Supplementary Material:
Mixwell: Sharp 2D Fluid Brushes for Progressive Physics-Based Mixing

DOUG L. JAMES, Stanford University, USA
ETHAN JAMES, Independent Researcher, USA

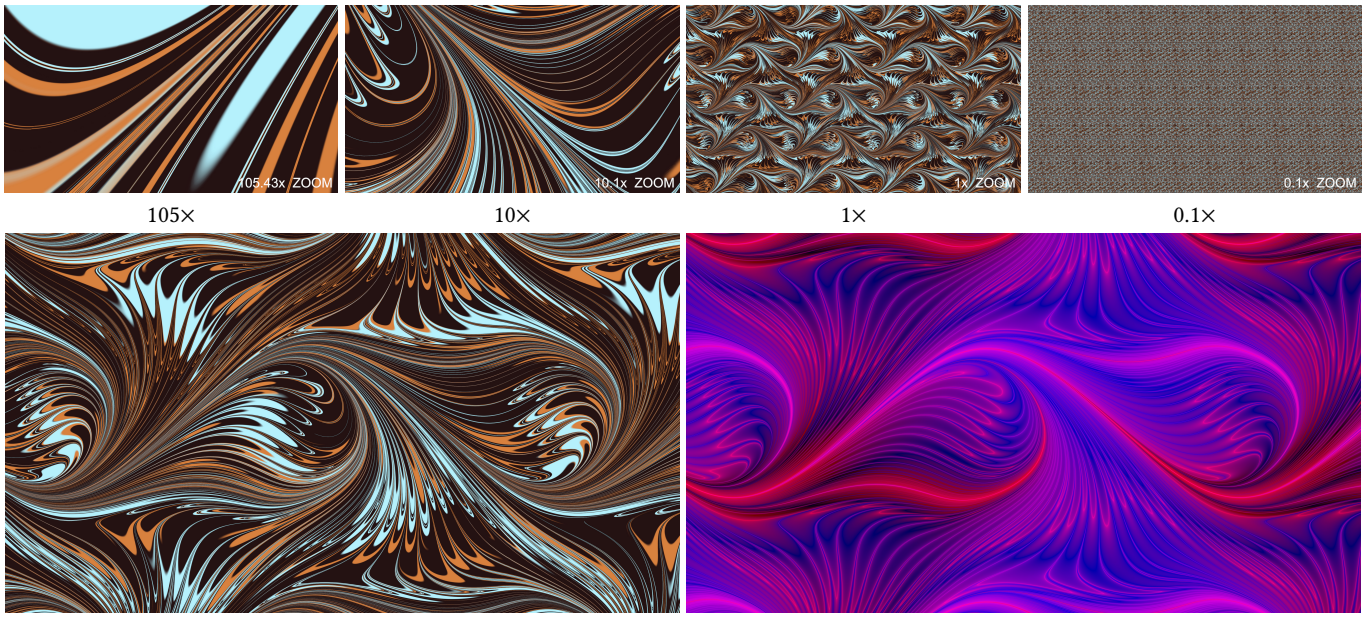
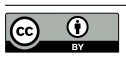


Fig. 1. **Mixwell One-step Newton Solver for `rdSegment`:** Mixwell introduces semi-analytical methods for potential flow around a cylinder, enabling interactive one-step simulation of complex hydrographic marbling patterns in GPU pixel shaders. (Top) This 1000×-zoomed fluid simulation of a Birdwing pattern is possible due to the Mixwell fluid solver's random access in space and time to advected texture displacements by modeling the Birdwing pattern using (Bottom-Right) cylindrical Reverse-Drift Functions (RDFs) that encode displacement fields of the Mixwell advection solver. (Bottom-Left) This high-quality 100 spp (samples/simulations per pixel) 4K image involves 3840×2160×100 = 829440000 ≈ 0.83 Gigasamples, each parallel sample evaluating a different long-time fluid integration problem. This frame renders in 20sec on a GeForce 4090 GPU, however 1spp HD previews run 400× faster at 20 FPS.

ACM Reference Format:
Doug L. James and Ethan James. 2026. Supplementary Material: Mixwell: Sharp 2D Fluid Brushes for Progressive Physics-Based Mixing. *ACM Trans. Graph.* 45, 4, Article 79 (July 2026), 7 pages. https://doi.org/10.1145/3811312

This document serves as the supplemental material for the paper *"Mixwell: Sharp 2D Fluid Brushes for Progressive Physics-Based Mixing."* We provide additional technical derivations and comparisons for the custom **one-step Newton-based advection solver**.

Authors' Contact Information: Doug L. James, Stanford University, Stanford, California, USA, djames@cs.stanford.edu; Ethan James, Independent Researcher, Half Moon Bay, California, USA, sparki262@gmail.com.



ACM 1557-7368/2026/7-ART79
https://doi.org/10.1145/3811312

1 One-step Advection in Sharp Mixwell Flow

We now revisit the Sharp Mixwell flow from a different perspective. Instead of repeatedly time stepping the Mixwell velocity field brush, we investigate the feasibility of using the semi-implicit Maxwell position-time solution directly to compute the final position in one step, given the initial position and elapsed time. For ideal flow around a cylinder, this map can be expressed in terms of Maxwell's classical time integral, which we invert using Newton's method and a custom monotone preconditioner. This Newton-based Mixwell advection scheme evaluates the same underlying flow as the Sharp Mixwell brush, but it supports arbitrarily long strokes and combs in a single step, without numerical time-stepping errors, providing fast random access to particle drifts in space and time. Unfortunately, we discover that it is generally slower and involves mathematical and physical singularities that make it less competitive for visual applications than the Mixwell brush techniques. Nevertheless, its implementation of `rdSegment` has been tested in GLSL, OpenCL, and HLSL, is fully functional, and produces visually good results

at a higher cost (see Figure 1). We include our exploration of this classical solution in this supplemental document for completeness.

