Eurographics Symposium on Geometry Processing 2015
Mirela Ben-Chen and Ligang Liu
(Guest Editors)

Volume 34 (2015), Number 5

Unconditionally Stable Shock Filters for Image and Geometry Processing

F. Prada and M. Kazhdan

Johns Hopkins University, Baltimore MD

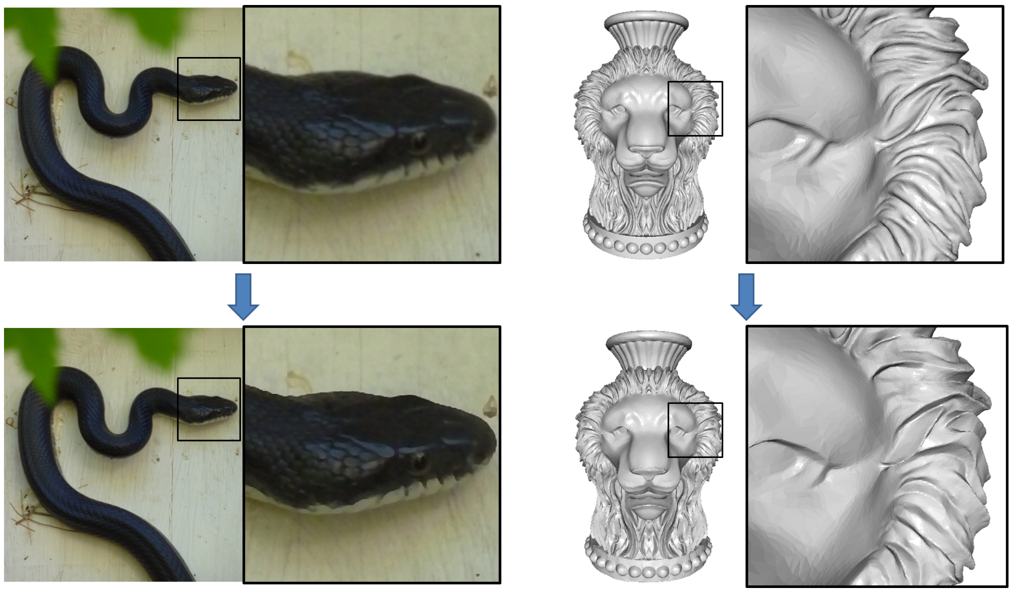

Figure 1: *Input (top) and filtered data (bottom) obtained using our Shock Filter advection. Applied to the colors of an image (left) the filtering process enhances the edges. Applied to the normals (right) the filter accentuates creases in the geometry.*

Abstract

This work revisits the Shock Filters of Osher and Rudin [OR90] and shows how the proposed filtering process can be interpreted as the advection of image values along flow-lines. Using this interpretation, we obtain an efficient implementation that only requires tracing flow-lines and re-sampling the image. We show that the approach is stable, allowing the use of arbitrarily large time steps without requiring a linear solve. Furthermore, we demonstrate the robustness of the approach by extending it to the processing of signals on meshes in 3D.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Hierarchy and geometric transformations I.4.3 [Image Processing and Computer Vision]: Enhancement—Sharpening and deblurring

1. Introduction

Over the past decades, numerous filters have been proposed for enhancing images and geometry. An important aim of many of these filters is edge-enhancement – generating a new image or surface that accentuates the boundaries separating smooth regions. In this work, we revisit the early *Shock Filters* proposed by Osher and Rudin [OR90] for image-processing and provide an interpretation of the original filter that results in a simple and robust implementation that is easily generalized to geometry-processing.

In Osher and Rudin's original implementation, the filtering is performed by evolving the image under a non-linear PDE that preserves extrema, encourages sharp discontinuities at edges, and becomes smooth everywhere else. Due to the non-linear nature of the PDE, time-integration requires careful implementation so as to avoid instabilities. As we show, the PDE used for Shock Filters can be understood as describing the advection of a signal along a flow. Thus, we are able to compute an unconditionally stable solution, taking arbitrarily large steps for the time-integration, without having to solve a linear system. Furthermore, the ability to implement advection over meshes makes our approach extend easily to signals on surfaces in 3D.

Casting Shock Filters in terms of advection provides several important contributions, including: A robust and simple implementation; An extension to filtering signals on surfaces; And a formulation that facilitates analysis.

We discuss these contributions in the remainder of the paper. We begin with a brief review of Shock Filters, as well as other techniques for enhancing images and geometry (§2). We present the relationship between Shock Filters and advection, providing an analysis of the underlying PDE (§3). We describe how the advection formulation can be extended to surfaces, enabling both the processing of signals over meshes as well as the processing of the geometry itself (§4). We discuss performance and compare to related methods (§5) and summarize our contributions (§6).

2. Review

Introduced more than two decades ago, *Shock Filters* [OR90] formulate image-processing as a PDE which evolves the image towards a steady-state solution which is piecewise smooth with sharp discontinuities (shocks) forming along edges. The PDE holds extrema fixed and evolves concave-up (resp.concave-down) regions towards their local minima (resp. maxima).

Shock Filters have been made more robust in recent work and have included an unconditionally stable implementations that uses an implicit time-integrator to solve the PDE coupled with anisotropic diffusion [AM94], regularized implementations that are more stable in the presence of signal noise [GSZ02], and fast implementations using smoothed histograms [VAKMS12].

In addition to Shock Filters, a variety of methods for edge-aware filtering have been proposed. Anisotropic diffusion [PM90, ALM92] smooths the image while constraining the diffusion not to cross edges. Bilateral filtering [TM98] replaces a pixel with the weighted average of its neighbors, adapting the weights so that more smoothing occurs between pixels on the "same side" of an edge. Laplacian sharpening [BCCZ08] amplifies high-frequency content. And, L_0 gradient minimization [XLXJ11] solves for the image which matches the input but has sparse gradients.

Though initially proposed for image-processing, many of these approaches have since been adapted to editing surface geometry, including anisotropic diffusion of geometry [CDR00, BXBT02] and normals [TWBO02], bilateral mesh denoising [FDCO03], and Laplacian/spectral sharpening [VL08, CLB*09].

More recently, there has also been a significant body of work that leverages priors, learned either from the image itself, frames of a video, or a large database of images, to perform edge-aware processing [Fat07, SXS08, SLJT08, FF10].

Though unconditionally stable solutions for PDEs have been proposed in numerous image-processing applications, these are often obtained through the solution of a large linear system. In contrast, our approach only requires tracing values along flow-lines. Thus, much like Stam's Unconditionally Stable Fluids [Sta99, Sta03], our method can use arbitrarily large time-steps and generalizes to meshes.

