Ultrafast Screen-Space Refractions and Caustics via Newton's Method

Chase Mayer
Harvard-Westlake School
Ulf Assarsson
Chalmers University of Technology and University of Gothenburg
Erik Sintorn
Chalmers University of Technology and University of Gothenburg

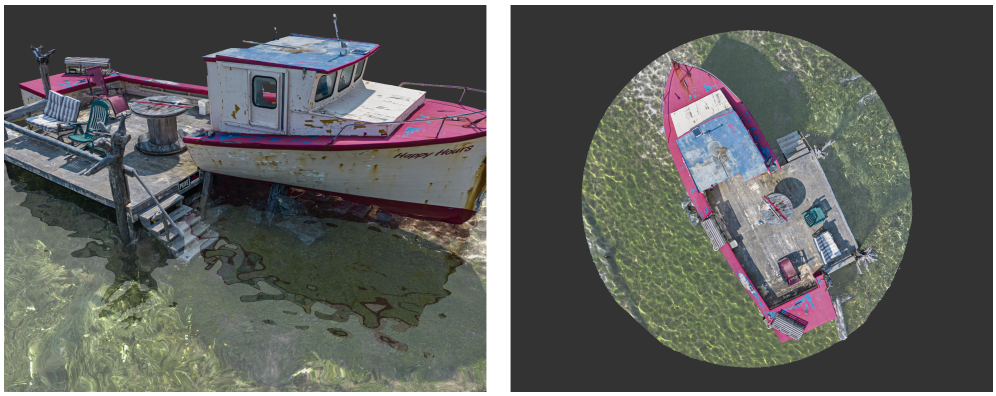


Figure 1. A boat scene rendered using our novel refraction method from two viewpoints. Caustics are also rendered using our method. Refractions take 0.048 ms of the rendering time. Our refraction method succeeds within four iterations for 83% of the pixels in the left image; the other pixels use previous work as a fallback [Sousa 2005]. About 98% of the pixels in the right image succeed within two iterations.

Abstract

We introduce a novel screen-space refraction method that replaces standard ray marching with Newton's method for root finding. For each pixel, we define a function as the difference between the refracted ray depth and the G-buffer depth, and iteratively refine the intersection using the tangent plane of the current G-buffer point. This achieves rapid convergence, typically within only a few iterations, thereby reducing per-pixel cost. We further analyze common failure cases, such as depth discontinuities or poor initial guesses, and propose practical strategies to detect and then mitigate or resolve them. We also demonstrate our approach on refracted light rays to accelerate screen-space caustics.

1. Introduction

Simulating refraction and caustics is essential for realistic rendering of transparent materials such as glass or water. While offline ray tracing can model these effects with high accuracy, interactive rendering must rely on approximations that balance visual fidelity and performance. A common solution is to operate in screen space, where only the visible geometry is considered using depth and normal buffers. This enables interactive refraction [Sousa 2005; Oliveira and Brauwers 2007; Wyman 2005; Ganestam and Doggett 2015] and approximate caustics [Wyman 2006; Meenrattanakorn and Lambers 2022] without access to the full scene geometry.

Screen-space refraction is typically implemented by ray marching through the depth buffer until an intersection is detected [McGuire and Mara 2014]. Although effective, ray marching is slow and may require dozens of steps per pixel to converge, particularly for thick transparent objects or grazing angles. Several works have proposed optimizations, including hierarchical searches [Uludag 2014; Hofmann et al. 2017] and specialized approximations for refractive objects [Wang and Zhang 2021]; yet the step-based nature of ray marching remains a bottleneck. Caustics add further complexity: methods range from image-space photon tracing [Meenrattanakorn and Lambers 2022] to hybrid ray tracing techniques [Wand and Strasser 2003] and sometimes rely on hardware ray tracing support on modern GPUs [Moreau and Doggett 2022]. Some offline techniques reduce the problem to a question of root-finding, encapsulating the constraints of refraction in a polynomial where all solutions can be recovered [Fan et al. 2024]. Currently, the fastest methods leverage the fact that all potential solutions lie in a one-dimensional space, optimizing the root-finding process [Granizo-Hidalgo and Holzschuch 2024].

In contrast, Newton's iterative root-finding method has been used in offline graphics to accelerate ray intersections of parametric surfaces [Toth 1985; Martin et al. 2000; Wang et al. 2002; Park et al. 2013]. The complexity of rendering caustics—namely, the possibility of several different light rays converging to the same spot under the strict physical conditions of refraction—makes it a difficult problem for path tracers. As such, Newton's method has been explored as an avenue for offline caustics rendering [Zeltner et al. 2020]. Its quadratic convergence suggests it could also improve efficiency in screen-space contexts, yet this direction has received little attention in real-time rendering.

Our contributions are the following:

- A real-time screen-space algorithm for refraction rays that replaces linear or hierarchical ray marching with Newton's method to determine the next step along the ray, often achieving significantly faster convergence.
- A discussion of failure cases together with robust fallback strategies.