1.1 Mixwell Time Inversion and One-step Advection

1.1.1 The Mixwell One-Step Advection Scheme. For a unit cylindrical tine moving at unit speed in the $\hat{X}$ direction, it takes time Δt to move a distance Δt in world space. Consider the cylinder's body frame, where it is located at the origin. The key advection task is to understand how a fluid particle with body-frame initial position (X_0, Y_0) moves in the flow after an arbitrary time $\Delta t \in \mathbb{R}$. The basic Mixwell advection scheme achieves this in *one step* as follows.

Mixwell Advection Scheme:

(1) Input: body-frame initial particle position, (X_0, Y_0), and Δt.
(2) If $Y_0 < 0$, rectify to the upper halfspace using $Y_0 \leftarrow |Y_0| \geq 0$.
(3) Calculate the trajectory's η_∞ using (8).
(4) Calculate the initial starting angle σ_0 using (13).
(5) Estimate X_{W0} by approximating (17).
(6) Obtain the start time $t_0 = -X_0 + X_{W0}$ from (20).
(7) Obtain the final time, $t_1 = t_0 + \Delta t$.
(8) Perform a time-inversion solve for the angle $\sigma_1 = \sigma(t_1, \eta_\infty)$.
(9) Evaluate the final position (X_1, Y_1) using $Y(\sigma_1, \eta_\infty)$ (22) then $X(\sigma_1, Y_1)$ (21).
(10) If the original Y_0 was rectified (since $Y_0 < 0$) in step 2, flip the sign on Y_1 back to the lower halfspace.

Since we only have Maxwell's time integral, $t = t(\sigma)$, we must perform a time inversion to estimate Maxwell's angle function, $\sigma(t)$. This is the hardest step, but one that is critical for one-step advection. For this core advection task, we now introduce a custom root-finding technique optimized to evaluate $\sigma(t, \eta_\infty)$ quickly and reliably.

1.1.2 On the robust estimation of $\sigma(t)$ with root finding. Given a particle at flow time t (on a curve η_∞), we want to know where—at what angle σ—the particle lies for a given η_∞ trajectory curve. We therefore seek to quickly and reliably estimate $\bar{\sigma}$ so that $t(\bar{\sigma}, \eta_\infty) = \bar{t}$ using a robust root-finding method on the singular function $t(\sigma, \eta_\infty)$ (plotted in Figure 2 (Top-Left)). Unfortunately, $t(\sigma, \eta_\infty)$ is costly to evaluate in a shader and is highly nonlinear in σ and η_∞, with singularities at $\sigma = \pm\pi/2$. These facts will make finding billions of roots robustly and quickly in a GPU shader problematic.

Robust root-finding methods exist for monotonic functions, and we have tested them on our problem. We found that the bisection (bracketing) methods and the improved generalizations [Brent 2013] were slow to converge and involved far too many iterations for our tolerances[1]. Faster converging methods, such as Newton's method, were simply unreliable given the wild derivatives, $t'(\sigma)$ (see Figure 2 (Bottom-Left)). Tighter root bracketing helps, but it is tricky to automate in a shader, in part because we need to compute roots in the vicinity of extreme singularities at $\sigma \pm \pi/2$. Fortunately, there is a faster and easier way for our specific function.

[1]Even using our monotone preconditioned function $M(t(\sigma))$, we found bisection took dozens of iterations for modest accuracy.

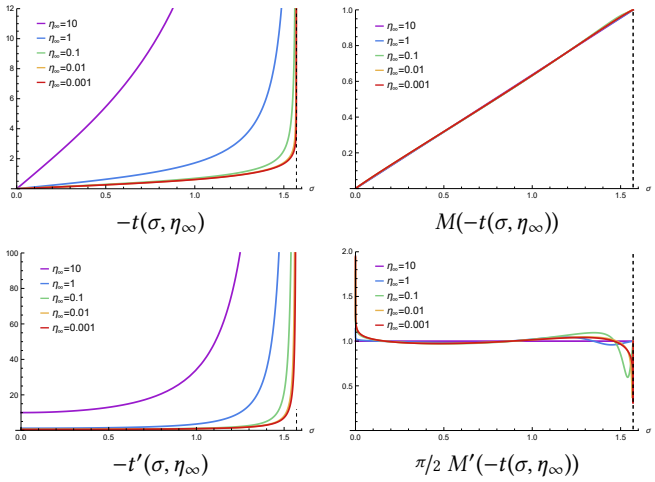


Fig. 2. **Monotone Preconditioning of Maxwell Time, $t(\sigma, \eta_\infty)$ for Root-ing Finding:** (Top-Left) Maxwell flow time vs $\sigma \in [0, \pi/2)$ angle is poorly conditioned and contains a (Bottom-Left) wide range of derivative values which complicates root-finding methods. In contrast, our monotone preconditioner, $M(\cdot)$, transforms the function to (Top-Right) be approximately a straight line, even for extremely small values of η_∞, with (Bottom-Right) a narrow range of derivative values around $2/\pi$ and bounded away from zero. Using this approach, Newton's method on $M(t(\sigma, \eta_\infty)) = M(\bar{t})$ converges rapidly and robustly for all η_∞ and $\bar{t}$ values of interest, thereby allowing us to evaluate $\sigma(t, \eta_\infty)$ needed by the Mixwell advection scheme. Note: we use negated time, $-t$, for plotting purposes only.