3. Shock Filters and Advection of Images

In this section we describe the relationship between Shock Filters and advection and show how this leads to a simple, unconditionally stable, implementation.

3.1. Shock Filters

The partial differential equation describing the time-evolution of a signal I under the Shock Filter is given by:

$$\frac{dI_t}{dt} = -\|\nabla I_t\| \cdot F(\mathcal{L}(I_t)) \qquad \text{s.t.} \quad I_0 = I \tag{1}$$

Here $\mathcal{L}$ is an edge-detector such as the Laplacian or the second derivative of I_t along the direction of the gradient:

$$\mathcal{L}(I_t) = \Delta I_t \qquad \text{or} \qquad \mathcal{L}(I_t) = (\nabla I_t)^T H_t(\nabla I_t)$$

(with H_t the Hessian of I_t) and F is a sign-preserving reweighting of the edge-detection response.

Intuitively, the incorporation of the $\|\nabla I_t\|$ term ensures that the time derivative is zero when the gradient vanishes, so that local extrema are fixed under the evolution. And, the sign-preservation of F ensures that the time derivative is negative (resp. positive) in regions that are concave-up (resp. concave-down) so that these regions evolve towards their local minima (resp. maxima).

3.2. Advection

Re-writing Equation 1 we get:

$$\frac{dI_t}{dt} = -\left\langle \frac{F(\mathcal{L}(I_t))}{\|\nabla I_t\|} \nabla I_t, \nabla I_t \right\rangle \qquad \text{s.t.} \quad I_0 = I.$$

Thus, the PDE governing Shock Filters is precisely the PDE describing the advection of the signal I along the vector field $\vec{u}_t = \frac{F(\mathscr{L}(I_t))}{\|\nabla I_t\|}\nabla I_t$. (The relationship between shock filters and advection in 1D is briefly alluded to in [AM94], though implications are not pursued.)

In particular, setting F to be the identity and $\mathscr{L}$ to be the edge-detector $\mathscr{L}(I_t) = \frac{(\nabla I_t)^T H_t(\nabla I_t)}{\|\nabla I_t\|}$ we obtain a particularly simple expression for the PDE:

$$\frac{dI_t}{dt} = -\frac{1}{2}\left\langle \nabla\|\nabla I_t\|^2, \nabla I_t \right\rangle. \quad (2)$$

This gives a simple, trivially parallelizable, implementation that approximates running a Shock Filter for time t by linearizing the problem and back-tracing along flow-lines:

```
SignalProcess( I , t )
1   P ← ||∇I||^2                         compute the potential
2   u⃗ ← ½∇P                                   compute the flow
3   return Advect*( I , −u⃗ , t )                       advect
```

Figure 2 compares the results of the filtering on an image of a market scene, with zoom-ins highlighting details in (a) the input, (b) the steady-state solution of the Shock Filter PDE [OR90], and (c) the steady-state solution of our advection. As the images show, our results are hard to distinguish from those obtained by solving the Shock Filter PDE.

Properties

Formulating Shock Filters in terms of advection provides an interpretation of the filtering process in terms of the potential $P = \|\nabla I\|^2$. Specifically, by flowing the image values along the gradients of this potential, we fix the signal values at the local minima of the potential (incuding the points where the gradient of I vanishes) and advect the values everywhere else towards the local maxima, thereby pushing the change in pixel values towards points where the gradient magnitude is already extremal. Additionally, this formulation makes several properties of Shock Filters easier to see:

Unconditional Stability Since advection sets the values in the new image by sampling the old image, instabilities cannot arise, regardless of the step-size t.

Preservation of Extrema Since $\vec{u}$ vanishes when the gradient is zero, values of local minima/maxima are not changed by the flow. Furthermore, assuming that the flow is bounded, $\|\vec{u}\| \le \gamma < \infty$, we know that after advecting for time ε, the value at a point p must be sampled from a disk of radius at most $\varepsilon \cdot \gamma$ about p. Thus, since the value at a local extremum is fixed under the flow and since the values of neighboring points are obtained by sampling from a neighborhood of the local extremum, the local extremum will continue to be a local extremum (with the same value) after advection.

Total Variation Preservation (TVP) in 1D Letting $\pi_t(p)$ be the map giving the position of a point that starts at p and flows with $\vec{u}_t$ for time t, we get:

$$I_t \circ \pi_t = I.$$

Then, the total variation becomes:

$$\begin{aligned} TV(I_t) &= \int \left\| \nabla I_t|_p \right\| dp \\ &= \int \left|\det\left(d\pi_t|_p\right)\right| \cdot \left\| \nabla I_t|_{\pi_t(p)} \right\| dp \\ &= \int \left\| \nabla (I_t \circ \pi_t)|_p \right\| dp \\ &= \int \left\| \nabla I|_p \right\| dp = TV(I) \end{aligned}$$

(with the second equality deriving from a change of variables and the fact that the determinant of π_t cannot be negative).

While the advection formulation makes it clear that Shock Filtering is TVP in 1D, it also suggests that it will not be in 2D. In particular, the advected image satisfies:

$$\nabla I|_p = \left(d\pi_t|_p\right) \circ \nabla I_t|_{\pi_t(p)}.$$

However, the condition that should be satisfied to ensure preservation of total variation is:

$$\left\|\nabla I|_p\right\| = \left|\det\left(d\pi_t|_p\right)\right| \cdot \left\|\nabla I_t|_{\pi_t(p)}\right\|.$$

Less formally, advection acts on gradients by scaling them by the reciprocal of the differential change of *length* along the flow-line. In contrast, total variation preservation requires that gradients be scaled by the reciprocal of the differential change in *area*. In general, these two definitions differ, with agreement implying that flow-lines are parallel.

3.3. Defining the Flow

We conclude by discussing the definition of the flow $\vec{u}$. We defer details of the spatial discretization to Appendix A.

Multi-Channel Signals Though one can apply the advection to multi-channel signals by processing the channels independently, we have found that it is often preferable to couple the channels so that the values in the different channels evolve together. Our advection formulation ensures that this happens so long as we use the same flow field $\vec{u}$ to define the filtering. In particular, given a multi-channel image $I(p) = \left(I^1(p), \ldots, I^d(p)\right)$, we define the flow field as:

$$\vec{u} = \frac{1}{2}\sum_{j=1}^{d} \nabla \left\|\nabla I^j\right\|^2.$$

Multi-Step Advection Our implementation only approximates Shock Filters as we fix the flow, setting $\vec{u}_t \equiv \vec{u}_0$. A more faithful implementation can be obtained by performing multiple (short) advection steps and updating the flow to $\vec{u}_t \equiv \frac{1}{2}\nabla\|\nabla I_t\|^2$ after each step.