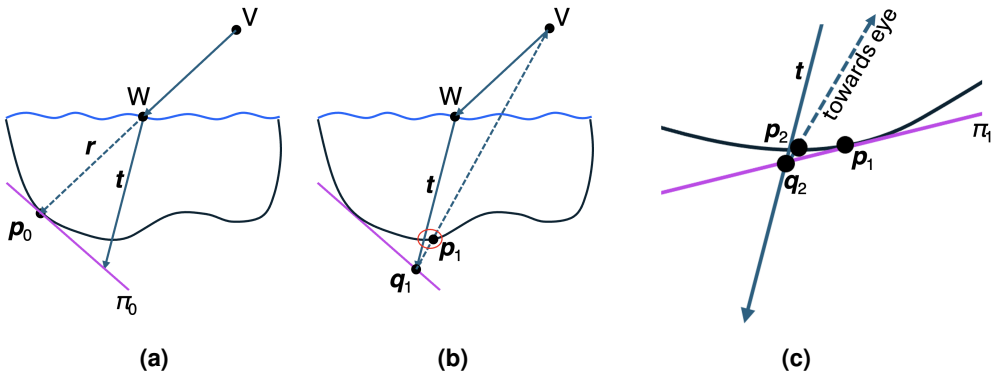


Figure 2. Illustration of our iterative Newton-based refraction solver. (a) Initialization: sample the G-buffer point p_0 and construct the tangent plane π_0 behind the refractive fragment W. (b) Intersect the refracted ray t with π_0, yielding point q_1, which is projected into screen space and re-sampled from the G-buffer to obtain p_1. If p_1 lies sufficiently close to the ray, convergence is achieved. (c) Otherwise, form a new tangent plane π_1 at p_1 and repeat the process until $p_i \approx q_i$ or a maximum number of iterations is reached.

We demonstrate our method by accelerating eye-ray refractions for general screen-space refraction rendering. We also accelerate light-ray refractions to improve the performance of Renou's screen-space caustics method [Renou August 27, 2020].

2. Our Method

We formulate refraction in screen space as an iterative Newton-based solver. For each fragment on a refractive surface, we seek the intersection between the refracted ray and the scene geometry stored in the G-buffer. Rather than advancing in fixed steps along the ray, our method repeatedly constructs a tangent-plane approximation of the scene at the current step and G-buffer position, intersects the refracted ray with this plane, and reprojects the result into screen space. The new screen-space point is then used to resample the G-buffer and construct the next plane, continuing until convergence or until a maximum iteration count is reached.

The algorithm runs in a single fragment shader pass and begins by sampling the geometry buffer directly behind the refractive object to obtain an initial estimate of the ray-scene intersection (Figure 2). Then, the tangent plane is formed, and the refracted ray is intersected with this plane to obtain the updated estimate. The result is projected into screen space, where it provides the lookup coordinates for the next iteration. If the geometry at the new G-buffer coordinate is close enough to the refracted ray, we consider convergence achieved. Otherwise, we continue up to a maximum number of iterations. If convergence is not achieved, fallback methods can optionally be used. The refracted color is obtained by sampling the color buffer at the final computed location.

Convergence is typically achieved within only a few iterations, yielding visually accurate refractions at substantially lower cost than traditional ray marching. Because intermediate estimates are often perceptually convincing, the iteration count may be tuned to trade accuracy for performance. Each iteration requires only a G-buffer lookup of the scene point and normal; in an even more efficient variant, the tangent plane can be stored directly in the G-buffer.

2.1. Newton's Method

The equivalence of our algorithm to Newton's method can be seen by formulating the ray-surface intersection as the root of a function defined by the difference between the refracted ray depth and the G-buffer depth, parameterized in screen space. In this formulation, our use of the tangent plane directly corresponds to the use of the tangent in Newton's method. A formal proof of equivalence is provided in Appendix A.

For Newton's method to converge, the initial estimate must be sufficiently close to the true solution—a condition often satisfied for refraction rays at typical indices of refraction. Convergence also requires local smoothness, which cannot be guaranteed but can be improved through mipmap filtering; by default, the filter size is chosen relative to the distance between the G-buffer sample and the refraction ray. In practice, we approximate opaque geometry with a local tangent plane, constructed either from G-buffer normals or from sampled positions when normals are unavailable. When conditions are met, the method exhibits quadratic convergence, and in our test scenes most pixels converged within two to six iterations.

2.2. Failure Cases

As with any screen-space technique, certain conditions can prevent convergence. For Newton's method to succeed, the geometry buffer must exist in all relevant regions and represent a sufficiently smooth (continuously differentiable) surface, and the initial estimate must be close to the true intersection. These assumptions may be violated in practice, leading to the following failure cases:

- **Occlusion or offscreen geometry:** If the true intersection lies outside the current view, no screen-space method can recover it.
- **Depth discontinuities:** At object boundaries, the G-buffer may fail to represent view-perpendicular surfaces, leading to missing or inconsistent scene information. Object boundaries violate the assumption of a continuously differentiable G-buffer, possibly preventing Newton's method from converging.
- **Highly curved surfaces:** Strong local curvature—for example, when the tangent plane tilts sharply toward the camera—can make the planar approximation unreliable and cause Newton's method to diverge.