1.2 Monotone Preconditioning for Fast Time Inversion

Motivation. First, we observe that, for a fixed η_∞, our nonlinear function $t(\sigma) = t(\sigma, \eta_\infty)$ is monotonic. Second, we observe that methods like Newton's method can find the root in one step if the function is linear, and they can converge rapidly and reliably if the problem is weakly non-linear. Therefore, we seek to transform our monotonic function to make it "more linear" and, thus, easier to solve. We do this by designing custom monotonic transformations to (i) remove singularities, (ii) reduce the range of slopes, and (iii) make root bracketing unnecessary. The following is not a general turn-the-crank approach to root finding, but one designed to work extremely well for Maxwell's time integral $t(\sigma)$, which is the center of our universe for hydrographic marbling.

Key Idea. Given a scalar, monotone, univariate function, $M(t)$, the root $\bar{\sigma}$ of

$$t(\bar{\sigma}) = \bar{t} \tag{1}$$

is unchanged by a monotone nonlinear transformation, M:

$$M(t(\bar{\sigma})) = M(\bar{t}) \quad \Longleftrightarrow \quad f(\bar{\sigma}) = \bar{f}, \tag{2}$$

where $f(\sigma) \equiv M(t(\sigma))$ and $\bar{f} \equiv M(\bar{t})$. Therefore, we are free to design problem-specific monotone preconditioners, M, which transform our root finding problem into one that our chosen root finding method works with more efficiency and reliability. Specifically, we now show how to transform the graph of $f(\sigma)$ to approximately a straight line for which Newton's method works quickly and reliably.

Preconditioner Design. The construction of the custom monotone preconditioner for Maxwell time inversion is mathematically involved. For that reason, we refer the reader to Appendix A for details. The resulting preconditioner is shown in Figure 2 and can be efficiently calculated, as we now discuss.

1.3 GPU Evaluation

We now discuss an efficient implementation of all parts in a manner suitable for shader evaluation.

1.3.1 Fast Shader Evaluation of Elliptic Integrals. We need to rapidly evaluate both complete and incomplete elliptic integrals for real k ($m > 0$) and imaginary k ($m < 0$) values, respectively, and do so robustly and with high precision in shaders. Specifically, for infinite strokes, we need complete integrals ($\sigma = \pi/2$)

$$K\left[\frac{-4}{\eta_\infty^2}\right] = F\left(\frac{\pi}{2}\middle|\frac{-4}{\eta_\infty^2}\right), \qquad E\left[\frac{-4}{\eta_\infty^2}\right] = E\left(\frac{\pi}{2}\middle|\frac{-4}{\eta_\infty^2}\right) \tag{3}$$

whereas finite strokes use incomplete elliptic integrals,

$$F\left(\phi\middle|\frac{4}{\eta_\infty^2+4}\right), \qquad E\left(\phi\middle|\frac{4}{\eta_\infty^2+4}\right), \tag{4}$$

for general angle, $\phi = \sigma$. Therefore, it is sufficient to implement $F(\phi \mid m)$ and $E(\phi \mid m)$ for $\phi \in [0, \pi/2]$ and $m \in (-\infty, 1)$.

We find that fast GPU shader evaluation of these elliptic functions can be done using Carlson symmetric forms and associated iterative numerical methods [Carlson 1995, 2024], or AGM for complete elliptic integrals [Borwein and Borwein 1987]. Please see Appendix B for mathematical details. In practice, we observe an excellent performance of more than 30 billion incomplete elliptic functions per second on a NVIDIA GeForce 4090 (see Figure 10).

1.3.2 Monotone-Preconditioned Newton Solver. In practice, to find $\bar{\sigma}$ such that $t(\bar{\sigma}) = \bar{t}$, we use Newton's method on the monotone-preconditioned problem (2), $0 = f(\sigma) - \bar{f}$ where $f(\sigma) = M(t(\sigma))$ and $\bar{f} = M(\bar{t})$. The resulting iteration is

$$\sigma_{n+1} = \sigma_n - \frac{f(\sigma_n) - \bar{f}}{f'(\sigma_n)}. \tag{5}$$

Conveniently, it turns out that the gradient, $f'(\sigma)$, can be evaluated inexpensively by careful use of the chain rule and known subexpressions (see Appendix C for details). Finally, we have verified that the Newton solver provides a robust and fast evaluation of the solution $\sigma(t, \eta_\infty)$ for all parameter values, as needed for shader evaluation.

1.4 RDF for Finite Strokes (`rdSegment`)

1.4.1 Finite Stroke Flow. Consider the RDF of a radius-ε tine moving along a line segment from $\boldsymbol{a} \to \boldsymbol{b}$. Due to the presence of the tine, we only consider points outside it, and since it is an RDF, we simulate $\boldsymbol{a} \leftarrow \boldsymbol{b}$ and, therefore, consider the final location $\boldsymbol{b}$. We are free to transform $\boldsymbol{b}$ to the origin and consider the reverse motion vector, $\boldsymbol{M} = \boldsymbol{a} - \boldsymbol{b}$. We also translate $\boldsymbol{p} -= \boldsymbol{b}$ and store this initial position as $\boldsymbol{p}_0$ for later reverse-drift calculation. To transform to the canonical problem with a unit cylinder and X-axis flow, we must transform $\boldsymbol{p}$ and $\boldsymbol{M}$ by $1/\varepsilon$ and then rotate $\boldsymbol{p}$ so that $\boldsymbol{M}$ lies suitably along the X-axis. We can choose the flow time as positive $\Delta t = \|\boldsymbol{a} - \boldsymbol{b}\|/\varepsilon$ and the rotation as $\boldsymbol{R} = \boldsymbol{R}(\hat{\boldsymbol{M}})$ (which rotates $\hat{\boldsymbol{x}}$ to $\hat{\boldsymbol{M}}$). We perform canonical unit-cylinder advection with pre/post rotations to obtain

$$\boldsymbol{p}_{new} = \boldsymbol{R}\ \text{advect}(\boldsymbol{R}^T\boldsymbol{p}, \Delta t). \tag{6}$$

Finally, the reverse-drift displacement (with radius scaling) is

$$\boldsymbol{u} = \varepsilon\,(\boldsymbol{p}_{new} - \boldsymbol{p}_0 + \boldsymbol{M}). \tag{7}$$

Note that this RDF assumes that the tine is already inserted at $\boldsymbol{b}$ and that $\|\boldsymbol{p} - \boldsymbol{b}\| > \varepsilon$, which is not always desirable.