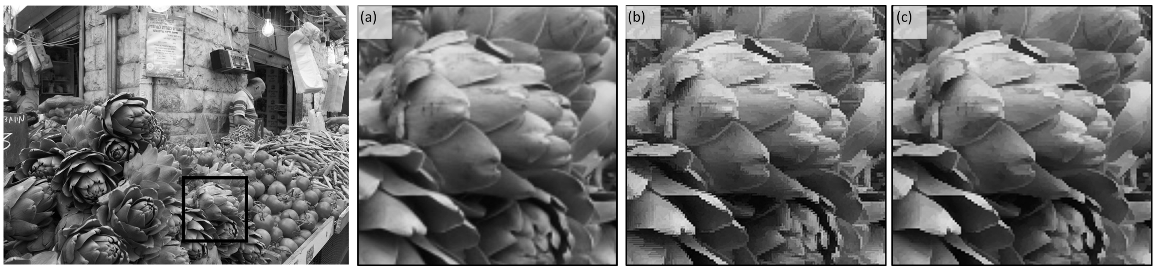

Figure 2: *Comparison of the steady-state solution ($t \to \infty$) obtained by filtering an image of a market scene (left). Zoom-ins show: (a) the initial image, (b) the solution to the Shock Filter PDE, and (c) our Shock Filter advection.*

While the naive approach for generating I_{t+1} is to step back along $\vec{u}_t$ for a unit of time and sample image I_t at the back-traced position, we have found that repeated sampling leads to undesirable smoothing. Instead, we compose the flows, taking a unit step back along $\vec{u}_t$, then a unit step back along $\vec{u}_{t-1}$, etc., and only then sampling the input image, I_0, at the back-traced position. (Noting that the first t steps taken in estimating $I_{t+1}(p)$ are the same t steps taken in estimating $I_t(p)$, we obtain an efficient implementation by storing and updating the source position for each pixel p.)

Figure 3 compares these implementations, showing results with the updated flow $\vec{u}_t \equiv \frac{1}{2}\nabla\|\nabla I_t\|^2$, using naive (top) and composed (middle) implementations, as well as the results using the fixed flow $\vec{u}_t \equiv \vec{u}_0$ (bottom). As the figure shows, the naive implementation loses fine feature detail over time, while results with flow composition are hard to distinguish from those obtained using a fixed flow field.

Robustness to Noise As pointed out in [GSZ02], a challenge of using Shock Filters is that the definition is sensitive to noise. Using our advection formulation, this instability can be removed by smoothing the flow field $\vec{u}$ before performing the advection. (Visualizations are shown in the next section, for applications in geometry processing.)

4. Extending to Surfaces

Using the **SignalProcess** algorithm we can also apply Shock Filters to processing signals on a surface by estimating the gradient to define the potential and the flow and then advecting. (Discretization details are described in Appendix A.)

Figure 4 shows a visualization of advection in filtering the texture on the "Pleo" model. As with images, advection produces a texture with enhanced edges, converging to a stable steady-state solution as the flow time goes to infinity.

4.1. Geometry Processing

Treating the normals of a mesh as the signal, we can sharpen the geometry by first sharpening the normals using the **SignalProcess** algorithm and then solving for the new geometry that best fits the normal field.

Following [YZX*04], we fit a surface to the sharpened normal field $\vec{N}$ by solving a screened Poisson equation:

1. Compute the gradient of the initial coordinate function c_0 (the function assigning a 3D position to each vertex):
$$\vec{v} = \nabla c_0$$
2. Project out the component in the normal direction:
$$\vec{w} = \vec{v} - \langle \vec{v}, \vec{N} \rangle \cdot \vec{N}$$
3. Solve the screened Poisson equation to get the positions of the surface whose normals match the target normals:
$$c_1 = \arg\min_c \left(\varepsilon \cdot \|c - c_0\|^2 + \|\nabla c - \vec{w}\|^2 \right)$$
$$\Downarrow$$
$$c_1 = (\varepsilon - \Delta)^{-1} (\varepsilon \cdot c_0 + \nabla \cdot \vec{w})$$
(In our evaluations we fix the regularization term $\varepsilon = 1$.)

Figures 5 and 6 show the results of our Shock Filter advection on the "David" and "Chinese Dragon" models. Figure 5 shows the results of the sharpening with increasing time steps while Figure 6 shows the steady-state solution obtained by performing increasing number of passes of umbrella smoothing on the potential before advecting. As the figures show, the approach is stable, providing more pronounced sharpening as advection time is increased. (With the exception of the results in Figure 2 we use a single smoothing pass in our evaluations.)

Remark Using the normals (N_x, N_y, N_z) as the signal, the potential $(\|\nabla N_x\|^2 + \|\nabla N_y\|^2 + \|\nabla N_z\|^2)$ is the sum of the squares of the principal curvatures. Thus, in the context of geometry processing, Shock Filters advect the normals so that discontinuities occur at extrema of total curvature.

Remark In sampling the normals at the back-traced positions we can either use nearest-point sampling (copying the normal from the triangle containing the back-traced position) or linear interpolation (initially averaging the per-face normals to the vertices to get a per-vertex signal and then interpolating using the barycentric coordinates of the back-traced position). As shown in Figure 7, linear interpolation