- **Poor initialization:** If the initial estimate is too far from the true intersection, convergence may fail, although this is uncommon for a majority of pixels at typical indices of refraction.

We detect failures by two complementary heuristics. First, intersection coordinates must fall within screen bounds. Second, we measure the consistency of the estimate with the refracted ray via pixel error in screen space; i.e., we compare how far the estimate lies from the refraction ray. We use an error threshold of one screen-space pixel, providing similar accuracy as ray marching. Pixels failing these tests can be handled gracefully by falling back to, for instance, fast approximate refraction methods [Sousa 2005], standard ray marching [McGuire and Mara 2014; Uludag 2014; Hofmann et al. 2017], or hardware ray tracing when available.

2.3. Double-Sided Refraction

Thick transparent objects require tracing both entry and exit refractions. We render back-face depths and normals into an auxiliary buffer [Wyman 2005], then apply the iterative solver twice: first to locate the exit point by refracting the ray through the back-face buffer, and then again to determine the intersection with the rest of the scene. This extension enables realistic double-sided refraction in glass-like objects and liquids, and, as in the single-sided case, most pixels converge within only a few iterations per step.

2.4. Caustics via Iterative Refraction Rays

We build on Renou's screen-space caustics approach [Renou August 27, 2020] (and the related ideas from Wallace's WebGL method [Wallace January 7, 2016]) to accelerate the light-ray refractions using our Newton-based solver. Renou's method is notable in that it can converge to the correct solution at sufficiently high buffer and mesh resolutions. The algorithm proceeds in four main stages:

1. **Shadow-map G-buffer generation:** Render the scene from the light's point of view into a G-buffer$_{SM}$. This resembles a shadow map storing depth and normals of all potential caustics receivers. It is also possible to use an ordinary shadow map and reconstruct normals from the depth readings. Using this G-buffer ensures that refracted light rays can be intersected even for receiving surfaces that are offscreen from the camera viewpoint.

2. **Caustics triangle mesh and refraction rays:** Tessellate the caustics-casting transparent surface (e.g., water) into a dense triangle mesh. For each mesh vertex, compute the refracted light ray direction using Snell's law and displace the vertex to its intersection with G-buffer$_{SM}$, using our Newton-based iterative solver rather than ray marching. In effect, each vertex lands on the receiver surface according to the refracted ray.

3. **Rasterization and caustics intensity:** Render the displaced mesh back into shadow-map space, forming caustics triangles. In the fragment shader, each pixel contributes a light intensity to the caustic map, proportional to how much the area of the triangle shrinks or expands under refraction. We compute this area change efficiently using screen-space derivatives (partial derivatives) rather than explicit triangle geometry, following Wallace's insight about computing area ratio via fragment derivatives. To improve robustness, we discard fragments whose depth values deviate significantly from the corresponding G-buffer sample (see Figure 3(c)).

4. **Camera pass with caustics lookup:** During the main camera rendering, apply our eye-ray refraction method and sample the caustics texture (reprojected from light space) to add refracted-light contributions to the scene. This overlays the caustic effect onto the refracted surfaces.

Here are some key advantages and properties of this method:

- The method makes few assumptions about the light source; while point lights are typical, any system that yields refracted rays per mesh vertex is compatible.
- By replacing costly ray marching in the light's G-buffer with our Newton-based solver, the caustics pass becomes significantly more efficient.
- The algorithm degrades gracefully: if convergence fails for certain vertices, we simply discard them (i.e., they do not emit caustic contributions), thereby avoiding artifacts from invalid triangles (see Figure 3(a)).
- With sufficiently fine mesh tessellation and a high-resolution G-buffer relative to the geometry curvature, the technique converges toward the exact geometric solution of caustics on the G-buffer geometry. At lower resolutions, the result is approximate but visually plausible.

In summary, our adaptation accelerates Renou's screen-space caustics by replacing ray marching with iterative root finding, while maintaining graceful degradation and capability of converging to correct results under sufficient resolutions.

3. Results and Discussion

We evaluate our method on a set of representative scenes commonly found in real-time applications, e.g., games:

- **Pool:** A swimming pool with a dynamic wavy water surface, producing both strong refractions and visible caustics. The large, flat structures are particularly well-suited for our method.

Scene	Cost per Iter.	# Iter.	Success Rate [%]	Cost Ours [ms]	Cost Ray Marching [ms]
Pool	0.026	3	75–99%	0.078	2.6
Buddha	0.042	5	82–85%	0.24	4.4
Boat	0.012	4	83–98%	0.048	3.8

Table 1. All costs are reported in milliseconds. Success rates indicate the percentage of refraction rays (i.e., screen pixels) that converged across various viewpoints shown in Table 2. *Cost Ours* denotes the additional execution time introduced by our refraction method within the full shading pipeline, whereas *Cost Ray Marching* refers to the corresponding overhead incurred by standard ray marching. A fixed number of iteration counts was used here, denoted by *# Iter*.