1.4.2 The `rdSegment` *RDF.* Our `rdSegment`$(\boldsymbol{p}, \varepsilon, \boldsymbol{a}, \boldsymbol{b})$ implementation assumes the insertion and removal of the tines by default (described in paper Appendix C), and therefore, it is valid for all $\boldsymbol{p}$ locations and does not have any tines left over for removal (see paper Figure 12). This is achieved by replacing the canonical advect$(\boldsymbol{p}, \Delta t)$ step with (i) tine insertion, (ii) advection, and (iii) tine removal. Finally, if you get confused about reverse drift and what goes where, keep in mind that the value of `rdSegment` at $\boldsymbol{p} = \boldsymbol{b}$ is simply $\boldsymbol{a} - \boldsymbol{b}$, i.e., the reverse drift moves $\boldsymbol{b}$ back to the start, $\boldsymbol{a}$.

2 Handling Singularities

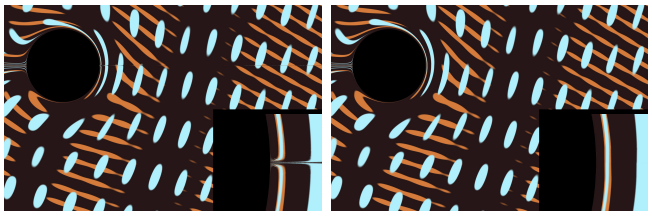
Singularities arise along the X axis in the advection calculations for two reasons: (1) the ill-conditioned legacy (σ, η_∞) coordinate system (recall Figure 3), and (2) the underlying physical singularity in the Maxwell time due to the stagnation points at $(X, Y) = (\pm 1, 0)$ (technically, hyperbolic fixed points). A robust practical implementation without ill pixels must address this dual singularity, as well as a few other points that we will briefly address now.

2.1 Patching the X-Axis's Dual Singularity

For the unit cylinder model problem, orbits with tiny $|\eta_\infty|$ along the X axis will suffer from coordinate singularities and unreliable Maxwell time and advection estimates. Ironically, many of these problems occur along the X axis, where the flow is smooth and, far from the disk, it approaches a pure translation (since $X \to -t$). Consider an initial position $\boldsymbol{P}_0 = (X, Y)$, and the displacement field $\boldsymbol{u}(X, Y) = (u_x, u_y) = (X_1 - X, Y_1 - Y)$ in smooth upwind regions ($\Delta t < 0$ and $X \geq 1$, or $\Delta t > 0$ and $X \leq 1$), or "post-crease" downwind areas ($\Delta t < 0$ and $X < X_{crease}$, or $\Delta t > 0$ and $X > X_{crease}$, where $X_{crease} = \Delta t + \text{sgn}(\Delta t)/2$). Observe that by symmetry, the u_x field is symmetric and smooth about $Y = 0$, while u_y is antisymmetric and smooth. A simple single-displacement sample patch that works well is to copy nearby u_x components and linearly interpolate u_y values as follows. If $|Y| \leq H$ where H is a small tolerance, e.g., $H = 0.05$, then we advect a nearby point $\boldsymbol{P}_{near} = (X, \text{sgn}(Y)H)$ to $\boldsymbol{P}_{new}$. We filter this nearby displacement $\boldsymbol{u} = p_{new} - p_{near}$ using

$$\tilde{\boldsymbol{u}} = (u_x, \frac{|Y|}{H}\, u_y), \tag{8}$$

which leaves u_x unchanged but scales u_y down to zero at $Y = 0$. We estimate the final advected point as $\boldsymbol{P}_0 + \tilde{\boldsymbol{u}}$. The patch is visually effective, as shown in Figure 3, and still incurs the cost of only a single advection call. Finally, we also considered a curvature correction to u_x, but it required an additional advection sample and was not visually significant in practice.

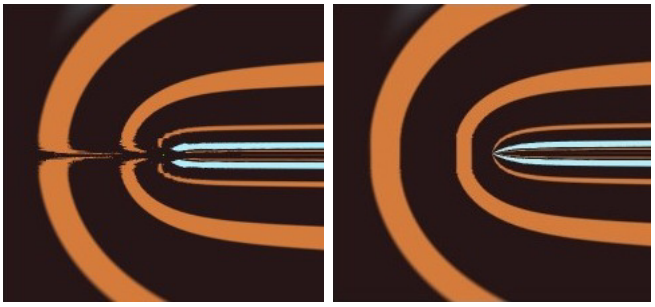


Before After

Fig. 3. **Patching the X-axis Singularity:** The degenerate (σ, η_∞) coordinate system, and the infinite Maxwell time, give rise to singularities along the X axis resulting in (Left) spurious displacements that produce inaccurate texture look-ups near the X axis shown here (extreme zoom shown inset). In contrast, (Right) our linear displacement patch yields smooth displacements across the X axis that greatly reduce texture lookup artifacts.

2.2 Cusp brush singularity

The `rdSegment` is a line-segment fluid brush that, through the process of tine removal, produces a cusp-like displacement field and strongly distorts the local circular flow neighborhood to a point. This process tends to introduce some additional artifacts along the X axis, localized near the brush tip, that reveal the dual singularity on the X axis and any shoddy Newton tolerances. Fortunately, the same displacement filtering patch can be used to improve the problem. The tip can also be cleaned up slightly by having the removal of the tines remove a small amount of paint: instead of mapping R to 0, we map $R + \varepsilon$ to 0. In our implementation, we use $\varepsilon = 0.0005$. The effect of cusp brush cleaning is shown in Figure 4. Finally, we observe that many patterns do not have unpaired `rdSegment` endpoints, so such artifacts would be rare even if left unpatched.