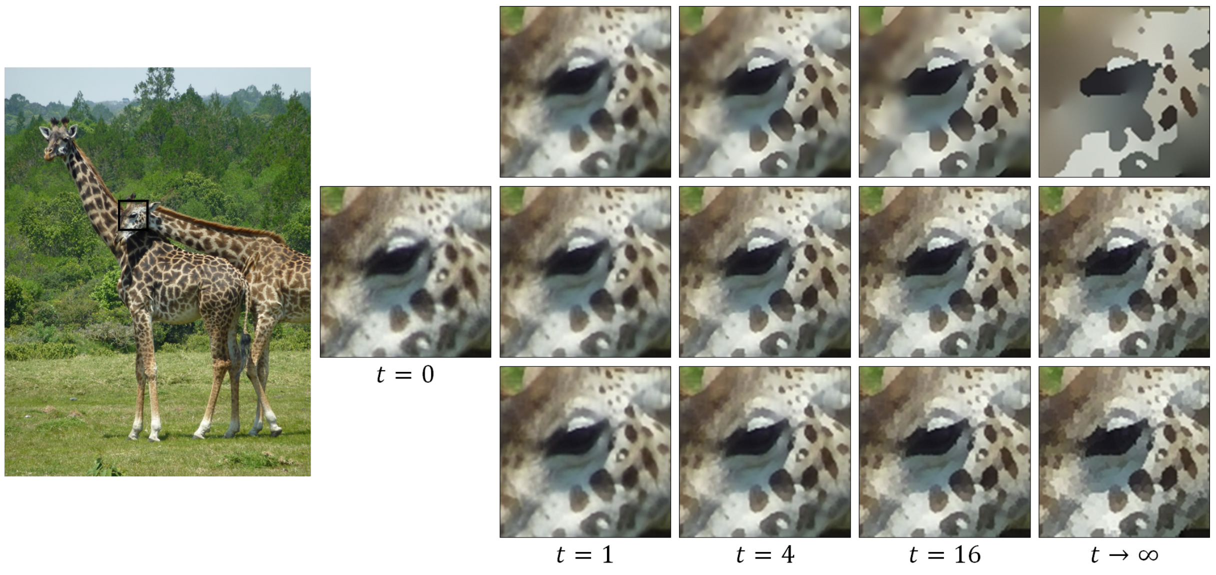

Figure 3: *Comparison of the advection solution obtained for different times steps: Zoom-ins show results with naive sampling (top), flow composition (middle), and a fixed flow field (bottom).*

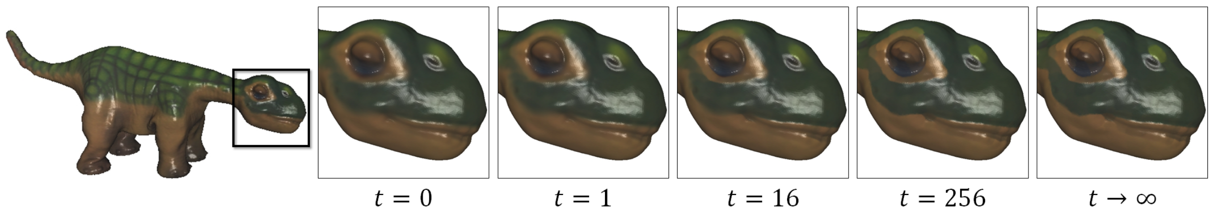

Figure 4: *Shock Filter advection used to sharpen a signal on a triangle mesh: Showing results for different time steps t.*

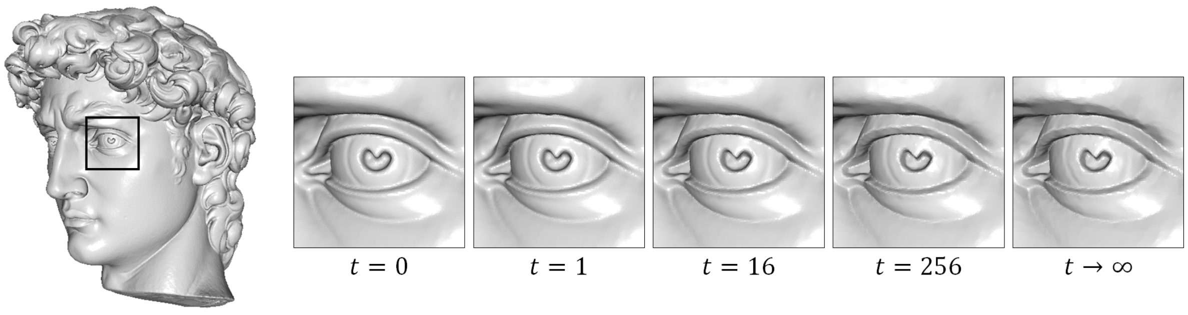

Figure 5: *Shock Filter advection used to sharpen geometry: Showing results for different time steps.*

has the desirable property of removing high-frequency noise from the signal and we use this approach in our evaluations.

This is further highlighted in Figure 8 which shows the results of Shock Filter advection applied to the "Monkeys" dataset [Pau12]. In this example, sampling with linear interpolation removes the high-frequency noise resulting from the structure-from-motion reconstruction and the steady state solution robustly advects the normals so that curvature concentrates along the edges of the model.

5. Performance

In this section we evaluate the performance of our method. All results were obtained using an Intel Core i7-4710MQ processor with 16 GB of RAM, parallelized across eight (hyper) threads using OpenMP [DM98]. For geometry-

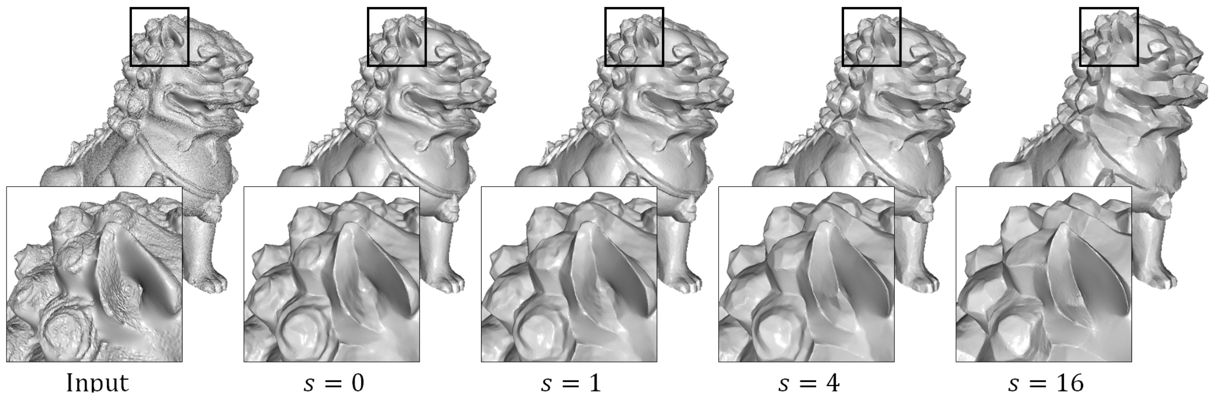

Figure 6: *Shock Filter advection used to sharpen geometry: Showing the solution with increasing number of smoothing passes.*

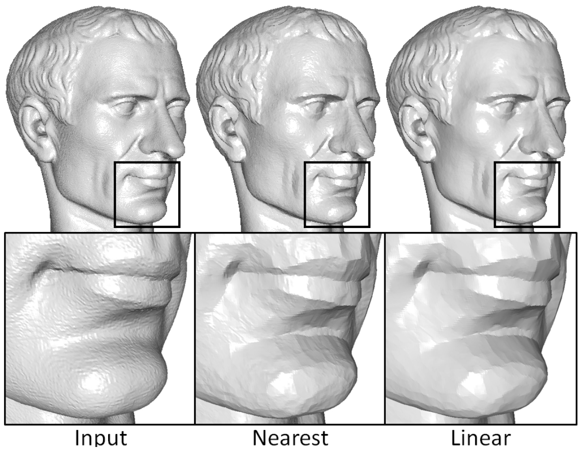

Figure 7: *Comparison of geometry processing with Shock Filter advection: Showing the original geometry (left), results with nearest-point sampling (middle), and results with linear interpolation (right).*

processing applications where we fit vertex positions to the sharpened normals, we solve the linear system using 25 iterations of a diagonally preconditioned Conjugate Gradients solver, initialized with the original vertex positions.