- **Buddha:** A thick, highly refractive glass object rendered against a skybox and uneven heightfield background. The Buddha model comes from the Stanford 3D Scanning Repository [Stanford Computer Graphics Laboratory 2023].
- **Boat:** A shallow-water scene with a bumpy lake bottom and dense, elongated seaweed introducing multiple depth discontinuities and refraction challenges [Air Digital 2023].

All experiments were performed at a resolution of 2000×1600 on an NVIDIA GeForce RTX 3070 with 14 GB of memory. Table 1 reports execution times, iteration counts, and success rates, and compares against standard ray marching. To ensure pixel-perfect ray marching results, we use the algorithm of McGuire and Mara [McGuire and Mara 2014]. In all scenes, we use fast screen-space methods as fallbacks in case of failure [Sousa 2005]. Table 2 displays visual results versus ray marching and includes graphs of convergence rates versus the number of iterations.

3.1. Performance

Across all test scenes, our method converges within two to six iterations for the majority of pixels. The Pool scene needs only three iterations to succeed for 75–99% of the pixels. Similar gains are observed in the Boat scene. The Buddha scene converges in five to six iterations for 85% of the pixels and also highlights the increased difficulty of double-sided refractions in complex geometries. Our per-iteration cost only depends on the number of refractive pixels given the view position and is otherwise independent of the scenes.

3.2. Convergence Behavior

Table 2 shows convergence rates across the test scenes. As expected from Newton's method, convergence is quadratic once the initial guess is sufficiently close, explaining why most pixels converge in only a few steps. Failures occur mainly at object sil-

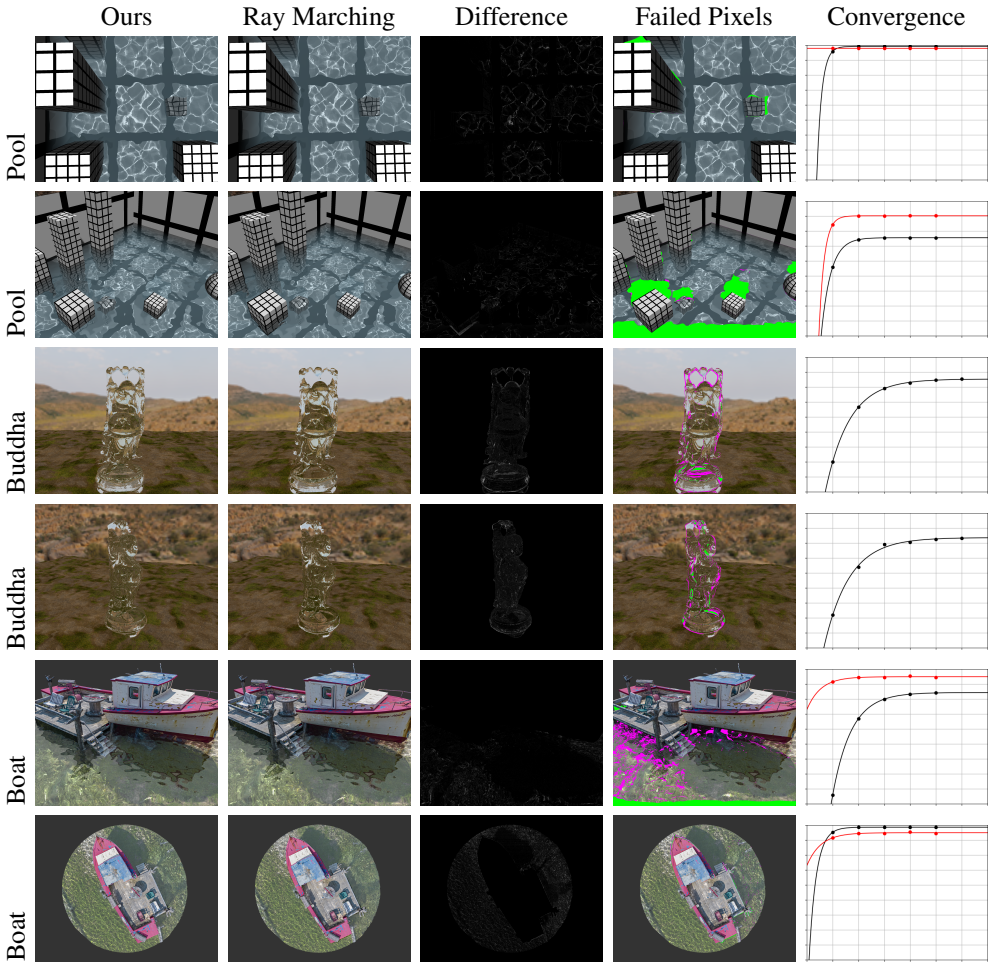


Table 2. Comparison of refraction results across our scenes for multiple viewpoints. Columns show (1) our results, (2) ray marching as ground truth, (3) their difference, (4) failure cases where the refracted position error exceeds one screen-space pixel (green where ray marching also fails, magenta where only our method fails), and (5) convergence rates for the percent of pixels succeeding (10–100%) vs number of iterative refinement steps (0–7), with black for eye-ray refractions and also red for light-ray refractions when caustics are present. Graphs generated with Matplotlib [Hunter 2007].

houettes, depth discontinuities, or strongly curved surfaces, where the tangent-plane approximation becomes unreliable. In these cases, we can use fallback methods, ensuring robustness without introducing severe artifacts.