Before After

Fig. 4. **Patching the Cusp Singularity:** (Left) Extreme zoom of the `rdSegment` line-segment brush shows artifacts at the removal location along the X axis due to the coordinate system and Maxwell time singularities. (Right) Fortunately, these are greatly reduced by adjusting tine removal and patching the X axis using a simple displacement filter.

2.3 Large-time Evaluation

Complex patterns can involve numerous RDF line segments, which create a high computational load due to the infinite support of the RDFs, even when many are near zero amplitude. In addition, it is another source of artifacts when evaluating `rdSegment` for large distances and poorly conditioned (σ, η_∞) locations. Similar to comb modeling, we can optionally use associated SDFs to mask RDFs in regions of significance while maintaining C^0 continuity. However, even evaluating distant near-zero values can be costly due to the Newton solve for large t and/or coordinate system singularity–which can also introduce visual artifacts. Fortunately, there is an easy fix, which is to define a sufficiently large maximum time $t_{\text{cutoff}} > 0$ to perform unit-tine advection, after which we use a far-field advection model. The simplest version that works well is pure translation, since for large $|t|$ we have $t \to -X$. Therefore, our implementation's advection calculations only need to use Newton solves within the $|t| < t_{\text{cutoff}}$ event horizon, after which simple translations with $\Delta t = -\Delta X$ suffice; in our implementation, we use $t_{\text{cutoff}} = 20$. The results are visually indistinguishable and are therefore not plotted here.

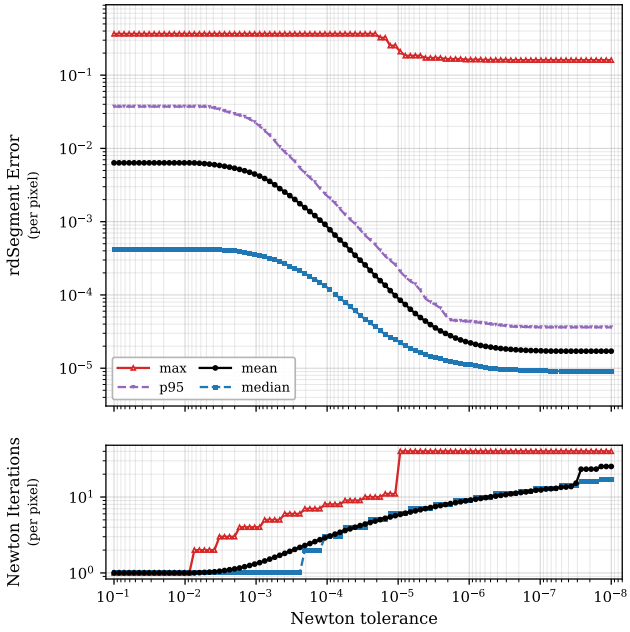


Fig. 5. **Accuracy of `rdSegment` RDF** using the Newton-solver-based evaluation, on the same float32 Houdini-Copernicus OpenCL implementation and test image as the adaptive midpoint case. Mean RDF error exhibits first-order convergence behavior with decreasing Newton tolerance TOL until the limits of the 32-bit floating point implementation are reached (near TOL$\approx 10^{-5}$); robust statistics (median, p_{95}) confirm the bulk of pixels converge cleanly, while the max curve is dominated by seam outliers near the brush tip. Newton iteration count grows logarithmically with decreasing TOL until a cap is reached (here set at 40).

3 Results and Discussion

The flow parameterization used by the Mixwell advection scheme is based on the classical (σ, η_∞) coordinate system, which is inherently ill-conditioned along the X-axis. This is a significant source of numerical singularities and approximation errors. Although we have found some practical workarounds, future work might identify better coordinate systems and formulations that result in analytically solvable integrals with well-conditioned numerical computations. Furthermore, it is not just the coordinate system that is problematic:

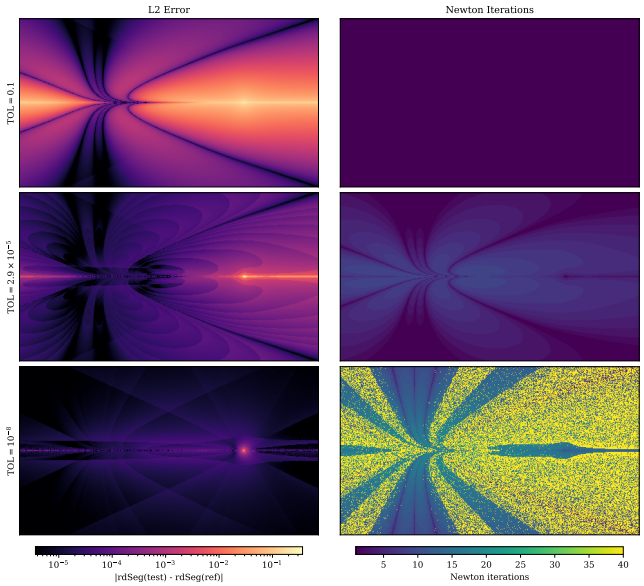


Fig. 6. **Spatial structure of `rdSegment` error and Newton iterations** at three representative tolerances from the convergence sweep of Figure 5. (Left column) Per-pixel Euclidean error against the reference solution; (Right column) Newton iteration count (capped at 40). At a loose tolerance (Top, TOL= 0.1), Newton terminates after a single iteration almost everywhere yet error is moderate outside the bright horizontal seam and brush-tip neighborhood. At a near-converged tolerance (Middle, TOL$\approx 3 \times 10^{-5}$), error is acceptable across the bulk of the image; however, residual error concentrates at the brush tip and along the seam, where the seam patch supplies only nominal accuracy. Pushing the float32 noise floor (Bottom, TOL= 10^{-8}), Newton iteration counts saturate at the 40-cap throughout most of the domain but brush-tip and seam-locality errors persist.

