ToF ReSTIR: Time-of-Flight Rendering with Spatio-temporal Reservoir Resampling Supplementary Material

JUHYEON KIM, Dartmouth College, USA
WOJCIECH JAROSZ, Dartmouth College, USA
ADITHYA PEDIREDLA, Dartmouth College, USA

CCS Concepts: • **Computing methodologies** → **Ray tracing**; **Computational photography**.

Additional Key Words and Phrases: Time-of-Flight imaging, Time-of-Flight rendering, real-time ray tracing

ACM Reference Format:
Juhyeon Kim, Wojciech Jarosz, and Adithya Pediredla. 2026. ToF ReSTIR: Time-of-Flight Rendering with Spatio-temporal Reservoir Resampling Supplementary Material. *ACM Trans. Graph.* 45, 4, Article 89 (July 2026), 2 pages. https://doi.org/10.1145/3811299

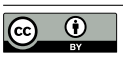
1 More Discussion on Temporal Manifold

Here, we provide additional discussion on Newton's method in the main manuscript.

Gradient Flow. An intuitive alternative for handling the additional degrees of freedom is to search along a continuous *gradient flow* $\xi(s)$, defined by $d\xi/ds = \nabla L(\xi(s))$. This approach guarantees the bijectivity of the mapping and can provide more robust search behavior than Newton's method, albeit at the cost of slower convergence (linear rather than quadratic). The determinant of the Jacobian can be evaluated using *Liouville's formula* [Chicone 2006], which integrates the divergence of the gradient field along the flow trajectory. While gradient flow resembles gradient descent used in optimization, our goal here is different: we are interested in the entire trajectory to ensure bijectivity and to correctly evaluate the Jacobian term. As a result, the trajectory must be followed using sufficiently small steps from the start to the end point (i.e., without adaptive step-size schemes such as Adam), ensuring that the solution remains on the same flow. This requirement makes gradient-flow-based approaches impractical in our setting.

Comparison with Euler and Runge–Kutta (RK) Methods. Enforcing a gauge condition is closely related to finite-difference integration methods such as Euler and Runge–Kutta (RK). Searching along the initial gradient direction can be interpreted as an explicit Euler method. One may also consider an implicit midpoint method, which replaces $m = (\nabla\ell(\xi) + \nabla\ell(\xi'))/2$ with $m = \nabla\ell((\xi + \xi')/2)$. This formulation also guarantees bijectivity and behaves similarly to the average-gradient method. Both approaches are second-order accurate and differ only in where the gradient and Hessian are evaluated (midpoint versus endpoint). However, since our objective is to solve for the endpoint ξ', the midpoint method still requires evaluating ξ', which introduces additional computational overhead. One could further consider higher-order symmetric RK variants to preserve bijectivity. While these methods may improve the stability of Newton's iteration, they incur significantly higher computational cost. Our average-gradient formulation, which uses gradients at the start and end points, corresponds to the trapezoidal (Crank–Nicolson) method [Ascher and Petzold 1998] in the numerical analysis literature.

Authors' Contact Information: Juhyeon Kim, Dartmouth College, USA, juhyeon.kim.gr@dartmouth.edu; Wojciech Jarosz, Dartmouth College, USA, wojciech.k.jarosz@dartmouth.edu; Adithya Pediredla, Dartmouth College, USA, adithya.k.pediredla@dartmouth.edu.



ACM 1557-7368/2026/7-ART89
https://doi.org/10.1145/3811299

2 Derivation for Doppler Frequency Rendering

Here, we derive the gradient and Hessian required for Newton's iteration in Doppler frequency rendering, which were omitted in the main manuscript.

For Doppler frequency rendering, we impose the following path-length derivative constraint:

$$\frac{\mathrm{d}}{\mathrm{d}t}\ell(\bar{\mathbf{y}}') = \frac{\mathrm{d}}{\mathrm{d}t}\ell(\bar{\mathbf{x}}) + \delta u, \tag{1}$$

and solve the updated constraint

$$F(\xi, \xi') = \begin{bmatrix} \frac{\mathrm{d}}{\mathrm{d}t}\ell(\xi') - \frac{\mathrm{d}}{\mathrm{d}t}\ell(\xi) - \delta u \\ G(\xi') - G(\xi) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{2}$$