3.3. Double-Sided Refractions

For thick transparent objects such as the Buddha, we extend the method to trace both entry and exit refractions using an auxiliary back-face buffer. Most interior pixels

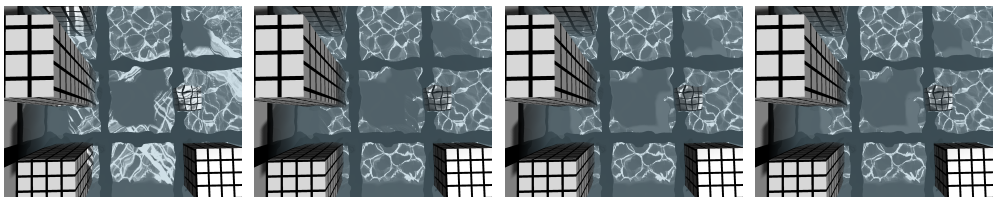


Figure 3. Handling of failure cases in the caustics pass. (a) No rejection of invalid refraction results. Triangles with failed vertex refractions produce severe artifacts. (b) Discarding triangles for which at least one vertex failed to converge removes most invalid contributions, revealing the valid ones. (c) Additionally discarding fragments whose depth deviates from the shadow-map depth eliminates nearly all remaining inconsistencies and yields a clean result. (d) Ray-marched reference.

converge successfully within a few iterations, producing convincing double-sided refractions (see Table 2). However, pixels near object boundaries are more error-prone. It is possible for pixels around the edge of a transparent object to generate estimates that lie outside of the object entirely; when the iterative refinement steps sample the back-face buffer, they find no relevant data. Thus, our method is prone to failure around object outlines for thick transparent objects. In contrast, most pixels closer to the middle of the object succeed after a few iterations.

3.4. Caustics

We also evaluate our Newton-based solver in the context of screen-space caustics, building on Renou's method but replacing the light-ray marching step with our iterative solver. The caustics triangle mesh uses a tessellation of 200×200 for the caustics-casting surface. When failures occur, discarding invalid caustics triangles and fragments avoids major artifacts (see Figure 3). Convergence graphs for the light-ray refractions are displayed in Table 2. The refractions converge in just one iteration for the caustics rays when the light source is located just above the water surface. A similar trend is observed for eye-ray refractions when the camera is positioned orthogonally above the refractive surface. (See the Pool and Boat scenes in the first and last rows of Table 2 for both cases.) This behavior arises because such a viewing configuration statistically minimizes refraction-induced displacements in screen space, causing the initial estimate to lie close to the true intersection point.

The full source code is provided via the GitHub link at the end of this paper.

3.5. Discussion

Our method requires only a single G-buffer fetch per iteration. In contrast, hierarchical screen-space traversals using mipmapping [Uludag 2014; Hofmann et al. 2017]

can reduce the number of steps compared to linear ray marching, but they cannot avoid descending to the leaf level and often need to traverse up and down multiple times before reaching the intersection. Likewise, binary-search ray marching also typically requires more than a handful of iterations. Since the number of texture accesses largely determines the cost of the shader, our method—with only three to six G-buffer fetches in total per ray—can typically achieve significant cost reductions for the majority of rays.

Moreover, the iterative approach offers predictable performance, making it well suited for real-time applications with strict frame-time budgets. Although failure cases remain, standard screen-space methods can serve as effective fallbacks, thereby mitigating visual errors.

4. Conclusion, Limitations, and Future Work

We have presented a real-time screen-space refraction method that replaces traditional ray marching with Newton's method, achieving rapid convergence typically within just a few iterations. To ensure robustness, we detect failure cases and resolve them through appropriate fallback strategies. The method has been demonstrated for both eye rays (enabling efficient refraction rendering) and light rays (where it accelerates the computation of screen-space caustics).