5.1. Image-Processing

We begin by considering applications of our advection formulation of Shock Filters to image-processing.

As demonstrated in Section 3, Osher and Rudin's PDE and our advection formulation provide different implementations of the same image sharpening technique. An advantage of the advection formulation is that the computational effort in determining a pixel's value adapts to the length of the stream-line traversed when back-tracing the flow. In contrast, the PDE formulation requires the same computational effort of each pixel.

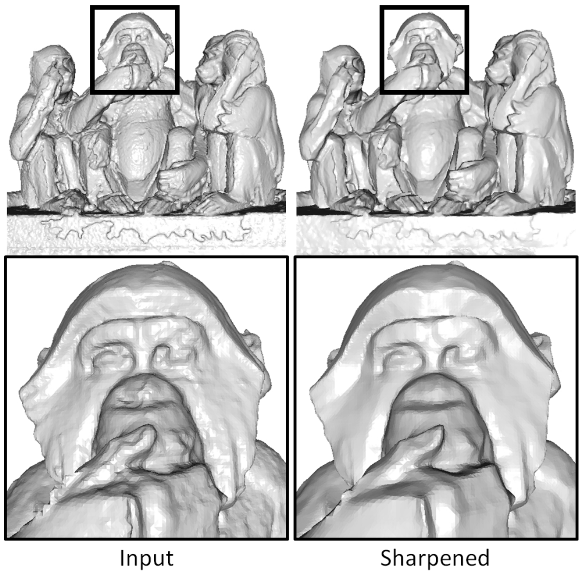

Figure 8: *Shock Filter advection applied to a noisy surface: Showing the original geometry (left) and the steady state ($t \to \infty$) solution (right).*

We evaluate the benefit of using advection by measuring the steady-state convergence rate, $\|I_t - I_0\|/\|I_\infty - I_0\|$, as a function of the time required to compute I_t.

Figure 9 plots the convergence rates of the two methods for the 2848×2136 image in Figure 2, sampling at time steps $t = 2^k$ with $k \in \{1, 2, \cdots, 9\}$ and approximating I_∞ by I_{2048}. As the plots show, while advection is slightly less efficient than the PDE solution at short time steps, the ability to adapt computation to stream-line length makes it significantly faster when computing the steady-state solution.

Remark As the flow $\vec{u} = \frac{1}{2}\nabla P$ is curl-free, stream-lines cannot form cycles and we expect the back-traced paths to eventually terminate at sources. This is corroborated by Figure 9

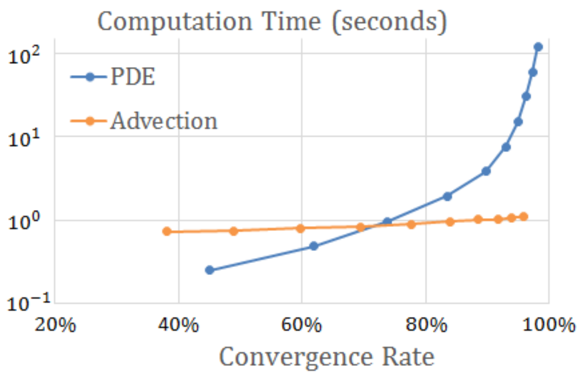

Figure 9: *Comparison of steady-state convergence times, as a function of convergence rate, for solutions obtained using the Shock Filter PDE and our Shock Filter advection.*

	Time Step (t)					
	1	4	16	64	256	1024
Naive	0.4	1.3	5.0	19.8	79.0	310.1
Composed	0.4	1.2	4.7	20.0	77.6	308.6
Fixed	0.4	0.4	0.5	0.6	0.6	0.6

Table 1: *Processing times for image sharpening (in seconds) using naive sampling, flow composition, and a fixed flow field, as a function of time step.*

which shows that the run-time of our (fixed flow) implementation approaches a constant as flow time increases.

In Figure 3 we showed that though our implementation uses a fixed flow field, it generates results that are nearly identical to those obtained when the flow field is updated at each time step. We have chosen to use the fixed flow field because we have found that most stream lines tend to be short so that run-time grows sub-linearly with the flow duration. In contrast, updating the flow at each time-step gives run-times that are linear in the duration of the flow.

This can be seen in Table 1 which gives the run-times for sharpening the 1368×1824 image in Figure 3, demonstrating the sub-linear complexity (in flow time) of the fixed flow-field implementation. In contrast, updating the flow-field (regardless of whether it is done by naively sampling from the previous solution or by using the composite flow and sampling from the input) has linear complexity.

Comparison to filtering/diffusion methods As with Shock Filters, edge-preserving image filtering operators (e.g. [PM90, ALM92, TM98, TT99]) encourage the gradient field to concentrate along sharp edges in the input. Away from these edges, however, they tend to smooth out the image and are commonly used to extract texture layer hierarchies [BPD06, FFDLS08, SSD09, CM11]. Figure 10 highlights this distinction by comparing our method to existing edge-preserving operators. While all methods preserve the silhouette boundaries like the slats and the saucer, the filtering approaches tend to lose detail in areas like the reflection, whereas Shock Filtering tends to enhance it.