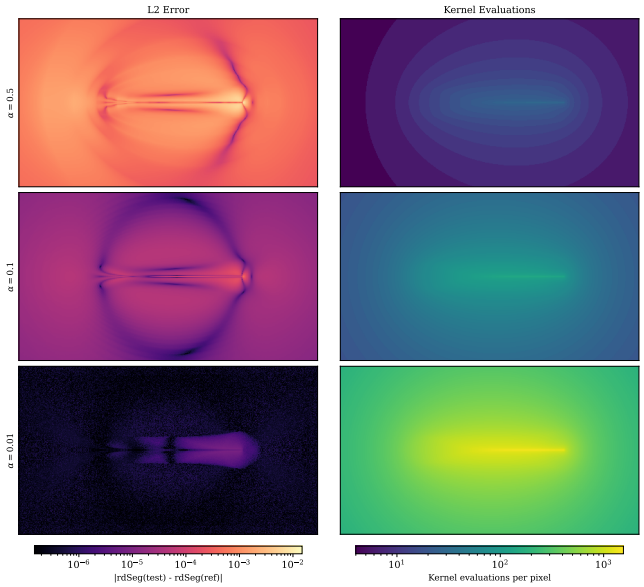


Fig. 7. **Spatial structure of `rdSegment` error and Mixwell brush kernel evaluations** at three representative step-size scalings α from the convergence sweep of paper Figure 13. (Left column) Per-pixel Euclidean error against the reference solution; (Right column) Kernel evaluation count per pixel. At a loose step (Top, $\alpha = 0.5$), the adaptive solver takes only ∼10 evaluations almost everywhere yet error is high across the bulk of the image, with the seam and brush-tip dominating. At the production setting (Middle, $\alpha = 0.1$), error is reduced by roughly an order of magnitude with cost rising to ∼50 evaluations per pixel; residual error concentrates at the brush tip and along the stroke. At a tight step (Bottom, $\alpha = 0.01$), error reaches the float32 noise floor outside the stroke and brush-tip neighborhoods, but kernel cost rises by another order of magnitude with a steep gradient toward the brush tip. Importantly, notice that the severe brush tip and seam errors that plague the Newton-based solver are markedly reduced or absent for the brush-based `rdSegment` solver.

Maxwell time itself is a singular quantity along the X axis due to the hyperbolic fixed points (stagnation points), and alternative time-like parameterizations are needed to better support advection.

These factors result in uneven convergence in the Newton solver on the `rdSegment` benchmark from the main paper, as plotted in Figure 5. The spatial structure of these errors concentrates near the brush tip and seam, regions that also require more Newton iterations to converge, as shown in Figure 6.

References

Jonathan M. Borwein and Peter B. Borwein. 1987. *Pi and the AGM: A Study in Analytic Number Theory and Computational Complexity.* Wiley, New York.

Richard P Brent. 2013. *Algorithms for minimization without derivatives.* Courier Corporation.

B. C. Carlson. 1995. Numerical computation of real or complex elliptic integrals. *Numerical Algorithms* 10, 1 (March 1995), 13–26. doi:10.1007/BF02198293

B. C. Carlson. 2024. Elliptic Integrals. In *NIST Digital Library of Mathematical Functions*, F. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier, B. I. Schneider, R. F. Boisvert, C. W. Clark, B. R. Miller, B. V. Saunders, H. S. Cohl, and M. A. McClain (Eds.). https://dlmf.nist.gov/ Section: 19.

A Designing a Monotone Preconditioner for Maxwell Time Inversion

We designed and explained our transformation incrementally by exploring a sequence of models (illustrated in Figure 8). Without loss of generality, we will assume $\eta_\infty > 0$ and plot nonnegative times $-t \geq 0$ for convenience.

Model A ($M_{\arctan}$). First, given the similar shape[2] to $-\tan(\sigma)$, we remove the singularities at $\pm\pi/2$ using $\arctan(-t)$, and map the result to $[-1, 1]$:

$$\tilde{f}(\sigma) = \frac{2}{\pi} \arctan(-t(\sigma)). \tag{9}$$

The bounded function $\tilde{f} \in [-1, 1]$ now makes it easier to bracket the root; however, it has a η_∞-dependent twist at the origin, which we can correct by replacing the slope of the function at $\sigma = 0$ with

[2]This similarity is no accident: far from the cylinder, we have $x \to 0$ and $Y \to \eta_\infty$, so $X \approx -t$ from (20). Then, (21) implies that $\sigma \approx \arctan(-t/\eta_\infty)$.

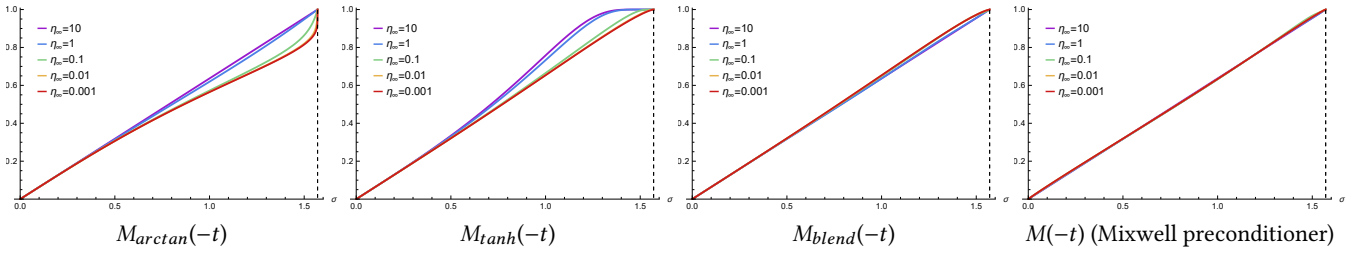


Fig. 8. **Monotone transformations of the root-finding problem,** $t(\sigma, \eta_\infty) = \bar{t}$**.**

