We also compared our algorithm with Photon Mapping [Jen96] and Practical Path Guiding [MGN17], both implemented inside Mitsuba 0.5 [Jak10]. All timings in this section are computed on a 2 GHz Quad-Core Intel Core i5 MacBook Pro with 32 GB of memory. Figure 10 shows the visual comparison of our method with Practical Path Guiding and Photon Mapping. Both scenes use similar computation times for all three methods (95 s for the Bumpy Sphere scene and 57 mn for the Vases scene).

For these scenes, where sharp reflective caustics are prominent, our algorithm produces better-quality images than Practical Path Guiding with the same time budget, as Photon Mapping achieves smoother images.

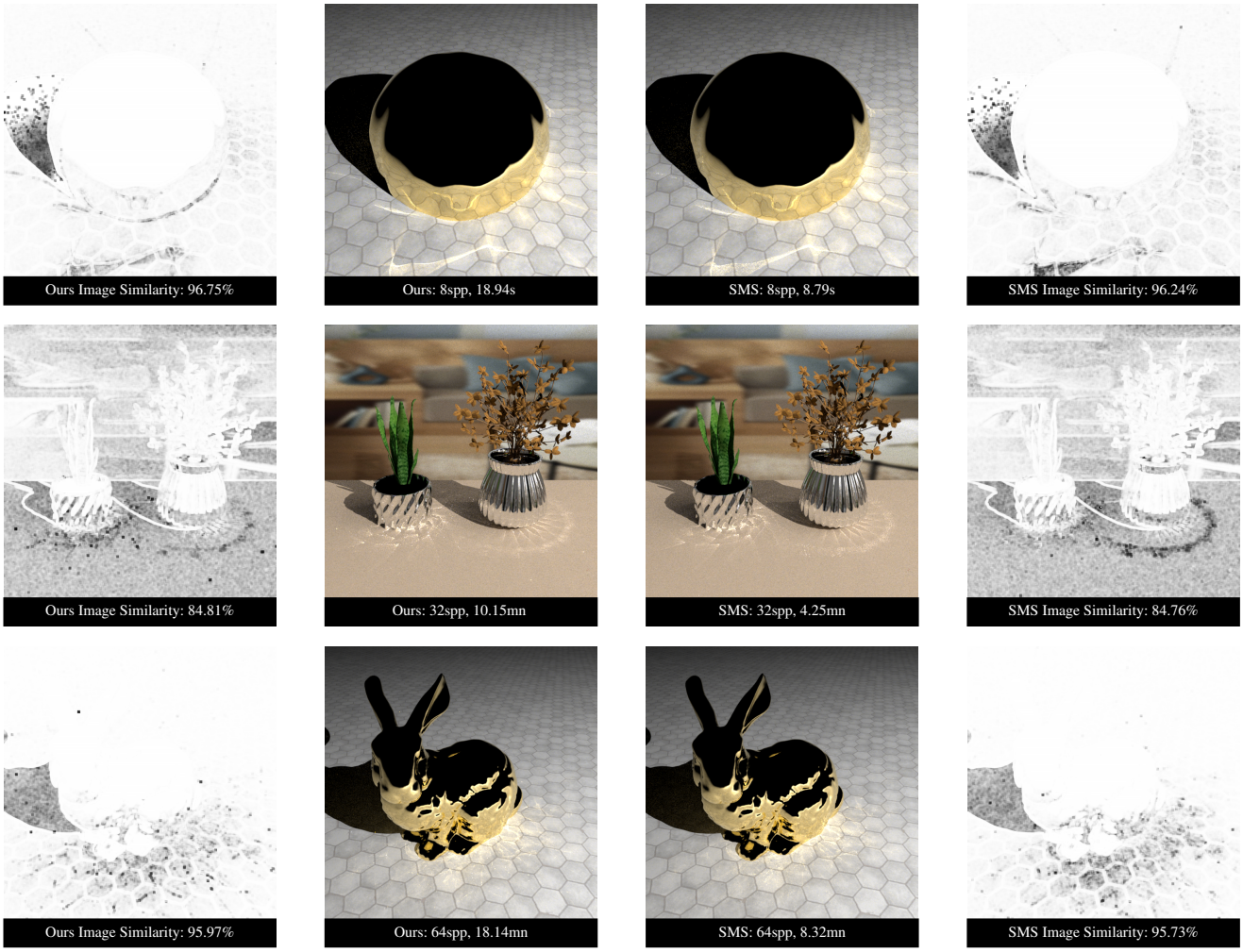


Figure 8: *Comparison between our algorithm and Specular Manifold Sampling. On a bumpy sphere (top), using 8 samples per pixel; on a scene containing 2 specular vases (middle), with 32 samples per pixel; and on a Stanford bunny scene (bottom), using 64 samples per pixel. We can see that both algorithms present similar visual results. On the leftmost and rightmost columns, the image similarity (SSIM) shows us the difference between each image and a reference image. The darker the region, the more difference there is with the reference image.*

6. Limitations and discussion

Our method is slower than Specular Manifold Sampling, and runs at comparable speed with Photon Mapping. Its key advantage, more than speed, is the ease of implementation: it only requires sampling the objective function at individual points on the surface; while computing the derivatives of the objective function requires having a local parameterization and local derivatives.

Our method is currently too slow to allow interactive rendering on the GPU, especially for specular reflections. Complicated caustics such as those in Figure 1 have several paths connecting each point to the light source. To compute all of these paths, we need to try using multiple random starting points, which impacts the total computation time. It could be possible to run interactively with one sample per pixel, but the results are of very low quality.

7. Conclusion and Future Works

We have presented an implementation of Manifold Next-Event Estimation that does not require computing the derivatives of the objective function. Our method uses the Nelder-Mead method to find the specular paths connecting receiver points to the light source. Even though it does not use the derivatives, it has a similar performance compared to Specular Manifold Sampling. This method can be extended to refractive caustics and multiple specular interfaces.

In future works, we want to extend this derivative-free approach to finding minima for other problems, such as inverse rendering.

References

[HDF15] HANIKA J., DROSKE M., FASCIONE L.: Manifold Next Event Estimation. *Computer Graphics Forum 34*, 4 (July 2015), 87–97. doi:

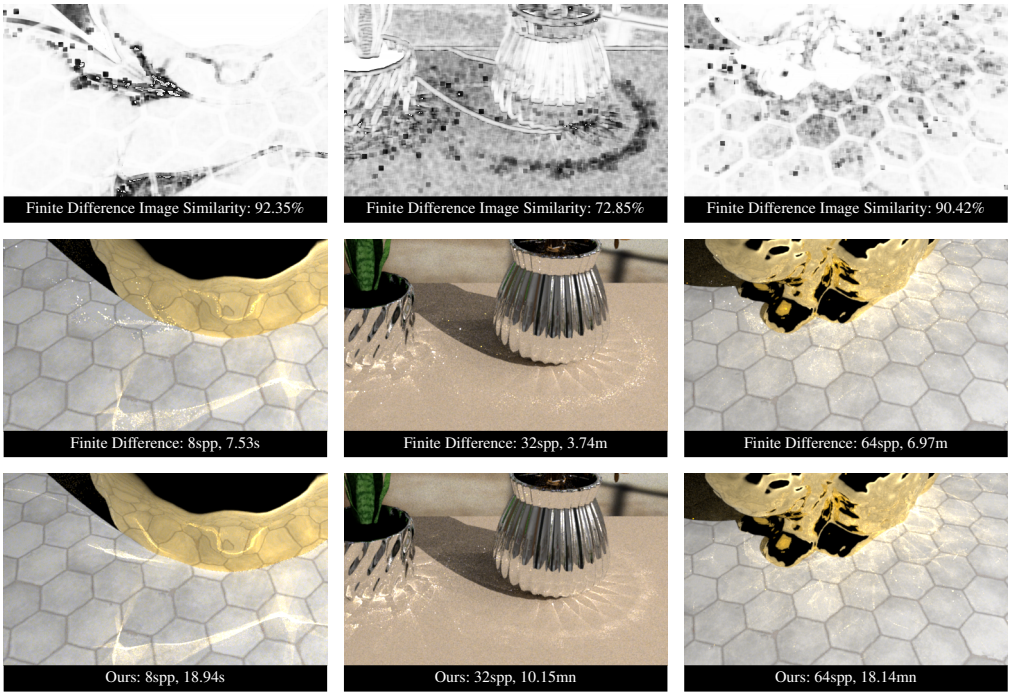


Figure 9: *Comparison between our method and Finite Differences on our three test scenes, with the same number of samples per pixel (spp) at each scene. The top row contains the image similarity (SSIM) between the Finite Difference method and a reference image. For the bumpy sphere and the vase scene, Finite Difference is not able to find all of the paths, while with the bunny scene, the missing paths are not that prominent, but the image is still slightly noisier.*

10.1111/cgf.12681. 1, 2, 3, 4

[Jak10] JAKOB W.: Mitsuba renderer, 2010. http://www.mitsuba-renderer.org. 6

[Jen96] JENSEN H. W.: Global Illumination using Photon Maps. In *Rendering Techniques '96*, Pueyo X., Schröder P., (Eds.). Springer Vienna, Vienna, 1996, pp. 21–30. doi:10.1007/978-3-7091-7484-5_3. 2, 6

[JM12] JAKOB W., MARSCHNER S.: Manifold exploration: a Markov Chain Monte Carlo technique for rendering scenes with difficult specular transport. *ACM Transactions on Graphics 31*, 4 (Aug. 2012), 1–13. doi:10.1145/2185520.2185554. 2, 5

[Kaj86] KAJIYA J. T.: The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques - SIGGRAPH '86* (1986), ACM Press. doi:10.1145/15922.15902. 2

[LZHJ20] LOUBET G., ZELTNER T., HOLZSCHUCH N., JAKOB W.: Slope-space integrals for specular next event estimation. *Transactions on Graphics (Proceedings of SIGGRAPH Asia) 39*, 6 (Dec. 2020). doi:0.1145/3414685.3417811. 2

[MGN17] MÜLLER T., GROSS M., NOVÁK J.: Practical Path Guiding for Efficient Light-Transport Simulation. *Computer Graphics Forum 36*, 4 (July 2017), 91–100. doi:10.1111/cgf.13227. 2, 6

[MH92] MITCHELL D., HANRAHAN P.: Illumination from curved reflectors. *SIGGRAPH Comput. Graph. 26*, 2 (jul 1992), 283–291. doi:10.1145/142920.134082. 2

[NDVZJ19] NIMIER-DAVID M., VICINI D., ZELTNER T., JAKOB W.: Mitsuba 2: A retargetable forward and inverse renderer. *Transactions on Graphics (Proceedings of SIGGRAPH Asia) 38*, 6 (Dec. 2019). doi:10.1145/3355089.3356498. 5

[NM65] NELDER J. A., MEAD R.: A Simplex Method for Function Minimization. *The Computer Journal 7*, 4 (Jan. 1965), 308–313. doi:10.1093/comjnl/7.4.308. 2

[VG97] VEACH E., GUIBAS L. J.: Metropolis light transport. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques - SIGGRAPH '97* (Not Known, 1997), ACM Press, pp. 65–76. doi:10.1145/258734.258775. 2

[WZHB09] WALTER B., ZHAO S., HOLZSCHUCH N., BALA K.: Single scattering in refractive media with triangle mesh boundaries. *ACM Trans. Graph. 28*, 3 (jul 2009). doi:10.1145/1531326.1531398. 2, 4