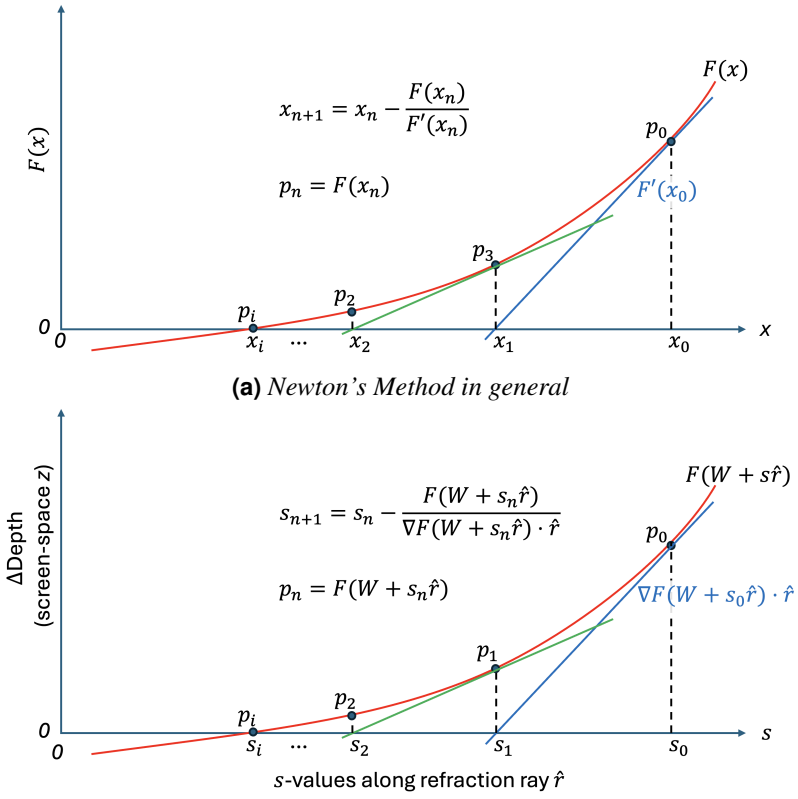
The primary limitation of our method is its reliance on screen-space information: geometry not visible to the camera cannot be incorporated into refractions without significant additional effort. Future work could explore techniques such as depth peeling or related approaches to account for occluded objects.

A further limitation is the planar approximation of opaque geometry, which may fail to converge on highly curved surfaces. More advanced representations—such as parabolic surfaces or higher-order Taylor expansions—could improve robustness. Another promising direction is to extend the method to reflection rays. While the primary ray hit often provides a poor initial estimate, reflected positions from the previous frame could serve as effective initializations.

A. Proof of Equivalence to Newton's Method

In this appendix, we show that our iterative refraction algorithm is equivalent to applying Newton's method to a scalar root-finding problem defined along the refracted ray. Figure 4 provides a visual reference for the proof.

Locally and for each refraction ray, we assume the opaque scene geometry can be represented by a twice continuously differentiable implicit surface $F(x) = 0$, where $x \in \mathbb{R}^3$. $F(x)$ at any given point x is given by the depth difference between x and the corresponding point on the G-buffer after projection into screen space. In practice, this surface is not available analytically but is sampled discretely via the G-buffer,



(a) *Newton's Method in general*

(b) *Our method*

Figure 4. (a) Geometrically, Newton's method for finding the root $F(x) = 0$ refines each iteration x_{n+1} as the intersection of the tangent of F at x_n with the x-axis. (b) In our method, the scene geometry is represented by an implicit surface $F(x) = 0$. Restricting this function to the refracted ray $x(s) = W + s\hat{r}$ yields the scalar function $g(s) = F(W + s\hat{r})$. In practice, the sampled value of $F(W + s\hat{r})$ corresponds to the signed screen-space depth difference between the G-buffer surface and the refracted ray at the projected location. The curve illustrates $g(s)$ for different positions s along the ray. The tangent lines shown correspond to the intersections of the surface tangent planes with the plane spanned by the refracted ray and the screen-space depth axis. Computing s_{n+1} as the intersection of the tangent plane at s_n with the refracted ray is algebraically identical to performing a Newton iteration.

which stores depth and normal information in screen space. Sampling the normal buffer at any given point provides the gradient of F at that point; this is clear for points where $F(x) = 0$, since the gradient of F is the normal of the surface. For other points that do *not* lie on the surface, $F(x) \neq 0$. Suppose, for example, that the depth difference between x and the corresponding point in the G-buffer is 1. Then ∇F yields the normal of the implicit surface $F(x) = 1$. The surface $F(x) = 1$ has the same shape as $F(x) = 0$, just shifted toward the camera, so its normal can be

retrieved by sampling the G-buffer.

Let the refracted ray be parameterized as $x(s) = W + s\hat{r}$, where W is the ray origin and $\hat{r}$ is the normalized refracted direction. Restricting F to this ray defines the scalar function $g(s) = F(W + s\hat{r})$. A root s^* of $g(s)$ corresponds exactly to an intersection between the refracted ray and the opaque scene geometry.

Although $F(x)$ is defined abstractly, in our screen-space implementation its sampled value at $x = W + s\hat{r}$ corresponds numerically to the signed depth difference between the G-buffer surface at the projected screen-space location and the refracted ray. Thus, finding a zero of $g(s)$ is equivalent to finding where the ray and the G-buffer surface coincide in depth.

We now show that intersecting the refracted ray with the tangent plane of the G-buffer surface at the current estimate is algebraically identical to performing a Newton iteration on $g(s)$.

Proof

Locally and individually for each refraction ray, the opaque scene can be represented by a C^2 implicit function $F(x) = 0$. Along the refracted ray $W + s\hat{r}$, we define $g(s) = F(W + s\hat{r})$; a root s^* of g corresponds exactly to the ray-surface intersection.