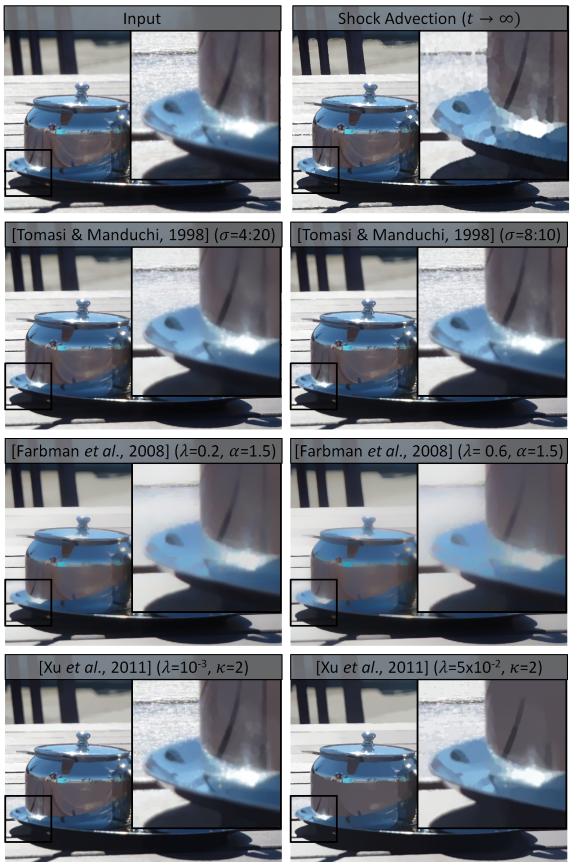
Input
Shock Advection ($t \to \infty$)
[Tomasi & Manduchi, 1998] (σ=4:20)
[Tomasi & Manduchi, 1998] (σ=8:10)
[Farbman et al., 2008] (λ=0.2, α=1.5)
[Farbman et al., 2008] (λ= 0.6, α=1.5)
[Xu et al., 2011] (λ=10^{-3}, κ=2)
[Xu et al., 2011] (λ=5×10^{-2}, κ=2)

Figure 10: *Comparison of our method (top right) to state-of-the-art techniques in edge-aware filtering. Labels indicate the parameter setting used to generate the different images.*

	Time Step (t)						
	1	4	16	64	256	1024	4098
Color	0.2	0.2	0.2	0.3	0.3	0.4	0.5
Positions	1.7	1.8	1.9	2.2	2.7	3.2	3.5

Table 2: *Processing times (in seconds) for sharpening geometry color and positions, as a function of time step.*

5.2. Geometry-Processing

In extending our advection formulation of Shock Filters to geometry-processing, we find similar performance, whether we are sharpening per-vertex colors or the geometry itself.

Table 2 gives the running times for processing the colors on the 213K vertex "Pleo" model (Figure 4) and the normals on the 2M vertex "David" model (Figure 5). As with image-

	Smoothing Passes (s)					
	0	1	2	4	8	16
Pre-Processing	1.9	1.8	1.8	1.8	1.8	1.8
Flow Computation	0.2	0.4	0.6	0.8	1.4	2.4
Advection	0.9	1.4	1.6	1.8	2.1	2.5
Geometry Fitting	1.0	1.1	1.1	1.1	1.0	1.1

Table 3: *Break-down of processing time (in seconds) for sharpening geometry, as a function of smoothing passes.*

processing, we find that the computational complexity is sub-linear in the duration of the flow, with run-times approaching a constant as flow time is increased. (Note that for the "David" results, fitting vertex positions to the sharpened normals requires setting up and solving a screened Poisson equation, which takes an additional 9.3 seconds on average.)

Finally, Table 3 gives a break-down of the running times for the different phases of sharpening the positions of the 656K vertex "Chinese Dragon" mesh (Figure 6), using a time step of $t = 2048$ and varying the number of smoothing passes. As expected, the running times for computing (*Pre-Processing*) and solving (*Geometry Fitting*) the screened Poisson system are constant independent of the number of smoothing passes, while the cost of computing the flow field (*Flow Computation*) grows linearly with the number of smoothing passes.

We also note that the time required for back-tracing along flow lines (*Advection*) grows with the number of smoothing passes. We believe that this is because successive smoothing passes remove (non-persistent) sources/sinks so that points back-trace longer paths to reach the steady-state.

Comparison to bilateral and mean-shift filters In Figure 11 we compare our approach to the geometry filtering methods of Solomon *et al.* [SCBW14] on two noisy "frog" datasets [Sol08], comparing the results of bilateral filtering (second column), mean-shift filtering (third column), and Shock Filter advection (fourth column). For low amplitude noise, all three successfully clean the data, and Shock Filter advection additionally sharpens the edges. For larger amplitude noise, bilateral filtering and Shock Filter advection fail to clean the data. However, by mollifying the normal field in a pre-processing step (by performing two passes of Laplacian smoothing on the normals while keeping the vertex positions unchanged) [JDD03], we get a signal that is successfully sharpened with Shock Filter advection (last column).

We stress that though our approach is robust to noise, it is not designed for mesh-denoising. We expect that when the input is noisy, applying state-of-the-art denoising techniques in a pre-processing step will improve the quality of the sharpened geometry.

6. Conclusion

We present a novel interpretation of Shock Filters as a PDE describing image advection. We show that this interpretation provides a simple and unconditionally stable implementation that extends to signal processing on triangle meshes. In the future, we would like to consider several extensions:

- We would like to explore other approaches for performing the advection (e.g. [AWO*14]).
- We would like to extend our work to the gradient-domain context by adapting the implementation to flow signal gradients, rather than values.
- Leveraging ideas from optical flow [LK81], we would like to explore a hierarchical extension, estimating the flow at coarser resolutions and using the low-resolution flow for advecting at higher resolutions. Using such an approach, it may be possible to "jump over" small edges at the finer resolution in order to converge at larger silhouettes.

Acknowledgements The author would like to express sincere gratitude to Jerry Prince and Justin Solomon for invaluable discussion and recommendations, and would additionally like to thank Justin Solomon and Mark Pauly for making their data available.

References

[ALM92] ALVAREZ L., LIONS P.-L., MOREL J.-M.: Image selective smoothing and edge detection by nonlinear diffusion. ii. *SIAM J. Numer. Anal. 29* (1992), 845–866. 2, 7

[AM94] ALVAREZ L., MAZORRA L.: Signal and image restoration using shock filters and anisotropic diffusion. *SIAM Journal of Numerical Analysis 31* (1994), 590–605. 2, 3

[AWO*14] AZENCOT O., WEISSMANN S., OVSJANIKOV M., WARDETZKY M., BEN-CHEN M.: Functional fluids on surfaces. *Computer Graphics Forum 33* (2014), 237–246. 8

[BCCZ08] BHAT P., CURLESS B., COHEN M., ZITNICK L.: Fourier analysis of the 2D screened Poisson equation for gradient domain problems. In *Proc. of the 10th European Conference on Computer Vision* (2008), pp. 114–128. 2