$2/\pi$, achieved by dividing t by

$$S(\eta_\infty) \equiv \frac{\pi}{2}\tilde{f}'(0) = \frac{1}{2}\left(\eta_\infty + \frac{\eta_\infty^2 + 2}{\sqrt{\eta_\infty^2 + 4}}\right) \tag{10}$$

so that the monotone function is (see plot in Figure 8 (Far-Left)):

$$M_{\text{arctan}}(t) = \frac{2}{\pi}\arctan\left(-\frac{t}{S(\eta_\infty)}\right). \tag{11}$$

This yields the root-finding problem of the form

$$f(\sigma) \equiv \frac{2}{\pi}\arctan\left(-\frac{t(\sigma)}{S(\eta_\infty)}\right) \overset{!}{=} \bar{f} \equiv \frac{2}{\pi}\arctan\left(-\frac{\bar{t}}{S(\eta_\infty)}\right). \tag{12}$$

As shown in Figure 8, the function $f(\sigma)$ is virtually a straight line between $(\pm\pi/2, \pm 1)$ for higher values of $\eta_\infty > 0$, but it does deviate from linearity at low values of η_∞ (e.g., less variation than 16% at $\eta_\infty = 0.0001$) and has a steep derivative at the endpoints, $\sigma = \pm\pi/2$.

Model B (M_{tanh}). A monotone function that crushes the input into the $[-1, 1]$ range while flattening the derivative more at $\sigma = \pm\pi/2$ is the tanh function. Following a similar derivation, the monotone model (plotted in Figure 8 (Mid-Left)) is

$$M_{\text{tanh}}(t) = \tanh\left(-\frac{2}{\pi}\frac{t}{S(\eta_\infty)}\right).$$

Unfortunately, this function alone has too flat a derivative at low values of η_∞ at the endpoints $\pm\pi/2$ (which is detrimental for Newton's method), but it is complementary to the arctan model, so let us combine them.

Model C (M_{blend}). Blending these two functions with a to-be-determined blending parameter α, we have the candidate model

$$M_{blend}(t; \alpha) = \alpha\, M_{\text{arctan}}(t) + (1 - \alpha)\, M_{\text{tanh}}(t). \tag{13}$$

The two terms have complementary derivative (w.r.t. σ) behavior at $\sigma = \pm\pi/2$, with arctan producing steep derivatives, whereas tanh flattens the function and has a zero derivative. Therefore, to obtain 3-point derivative control, we can fit α so that the end-point derivatives are $2/\pi$ at $\sigma = \pm\pi/2$ for all $\eta = \eta_\infty$. Exploiting the convenient fact that $M'_{\text{tanh}} = 0$ there, it follows that the solution to the derivative constraint $2/\pi = \alpha\, M'_{\text{arctan}}(t(\pm\pi/2))$ is

$$\alpha^*(\eta) = \frac{2\eta\sqrt{\eta^2 + 4}}{\eta^2 + \eta\sqrt{\eta^2 + 4} + 2}, \quad \eta \geq 0. \tag{14}$$

The resulting preconditioned curve, $M_{blend}(t(\sigma); \alpha^*)$, plotted in Figure 8 (Mid-Right), matches the derivatives at three points and is nearly linear over a wide range of η_∞ values, with only a small fluctuation for η_∞ near zero due to the imperfect M_{tanh} fit there, which now improves with M_{tanha} (see Figure 9).

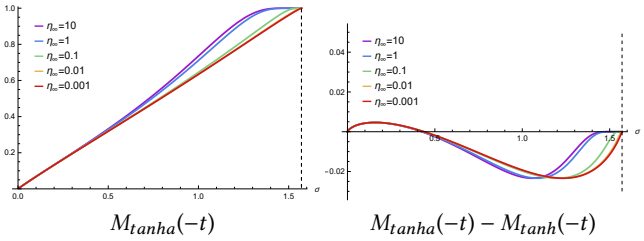


Fig. 9. **The M_{tanha} monotone preconditioner** is (Left) visually similar to M_{tanh} but (Right) has some subtle differences that ultimately improve the resulting blended fit with M_{arctan} to produce our final monotone preconditioner, M.

Model D (M_{tanha}). To further reduce fluctuations, we use the $\mathcal{L}_2$ error measure of the model $f(\sigma)$ deviation from a straight line:

$$E[f] = \int_0^{\pi/2}\left(f(\sigma) - \frac{2\sigma}{\pi}\right)^2 d\sigma. \tag{15}$$

We modify the $M_{\text{tanh}}(t)$ model by introducing an augmented Tanh function with a power parameter, a,

$$\tanh_a(t) \equiv \text{sgn}(t)\tanh(|t|^a) \tag{16}$$

and the slope correction parameter, s:

$$M_{\text{tanha}}(t; a, s) = \tanh_a\left(-\frac{2}{\pi}\frac{t}{s\, S(\eta_\infty)}\right) \tag{17}$$

(also with $M'_{\text{tanha}} = 0$ at $\sigma = \pm\pi/2$ so we can reuse α^* in (14)). Numerically minimizing $E[M_{\text{tanha}}(t(\sigma; \eta_\infty); a, s)]$ as $\eta_\infty \to 0$ yielded the fit parameters $a^* \approx 0.956764$ and $s^* \approx 1.05782$, with a final error E^* approximately 5% of the original M_{arctan} model.