We can use Newton's method to approximate this root by repeating the iteration

$$s_{n+1} = s_n - \frac{g(s_n)}{g'(s_n)}.$$

Intersecting the ray with the surface tangent plane at the current iteration $X_n = W + s_n\hat{r}$ is algebraically identical to performing a Newton step $s_{n+1} = s_n - \frac{g(s_n)}{g'(s_n)}$.

Taylor (first-order) linear approximation of F around X_n is

$$F(X) \approx F(X_n) + \nabla F(X_n) \cdot (X - X_n).$$

To approximate the root $F(X) = 0$, we set the linear model to zero:

$$F(X_n) + \nabla F(X_n) \cdot (X - X_n) = 0.$$

This equation defines an affine plane in $\mathbb{R}^3$. Substituting $X = W + s\hat{r}$ into the linear approximation gives

$$F(X_n) + \nabla F(X_n) \cdot (W + s\hat{r} - X_n).$$

Since $X_n = W + s_n\hat{r}$, we have $W + s\hat{r} - X_n = (s - s_n)\hat{r}$. Thus,

$$F(X_n) + \nabla F(X_n) \cdot ((s - s_n)\hat{r}) = 0.$$

Factor out $s - s_n$:

$$F(X_n) + (s - s_n)(\nabla F(X_n) \cdot \hat{r}) = 0.$$

Solve for s:

$$s - s_n = -\frac{F(X_n)}{\nabla F(X_n) \cdot \hat{r}}.$$

However, $F(X_n) = g(s_n)$ and $g'(s) = \nabla F(X_n) \cdot \hat{r}$ by the chain rule. Therefore,

$$s = s_n - \frac{g(s_n)}{g'(s_n)}.$$

This is precisely the same expression as Newton's method. Newton's method is known to converge locally ($\lim_{n\to\infty} s_n = s^*$) as long as g is continuously differentiable in a neighborhood around s^*, $g'(s^*) \neq 0$, and the initial guess s_0 is sufficiently close to the root s^*. Thus, our method converges as long as the geometry buffer encodes a continuously differentiable surface, the refracted ray $\hat{r}$ is not parallel to the surface at the point of intersection, and our initial guess is close enough to the true intersection point. These assumptions can cause errors when they are not met; we discuss these errors and the methods to resolve them in Section 2.2. It is also worth noting that the geometry buffer, being discrete, can never be truly continuously differentiable. However, for most geometry buffers of reasonable resolution, the discontinuities at each pixel are not a problem.

Acknowledgments

This research was partially funded by Sweden's Innovation Agency.

Index of Supplemental Materials

The three videos are available:

Pool: https://jcgt.org/published/0015/01/03/PoolVideo.mp4

Buddha: https://jcgt.org/published/0015/01/03/BuddhaVideo.mp4

Boat: https://jcgt.org/published/0015/01/03/BoatVideo.mp4

In addition, the code is available on GitHub at https://github.com/TheManTheMythTheGameDev/NewtonsMethodRefraction.

References

AIR DIGITAL. Happy hours—Salmon Beach Boat, New Brunswick. 3D model, *Sketchfab*, https://sketchfab.com/3d-models/happy-hours-salmon-beach-boat-new-brunswick-4e05e17f154f40158e2dd2b32bba75df, September 2023. Accessed October 3, 2025. 50

FAN, Z., GUO, J., WANG, Y., XIAO, T., ZHANG, H., ZHOU, C., CHEN, Z., HONG, P., GUO, Y., AND YAN, L.-Q. Specular polynomials. *ACM Transactions on Graphics*, 43(4): 126:1–126:13, July 2024. URL: https://doi.org/10.1145/3658132. 45

GANESTAM, P. AND DOGGETT, M. Real-time multiply recursive reflections and refractions using hybrid rendering. *The Visual Computer*, 31(10):1395–1403, 2015. URL: https://doi.org/10.1007/s00371-014-1021-7. 45

GRANIZO-HIDALGO, A. AND HOLZSCHUCH, N. Interactive rendering of caustics using dimension reduction for manifold next-event estimation. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 7(1):12:1–12:16, May 2024. URL: https://doi.org/10.1145/3651297. 45

HOFMANN, N., BOGENDÖRFER, P., STAMMINGER, M., AND SELGRAD, K. Hierarchical multi-layer screen-space ray tracing. In *Eurographics/ACM SIGGRAPH Symposium on High Performance Graphics*, pages 18:1–18:10. ACM, 2017. URL: https://doi.org/10.1145/3105762.3105781. 45, 48, 52

HUNTER, J. D. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. URL: https://doi.org/10.1109/MCSE.2007.55. 51

MARTIN, W., COHEN, E., FISH, R., AND SHIRLEY, P. Practical ray tracing of trimmed NURBS surfaces. *Journal of Graphics Tools*, 5(1):27–52, 2000. URL: https://doi.org/10.1080/10867651.2000.10487519. 45

MCGUIRE, M. AND MARA, M. Efficient GPU screen-space ray tracing. *Journal of Computer Graphics Techniques (JCGT)*, 3(4):73–85, 2014. URL: http://jcgt.org/published/0003/04/04/. 45, 48, 50