[BN08] BELKIN M., NIYOGI P.: Towards a theoretical foundation for Laplacian-based manifold methods. *Journal of Computer and System Sciences 74* (2008), 1289–1308. 10

[BPD06] BAE S., PARIS S., DURAND F.: Two-scale tone management for photographic look. *ACM Trans. on Graphics 25* (2006), 637–645. 7

[BXBT02] BAJAJ C. L., XU G. L., BAJAJ R. L., T G. X.: Anisotropic diffusion of subdivision surfaces and functions on surfaces. *ACM Trans. on Graphics 22* (2002), 4–32. 2

[CDR00] CLARENZ U., DIEWALD U., RUMPF M.: Anisotropic geometric diffusion in surface processing. *Visualization Conference, IEEE 0* (2000), 70. 2

[CLB*09] CHUANG M., LUO L., BROWN B., RUSINKIEWICZ S., KAZHDAN M.: Estimating the Laplace-Beltrami operator by restricting 3D functions. *Computer Graphics Forum* (2009), 1475–1484. 2

[CM11] CHOUDHURY A., MEDIONI G.: Perceptually motivated automatic sharpness enhancement using hierarchy of non-local means. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on* (2011). 7

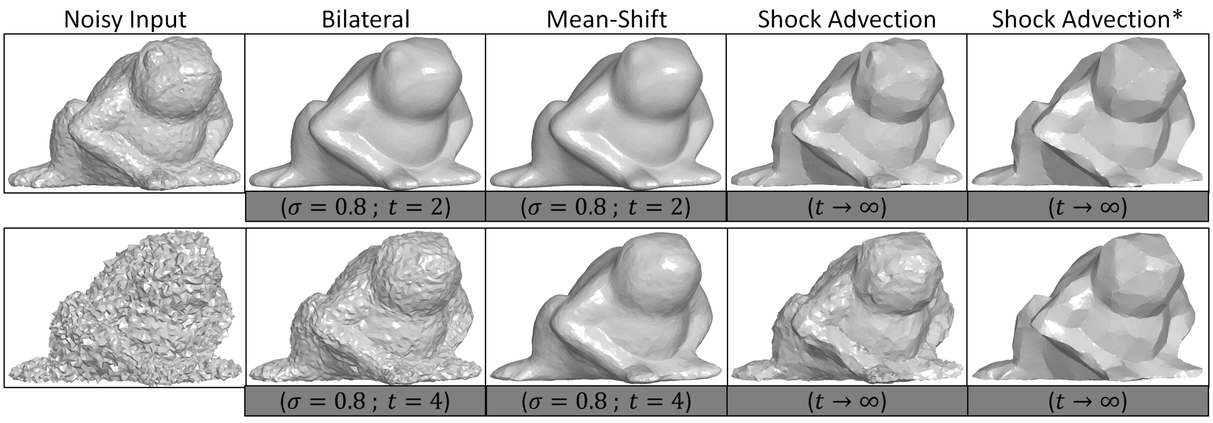

Figure 11: *Comparison to related methods: Showing the noisy input (left column), results of bilateral and mean-shift filtering [SCBW14] (second and third columns), results obtained with Shock Filter advection (fourth column), and results obtained by applying Shock Filter advection to the mollified normals (last column). Labels indicate the parameters used to generate the different geometries.*

[DM98] DAGUM L., MENON R.: OpenMP: An industry-standard API for shared-memory programming. *IEEE Computational Science and Engineering 5* (1998), 46–55. 5

[Fat07] FATTAL R.: Image upsampling via imposed edges statistics. *ACM Trans. on Graphics 26* (2007). 2

[FDCO03] FLEISHMAN S., DRORI I., COHEN-OR D.: Bilateral mesh denoising. *ACM Trans. on Graphics 22* (2003), 950–953. 2

[FF10] FREEDMAN G., FATTAL R.: Image and video upscaling from local self-examples. *ACM Trans. on Graphics* (2010). 2

[FFDLS08] FARBMAN Z., FATTAL R., D. LISCHINSKI D., SZELISKI R.: Edge-preserving decompositions for multi-scale tone and detail manipulation. *ACM Trans. on Graphics 27* (2008), 67:1–67:10. 7

[GSZ02] GILBOA G., SOCHEN N., ZEEVI Y.: Regularized shock filters and complex diffusion. In *Computer Vision - ECCV 2002*, Heyden A., Sparr G., Nielsen M., Johansen P., (Eds.), vol. 2350 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2002, pp. 399–413. 2, 4

[JDD03] JONES T., DURAND F., DESBRUN M.: Non-iterative, feature-preserving mesh smoothing. *ACM Trans. on Graphics 22* (2003), 943–949. 8

[LK81] LUCAS B., KANADE T.: An iterative image registration technique with an application to stereo vision (darpa). In *Proc. of the 1981 DARPA Image Understanding Workshop* (1981), pp. 121–130. 8

[OR90] OSHER S., RUDIN L.: Feature-oriented image enhancement using shock filters. *SIAM Journal of Numerical Analysis 27* (1990), 919–940. 1, 2, 3

[Pau12] PAULY M.: Statue Model Repository, 2012. URL: http://lgg.epfl.ch/statues_dataset.php. 5

[PM90] PERONA P., MALIK J.: Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. on Pattern Analysis and Machine Intelligence 12* (1990), 629–639. 2, 7

[SCBW14] SOLOMON J., CRANE K., BUTSCHER A., WOJTAN C.: A general framework for bilateral and mean shift filtering. *CoRR abs/1405.4734* (2014). 8, 9

[SLJT08] SHAN Q., LI Z., JIA J., TANG C.-K.: Fast image/video upsampling. *ACM Trans. on Graphics* (2008). 2

[Sol08] SOLOMON J.: Bilateral and Mean-Shift Data, 2008. URL: http://people.csail.mit.edu/jsolomon/share/bilateral_data/. 8

[SSD09] SUBR K., SOLER C., DURAND F.: Edge-preserving multiscale image decomposition based on local extrema. In *ACM SIGGRAPH Asia 2009 Papers* (2009). 7

[Sta99] STAM J.: Stable fluids. In *ACM SIGGRAPH Conference Proceedings* (1999), pp. 121–128. 2

[Sta03] STAM J.: Flows on surfaces of arbitrary topology. *ACM Trans. on Graphics 22* (2003), 724–731. 2

[SXS08] SUN J., XU Z., SHUM H.-Y.: Image super-resolution using gradient profile prior. *Computer Vision and Pattern Recognition* (2008). 2

[SY04] SHI L., YU Y.: Inviscid and incompressible fluid simulation on triangle meshes. *Computer Animation and Virtual Worlds 15* (2004), 173–181. 10