The only modification from the transient case is that the path-length functional ℓ is replaced by its temporal derivative $\frac{\mathrm{d}}{\mathrm{d}t}\ell$. As before, we first evaluate the gradient and Hessian with respect to $\mathbf{p}$, and then transform them to ξ. The path length is given by

$$\ell(\mathbf{p}) = \|\mathbf{p}_1 - \mathbf{p}\| + \|\mathbf{p}_2 - \mathbf{p}\| \tag{3}$$

and its temporal derivative (path velocity) is

$$\frac{\mathrm{d}}{\mathrm{d}t}\ell(\mathbf{p}) = \frac{\mathrm{d}}{\mathrm{d}t}\left(\|\mathbf{p}_1 - \mathbf{p}\| + \|\mathbf{p}_2 - \mathbf{p}\|\right). \tag{4}$$

Define

$$\mathbf{r}_i := \mathbf{p}_i - \mathbf{p}, \quad d_i := \|\mathbf{r}_i\|, \quad \mathbf{d}_i := \frac{\mathbf{r}_i}{d_i} \qquad (i \in \{1, 2\}), \tag{5}$$

and velocities

$$\mathbf{v}_1 := \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{p}_1, \quad \mathbf{v}_2 := \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{p}_2, \quad \mathbf{v} := \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{p}. \tag{6}$$

Using $\frac{\mathrm{d}}{\mathrm{d}t}\|\mathbf{r}_i\| = \frac{\mathbf{r}_i}{\|\mathbf{r}_i\|} \cdot \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{r}_i$ and $\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{r}_i = \frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{p}_i - \mathbf{p}) = \mathbf{v}_i - \mathbf{v}$, we obtain

$$\frac{\mathrm{d}}{\mathrm{d}t}\ell(\mathbf{p}) = \frac{\mathrm{d}}{\mathrm{d}t}\|\mathbf{p}_1 - \mathbf{p}\| + \frac{\mathrm{d}}{\mathrm{d}t}\|\mathbf{p}_2 - \mathbf{p}\| \tag{7}$$

$$= \mathbf{d}_1 \cdot (\mathbf{v}_1 - \mathbf{v}) + \mathbf{d}_2 \cdot (\mathbf{v}_2 - \mathbf{v}). \tag{8}$$

By defining $\mathbf{u}_i = \mathbf{v}_i - \mathbf{v}$, we have

$$\frac{\mathrm{d}}{\mathrm{d}t}\ell(\mathbf{p}) = \mathbf{d}_1 \cdot \mathbf{u}_1 + \mathbf{d}_2 \cdot \mathbf{u}_2. \tag{9}$$

For brevity, we denote this quantity by $u(\mathbf{p})$.

We now evaluate the gradient and Hessian of u with respect to $\mathbf{p}$. Using $\nabla_{\mathbf{r}}\mathbf{d} = \frac{1}{d}(I - \mathbf{d}\mathbf{d}^\top)$ and $\nabla_{\mathbf{p}}\mathbf{r}_i = -I$, we obtain

$$\nabla_{\mathbf{p}} u(\mathbf{p}) = -\frac{I - \mathbf{u}_1\mathbf{u}_1^T}{d_1} - \frac{I - \mathbf{u}_2\mathbf{u}_2^T}{d_2}. \tag{10}$$

Define $c_i = \mathbf{d}_i \cdot \mathbf{u}_i$, then the Hessian is

$$\nabla^2_{\mathbf{p}} u(\mathbf{p}) = -\sum_{i=1}^{2} \frac{1}{d_i^2}\left[\mathbf{d}_i\mathbf{u}_i^\top + \mathbf{u}_i\mathbf{d}_i^\top - 2c_i\,\mathbf{d}_i\mathbf{d}_i^\top - c_i I\right] \tag{11}$$

Finally, we transform derivatives to the ξ domain using $J = \frac{\partial \mathbf{p}}{\partial \xi}$:

$$\nabla_\xi u(\xi) = J^T \nabla_{\mathbf{p}} u(\mathbf{p}) \tag{12}$$

and

$$\nabla^2_\xi u(\xi) = J^T \nabla^2_{\mathbf{p}} u(\mathbf{p}) J, \tag{13}$$

where, as in the path-length case, we ignore surface curvature terms.

3 Additional Results

In this section, we present additional results and implementation details that were not included in the main paper.