MEENRATTANAKORN, K. AND LAMBERS, M. Real-time caustics using cascaded image-space photon tracing. In *Vision, Modeling, and Visualization*, VMV 2022, pages 127–134. The Eurographics Association, 2022. URL: https://doi.org/10.2312/vmv.20221212. 45

MOREAU, P. AND DOGGETT, M. Real-time rendering of indirectly visible caustics. In *Proceedings of the 17th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, volume 1, pages 39–48. SciTePress, 2022. URL: https://doi.org/10.5220/0010777800003124. 45

OLIVEIRA, M. M. AND BRAUWERS, M. Real-time refraction through deformable objects. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, I3D '07, page 89–96. Association for Computing Machinery, 2007. URL: https://doi.org/10.1145/1230100.1230116. 45

PARK, T., JI, J., AND KO, K. H. A second order geometric method for ray/-parametric surface intersection. *Computer Aided Geometric Design*, 30(8):795–804, 2013. URL: https://www.sciencedirect.com/science/article/pii/S0167839613000678. 45

RENOU, M. Real-time rendering of water caustics. *Medium*, https://medium.com/@martinRenou/real-time-rendering-of-water-caustics-59cda1d74aa, August 27, 2020. Accessed July 2025. 46, 48

SOUSA, T. Generic refraction simulation. In PHARR, M., editor, *GPU Gems 2*, chapter 19. Addison-Wesley, 2005. URL: https://developer.nvidia.com/gpugems/gpugems2/part-ii-shading-lighting-and-shadows/chapter-19-generic-refraction-simulation. 44, 45, 48, 50

STANFORD COMPUTER GRAPHICS LABORATORY. The Stanford 3D scanning repository. https://graphics.stanford.edu/data/3Dscanrep/, 2023. Last modified April 6, 2023. 50

TOTH, D. L. On ray tracing parametric surfaces. *ACM SIGGRAPH Computer Graphics*, 19 (3):171–179, 1985. URL: https://doi.org/10.1145/325334.325233. 45

ULUDAG, Y. Hi-Z screen-space cone-traced reflections. In ENGEL, W., editor, *GPU Pro 5*, chapter 13, pages 149–192. A K Peters/CRC Press, 2014. URL: https://www.taylorfrancis.com/chapters/edit/10.1201/b16721-13/. 45, 48, 52

WALLACE, E. Rendering realtime caustics in WebGL. *Medium*, https://medium.com/@evanwallace/rendering-realtime-caustics-in-webgl-2a99a29a0b2c, January 7, 2016. Accessed July 2025. 48

WAND, M. AND STRASSER, W. Real-time caustics. *Computer Graphics Forum*, 22(3):611–620, 2003. URL: https://doi.org/10.1111/1467-8659.t01-2-00709. 45

WANG, S.-W., SHIH, Z.-C., AND CHANG, R. C. An efficient and stable ray tracing algorithm for parametric surfaces. *Journal of Information Science and Engineering*, 18 (4):541–561, 2002. URL: https://api.semanticscholar.org/CorpusID:15339898. 45

WANG, X. AND ZHANG, R. Rendering transparent objects with caustics using real-time ray tracing. *Computers & Graphics*, 96:36–47, May 2021. URL: https://doi.org/10.1016/j.cag.2021.03.003. 45

WYMAN, C. Interactive image-space refraction of nearby geometry. In *Proceedings of the 3rd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, GRAPHITE '05, page 205–211. Association for Computing Machinery, 2005. URL: https://doi.org/10.1145/1101389.1101431. 45, 48

WYMAN, C. Interactive image-space techniques for approximating caustics. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, pages 153–160. Association for Computing Machinery, 2006. URL: https://doi.org/10.1145/1111411.1111439. 45

ZELTNER, T., GEORGIEV, I., AND JAKOB, W. Specular manifold sampling for rendering high-frequency caustics and glints. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 39(4):149:1–149:15, July 2020. URL: https://doi.org/10.1145/3386569.3392408. 45

Author Contact Information

Chase Mayer
Harvard-Westlake School
3700 Coldwater Canyon Ave.
Studio City, CA 91604
mayerchase559@gmail.com

Ulf Assarsson
Chalmers University of Technology and University of Gothenburg
Department of Computer Science and Engineering
412 96 Göteborg, Sweden

Erik Sintorn
Chalmers University of Technology and University of Gothenburg
Department of Computer Science and Engineering
412 96 Göteborg, Sweden

Chase Mayer, Ulf Assarsson, and Erik Sintorn, Ultrafast Screen-Space Refractions and Caustics via Newton's Method, *Journal of Computer Graphics Techniques (JCGT)*, vol. 15, no. 1, 44–59, 2026
http://jcgt.org/published/0015/01/03/

Received:	2025-11-10		
Recommended:	2026-02-23	Corresponding Editor:	Beibei Wang
Published:	2026-04-19	Editor-in-Chief:	Alexander Wilkie