[ZGJ20] ZELTNER T., GEORGIEV I., JAKOB W.: Specular manifold sampling for rendering high-frequency caustics and glints. *ACM Transactions on Graphics 39*, 4 (Aug. 2020). doi:10.1145/3386569.3392408. 2, 3, 4, 5

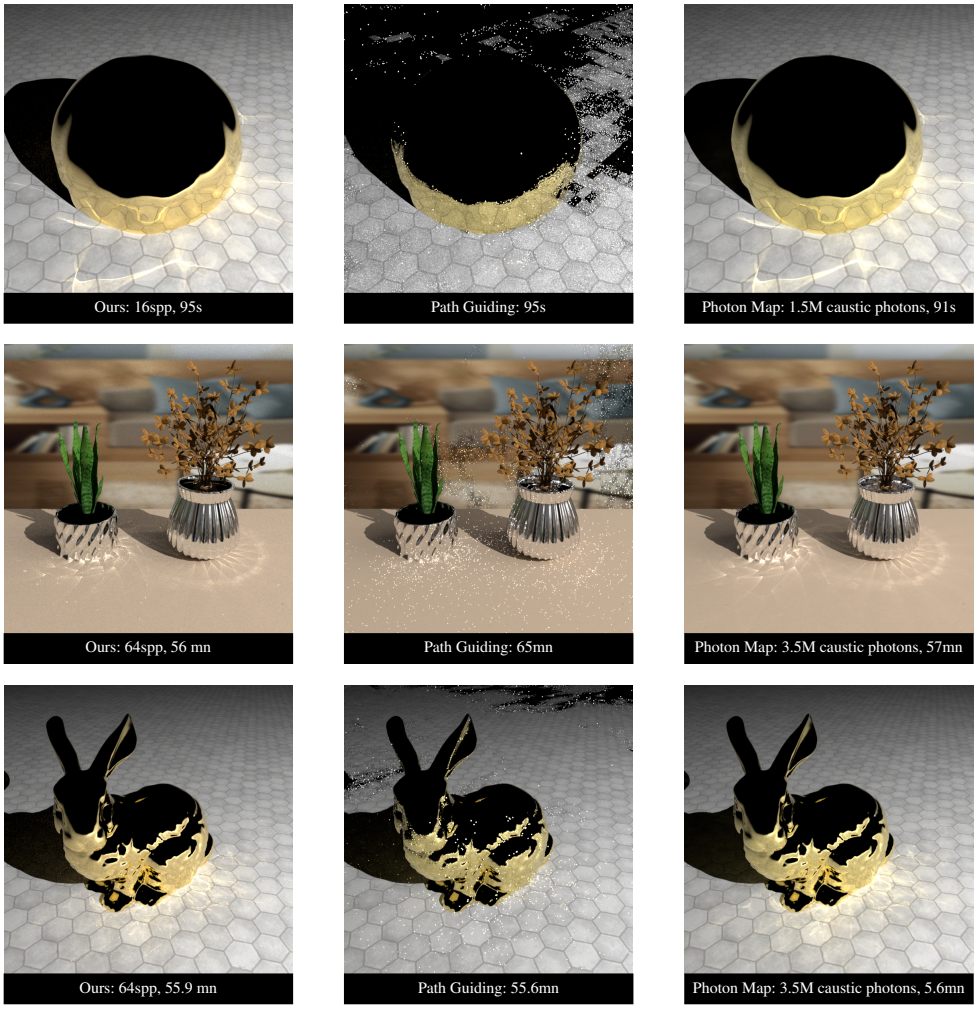


Figure 10: *Equal time comparison between our algorithm (left), Practical Path Guiding (middle), and Photon Mapping (right). In scenes with reflective caustics, Path Guiding has trouble finding the path of light, resulting in noisy images with no defined caustics. For the bumpy sphere scene, our algorithm and Photon Mapping present similar results at the same computation time. For the vase scene, Photon Mapping presents less noise and better-defined caustics. For the bunny scene, Photon mapping achieves the same quality of the image in 10 times less time.*