Coordinate Comparison. We compare (1) a local infinite plane parameterization and (2) a global hemispherical parameterization with ray tracing, both initialized from $\mathbf{p}_2$. Theoretically, the local plane approach is faster, as it does not require ray tracing during iteration; however, it struggles in regions with dense triangle meshes. The error map under an equal time budget of 5 seconds is shown in Fig. 1. As expected, the local plane approach performs worse in regions with dense geometry (yellow-boxed area) compared to the ray tracing method. However, it can be more accurate in certain regions (red-boxed area), particularly near the edges of large planar surfaces. This behavior arises because the hemispherical parameterization is nonlinear in polar coordinates, whereas the local plane provides a more faithful representation when the surface is approximately planar. Except for trivial cases (e.g., scenes composed only of fine meshes or only of large quads), a practical heuristic is to switch between the two parameterizations based on the area of the hit triangle. This strategy is also used in the main manuscript.

We also provide additional coordinate systems (e.g., barycentric coordinates) in our implementation. Users can further implement custom parameterizations tailored to their needs, such as those based on SDF-defined implicit surfaces.

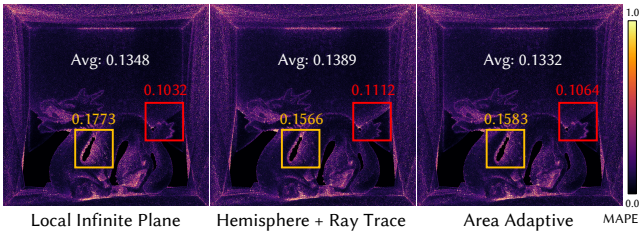


Fig. 1. Comparison of coordinate parameterizations for Newton search. The local infinite plane performs well on large planar surfaces, while the hemispherical parameterization with ray tracing is more robust on fine meshes. We adopt an area-adaptive strategy that selects the parameterization based on the triangle size at the reconnection vertex.

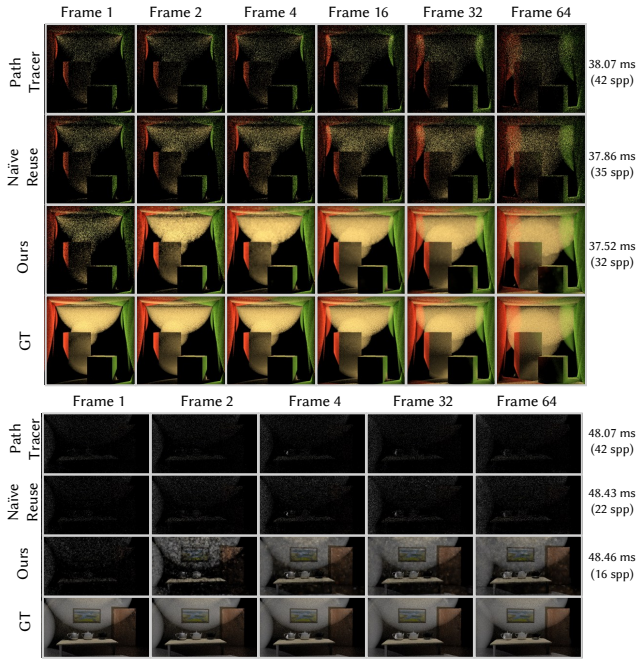


Fig. 2. Frame-to-frame transient rendering results demonstrating a clear advantage of our proposed method over baseline approaches.

Temporal Progress of Transient Rendering. We show the frame-to-frame progression of transient rendering with temporal reuse on static scenes in Fig. 2 under an equal-time budget. We observe that, after only a few frames (about four frames), our method already converges toward the ground-truth transient frames, whereas naive path tracing or naive path reuse fails to produce meaningful results.

Other Implementation Details. Basically, we implement our algorithm within the Falcor framework [Kallweit et al. 2022]. Instead of using the built-in `PathTracer` render pass, we implement a custom lightweight inline ray-tracing–based path tracer, which serves as the backbone for ToF rendering. For ellipsoidal path connection, we reuse the existing light BVH data structure to store scene triangles. For the NLOS imaging demonstration, we explicitly ignore the back wall during ellipsoidal connection to improve sampling efficiency.

References

Uri M Ascher and Linda R Petzold. 1998. *Computer methods for ordinary differential equations and differential-algebraic equations.* SIAM.

Carmen Chicone. 2006. *Ordinary differential equations with applications.* Springer.

Simon Kallweit, Petrik Clarberg, Craig Kolb, Tom'aš Davidovič, Kai-Hwa Yao, Theresa Foley, Yong He, Lifan Wu, Lucy Chen, Tomas Akenine-Möller, Chris Wyman, Cyril Crassin, and Nir Benty. 2022. The Falcor Rendering Framework. https://github.com/NVIDIAGameWorks/Falcor https://github.com/NVIDIAGameWorks/Falcor.