[Tau95] TAUBIN G.: A signal processing approach to fair surface design. In *ACM SIGGRAPH Conference Proceedings* (1995), pp. 351–358. 10

[TM98] TOMASI C., MANDUCHI R.: Bilateral filtering for gray and color images. In *Proc. of the International Conference on Computer Vision* (1998), pp. 839–846. 2, 7

[TT99] TUMBLIN J., TURK G.: LCIS: A boundary hierarchy for detail-preserving contrast reduction. In *ACM SIGGRAPH Conference Proceedings* (1999), pp. 83–90. 7

[TWBO02] TASDIZEN T., WHITAKER R., BURCHARD P., OSHER S.: Geometric surface smoothing via anisotropic diffusion of normals. In *VIS '02: Proc. of the conference on Visualization '02* (2002), pp. 125–132. 2

[VAKMS12] VACAVANT A., ALBOUY-KISSI A., MENGUY P., SOLOMON J.: Fast smoothed shock filtering. In *Pattern Recognition (ICPR), 2012 21st International Conference on* (2012), pp. 182–185. 2

[VL08] VALLET B., LÉVY B.: Spectral geometry processing with manifold harmonics. *Computer Graphics Forum 27* (2008), 251–160. 2

[XLXJ11] Xu L., Lu C., Xu Y., Jia J.: Image smoothing via L_0 gradient minimization. *ACM Trans. on Graphics* (2011). 2

[YZX*04] Yu Y., Zhou K., Xu D., Shi X., Bao H., Guo B., Shum H.-S.: Mesh editing with Poisson-based gradient field manipulation. *ACM Trans. on Graphics 23* (2004), 644–651. 4

Appendix A: Spatial Discretization

Implementing Shock Filter advection requires discretizing the flow, smoothing the potential, and tracing flow-lines. We describe the implementation details for each.

To consolidate the discussion, we will assume that the domain is a mesh, $M=(V,F)$, with V the vertex set and F the face set. (For images, V is the set of pixels and F is the set of squares joining four adjacent pixels.)

Discretizing the Flow

Given a signal $I:V\to\mathbb{R}$, we define the gradient as a piecewise constant function assigning a vector to each face, $\nabla I: F\to\mathbb{R}^2$. For images, the vector associated to a face is obtained by computing the finite differences along edges of the mesh and setting the x/y-coordinates to the average difference on adjacent horizontal/vertical edges. For surfaces, the vector is defined as the gradient of the linear interpolant of the values at the triangle's corners. (For processing geometry, the initial normal field is defined per-face so we compute a per-vertex signal by taking the area-weighted average.)

Given the vector field, we compute its square norm on each face to get a function $\|\nabla I\|^2: F\to\mathbb{R}$, which we turn into the vertex-valued potential, $P:V\to\mathbb{R}$ by setting the value at a vertex to the area-weighted average of the values in adjacent faces.

Finally, taking the gradient of the potential as above, we get a piecewise constant vector-field assigning a flow direction to each face, $\vec{u}=\nabla P: F\to\mathbb{R}^2$

Smoothing

To obtain a smoother flow, we apply umbrella smoothing to the signal prior to computing gradients, using Gaussian weights [Tau95, BN08]:

$$I[v]\leftarrow\frac{I[v]+\sum_{w\in N(v)}\alpha_{vw}\cdot I[w]}{1+\sum_{w\in N(v)}\alpha_{vw}}\quad\text{with}\quad\alpha_{vw}=e^{-\|(v-w)/\bar{\ell}\|^2}$$

where $N(v)$ is the set of one-ring neighbors of v and $\bar{\ell}$ is the average edge length. Smoothing is performed before both gradient calculations (i.e. before computing the potential $P=\|\nabla I\|^2$ and before computing the flow field $\vec{u}=\nabla P$).

Tracing Flow-Lines

Given a flow field $\vec{u}$, a point $p\in M$ and a time step $t\geq 0$, we trace the position of p along the flow by iteratively sampling the flow field and stepping along the flow direction, ensuring that no individual step is larger than a fixed $\delta>0$:

```
Trace( $p$ , $\vec{u}$ , $t$ )
1  $\delta \leftarrow \bar{\ell}/10$
2  $\vec{v} \leftarrow \vec{u}(p)$ , $\vec{v}_0 \leftarrow \vec{v}$
3  while( $t>0$ AND $\langle\vec{v},\vec{v}_0\rangle>0$ )
4      $dt \leftarrow \min(t, \delta/|\vec{v}|)$          compute the step-size
5      $p \leftarrow p+\vec{v}\cdot dt$                   advance the position
6      $t \leftarrow t-dt$                                 reduce the flow time
7      $\vec{v}_0 \leftarrow \vec{v}$ , $\vec{v} \leftarrow \vec{u}(p)$     update flow directions
8  return $p$
```

For images, Step 5 only requires vector addition. For surfaces, this step is implemented as in [SY04] traveling along the (straight-line) geodesic from p in direction $\vec{v}$. If the geodesic intersects a triangle edge, the edge-adjacent triangle is unfolded into the plane of the current triangle and the path is extended into the next triangle.

Face Advection For geometry processing, normals are defined per-face and we compute the advected position by starting at the center of the face and following the flow:

```
AdvectFaces( $I$ , $\vec{u}$ , $t$ )
1  for $f\in F$
2      $\bar{f} \leftarrow$ Center( $f$ )          compute the face center
3      $p \leftarrow$ Trace( $\bar{f}$ , $\vec{u}$ , $t$ )    flow from the center
3      $J[f] \leftarrow I(p)$                    sample
4  return $J$
```

Vertex Advection For signals defined per-vertex, the situation is more challenging since the vector-field is only defined in the interior of faces. To address this, we compute the value at a vertex by offsetting into each of the adjacent faces, flowing from the offset positions, and taking the average:

```
AdvectVertices( $I$ , $\vec{u}$ , $t$ )
1   $\varepsilon \leftarrow 0.001$
2   for $v\in V$
3       $J[v] \leftarrow 0$
4       for $f\ni v$
5           $\bar{f} \leftarrow$ Center( $f$ )                compute the face center
6           $p \leftarrow v\cdot(1-\varepsilon)+\bar{f}\cdot\varepsilon$   offset into the triangle
7           $q \leftarrow$ Trace( $p$ , $\vec{u}$ , $t$ )         flow from the offset
8           $J[f] \leftarrow J[f]+I(q)$                 increment
9       $J[v] \leftarrow J[v]/$Valence( $v$ )              average
10  return $J$
```