The Mixwell Monotone Preconditioner, $M(t)$. Putting it all together, the final blended model is our Mixwell monotone preconditioner:

$$\boxed{M(t) = \alpha^*(\eta)\, M_{\text{arctan}}(t) + (1 - \alpha^*(\eta))\, M_{\text{tanha}}(t; a^*, s^*)} \tag{18}$$

and is plotted in Figures 2 and 8. This model crushes the Maxwell time function to a straight line and allows us to perform a time inversion to estimate the Maxwell flow angle, σ, quickly using Newton's method for all parameter values of interest, often utilizing only a single Newton iteration.

B GPU Evaluation of Elliptic Functions using Carlson Symmetric Forms

Carlson symmetric forms. For numerical reasons, it is convenient to express our integrals in terms of Carlson symmetric forms of elliptic integrals, a small canonical set of elliptic integrals to which all others may be reduced [Carlson 1995, 2024]. In our case, we only require two such Carlson forms,

$$R_F(x,y,z) = \frac{1}{2}\int_0^\infty \frac{dt}{\sqrt{(t+x)(t+y)(t+z)}}, \tag{19}$$

$$R_D(x,y,z) = \frac{3}{2}\int_0^\infty \frac{dt}{(t+z)\sqrt{(t+x)(t+y)(t+z)}}, \tag{20}$$

in order to express our incomplete elliptic integrals:

$$F(\phi \mid m) = \sin\phi\ R_F(\cos^2\phi, 1 - m\ \sin^2\phi, 1) \tag{21}$$

$$E(\phi \mid m) = \sin\phi\ R_F(\cos^2\phi, 1 - m\ \sin^2\phi, 1) \tag{22}$$

$$-\frac{1}{3}m\sin^3\phi\, R_D(\cos^2\phi, 1 - m\ \sin^2\phi, 1). \tag{23}$$

Note that to evaluate $F(\phi \mid m)$ and $E(\phi \mid m)$, we only need to evaluate two Carlson forms if we exploit subexpressions, since

$$E(\phi \mid m) = F(\phi \mid m) - \frac{1}{3}m\sin^3\phi\, R_D(\cos^2\phi, 1 - m\ \sin^2\phi, 1),$$

which gives a roughly 50% performance boost to our solver.

Carlson Numerical Approximations. We follow Carlson [1995] and use the so-called duplication theorem to iteratively approximate R_F and R_D. The duplication theorem for R_F observes that

$$R_F(x,y,z) = R_F\left(\frac{x+\lambda}{4}, \frac{y+\lambda}{4}, \frac{z+\lambda}{4}\right) \tag{24}$$

$$\lambda = \sqrt{xy} + \sqrt{yz} + \sqrt{zx}. \tag{25}$$

Rapid convergence for recursive evaluation leads to a fast evaluation method. Specifically, we implement Carlson's R_F function using his algorithm on p.16 of Carlson [1995], and Carlson's R_D function using the algorithm on p.20 of Carlson [1995]. Using these implementations in GLSL or OpenCL, we evaluate the elliptic integrals in (3) and (4). See Figure 10 for Shadertoy plots of Carlson and elliptic functions.

C Newton Gradient Evaluation, $f'(\sigma)$

Given $f(\sigma) = M(t(\sigma))$, we need its derivative for Newton's method, which follows from the chain rule:

$$f'(\sigma) = M'(t(\sigma))\ t'(\sigma). \tag{26}$$

The first derivative, $M'(t)$, only involves derivatives of expressions involving tanh and $\tanh_a$ functions:

$$M'(t) = \alpha^*(\eta)\, M'_{\text{arctan}}(t) + (1 - \alpha^*(\eta))\, M'_{\text{tanha}}(t; a^*, s^*) \tag{27}$$

where α^* is given by (14) and

$$M'_{\text{arctan}}(t) = -\frac{2}{\pi\, S\left(1 + \frac{t^2}{S^2}\right)}, \tag{28}$$

$$M'_{\text{tanha}}(t; a, s) = -ac|ct|^{a-1}\text{sech}^2\left(|ct|^a\right),\ c = \frac{2}{\pi s\, S(\eta_\infty)}. \tag{29}$$

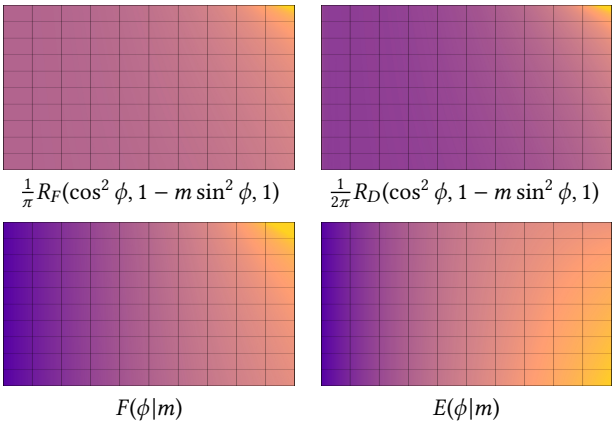


Fig. 10. **Shader evaluation of Carlson and Elliptic special functions** for the entire range of values $(\phi, m) \in [0, \pi/2] \times (0, 1)$ (where $m = 4/(4 + \eta_\infty^2)$) used in finite stroke evaluation. Using Carlson iterations, our GLSL implementation can evaluate each of these functions in GLSL at effectively 40000 FPS for a 1200×675 image using a low Carlson TOL= 0.0001.

The $t'(\sigma)$ derivative in (26) should not be obtained by differentiating $t(\sigma, \eta_\infty)$ in (20), since we already know the answer as the integrand of the original defining Maxwell time integral (12):

$$t'(\sigma) = -\frac{1}{4}\frac{\left(\eta_\infty + \sqrt{\eta_\infty^2 + 4\cos^2\sigma}\right)^2}{\cos^2\sigma\sqrt{\eta_\infty^2 + 4\cos^2\sigma}}, \quad \text{where} \quad \eta_\infty > 0. \tag{30}$$