The Heat Method for Distance Computation

By Keenan Crane, Clarisse Weischedel, and Max Wardetzky

Abstract

We introduce the heat method for solving the single- or multiple-source shortest path problem on both flat and curved domains. A key insight is that distance computation can be split into two stages: first find the direction along which distance is increasing, then compute the distance itself. The heat method is robust, efficient, and simple to implement since it is based on solving a pair of standard sparse linear systems. These systems can be factored once and subsequently solved in near-linear time, substantially reducing amortized cost. Real-world performance is an order of magnitude faster than state-of-the-art methods, while maintaining a comparable level of accuracy. The method can be applied in any dimension, and on any domain that admits a gradient and inner product—including regular grids, triangle meshes, and point clouds. Numerical evidence indicates that the method converges to the exact distance in the limit of refinement; we also explore smoothed approximations of distance suitable for applications where greater regularity is desired.

1. INTRODUCTION

The multiple-source shortest path problem seeks the distance from each point of a domain to the closest point within a given subset; different versions of this problem are fundamental to a wide array of problems across computer science and computational mathematics. Solutions date back at least to Dantzig’s work on linear programs; typically the problem is formulated in terms of a weighted graph, as in Dijkstra’s algorithm. Often, however, one wishes to capture the distance on a continuous domain; a key example is illustrated in Figure 1 (left) where the graph distance will overestimate the straight-line Euclidean distance, no matter how fine the grid becomes. In 2D, an important development was the formulation of “exact” algorithms, where paths can cut through the faces of a triangulation; a great deal of subsequent work has focused on making these \( O(n^2) \) algorithms practical for large datasets. However, for problems in data analysis and scientific computing it is not clear that the cost and complexity of exact algorithms are always well-justified, since the triangulation itself is only an approximation of the true domain (see Figure 4).

A very different approach is to formulate the problem in terms of partial differential equations (PDEs), where domain approximation error can be understood via, for example, traditional finite element analysis. However, the particular choice of continuous formulation has a substantial impact on computation. The heat method was inspired by S.R.S. Varadhan’s classic result in differential geometry relating heat diffusion and geodesic distance, which measures the length along shortest and straightest curves through the domain rather than straight lines through space. Our key observation is that one can decompose distance computation into two stages: first determine the direction along which distance increases, then recover the distance itself. Moreover, since each stage amounts to a standard problem in numerical linear algebra, one can leverage existing algorithms and software to improve the efficiency and robustness of distance computation. Although this approach can in principle be used in the context of graph distance, its real utility lies in approximating the distance on continuous, curved domains. This approach has proven effective for a diverse range of applications in computational neuroscience, geometric modeling, medical imaging, computational design, and machine learning (Section 2), and has recently inspired more accurate variations of our original method.

1.1. Formulation

Imagine touching a scorching hot needle to a single point on a surface. Over time heat spreads out over the rest of the domain and can be described by a function \( k_x(y) \) called the heat kernel, which measures the heat transferred from a source \( x \) to a destination \( y \) after time \( t \). A well-known relationship between heat and distance is Varadhan’s formula, which states that the distance between points \( x \) and \( y \) at time \( t \) is given by

\[
\text{distance}(x, y) = \frac{1}{\sqrt{2\pi t}} \int_0^\infty e^{-\frac{(s-x)^2}{2t}} ds
\]

The original version of this paper was published in ACM Transactions on Graphics 32, 5 (Sept. 2013).
which says that the distance $\phi$ between any pair of points $x, y$ on a curved domain can be recovered via a simple pointwise transformation of the heat kernel:

$$\phi(x, y) = \lim_{t \to \infty} \sqrt{-4t \log k_{x}(y)}.$$  \hfill (1)

The intuition behind this behavior stems from the fact that heat diffusion can be modeled as a large collection of hot particles taking random walks starting at $x$: any particle that reaches a distant point $y$ after a small time $t$ has had little time to deviate from the shortest possible path. Previously, however, this relationship had not been exploited by numerical algorithms that compute distance.

Why had Varadhan’s formula been overlooked in this context? The main reason, perhaps, is that it requires a precise numerical reconstruction of the heat kernel, which is difficult to obtain—applying the formula to a mere approximation of $k_{x}$ does not yield the correct result, as illustrated in Figures 2 and 8. The heat method circumvents this issue by working with a broader class of inputs, namely any function whose gradient is parallel to the gradient of the true distance function. We can then separate computation into two stages: first find the gradient, then recover the distance itself.

Relative to existing algorithms, the heat method offers two major advantages. First, it can be applied to virtually any type of geometric discretization, including regular grids, polygonal meshes, and point clouds. Second, it involves only sparse linear systems, which can be prefactored once and rapidly resolved many times—this feature substantially reduces the amortized cost for applications that require repeated distance queries on a fixed geometric domain. Moreover, because the heat method is built on standard linear PDEs that are widespread in scientific computing, it can immediately take advantage of new developments in numerical linear algebra and parallelization.

2. RELATED WORK

The prevailing approach to distance computation is to solve the eikonal equation

$$|\nabla \phi| = 1,$$  \hfill (2)

subject to boundary conditions $\phi|_{\gamma} = 0$ over some subset $\gamma$ of the domain (like a point or a curve). Intuitively, this equation says something very simple: as we move away from the source, the distance function $\phi$ must change at a rate of “one meter per meter.” Computationally, however, this formulation is nonlinear and hyperbolic, making it difficult to solve directly. Typically one applies an iterative relaxation scheme such as Gauss-Seidel—special update orders are known as fast marching and fast sweeping, which are some of the most popular algorithms for distance computation on regular grids and triangulated surfaces. These algorithms can also be used on implicit surfaces, point clouds, and polygon soup, but only indirectly: distance is computed on a simplicial mesh or regular grid that approximates the original domain. Implementation of fast marching on simplicial grids is challenging due to the need for nonobtuse triangulations (which are notoriously difficult to obtain) or else an iterative unfolding procedure that preserves monotonicity of the solution; moreover these issues are not well-studied in dimensions greater than two. Fast marching and fast sweeping have asymptotic complexity of $O(n \log n)$ and $O(n)$, respectively.
but sweeping is often slower due to the large number of sweeps required to obtain accurate results.\textsuperscript{16}

One drawback of these methods is that they do not reuse information: the distance to different source sets $\gamma$ must be computed entirely from scratch each time. Also note that both sweeping and marching present challenges for parallelization: priority queues are inherently serial, and irregular meshes lack a natural sweeping order.

In a different development, Mitchell et al.\textsuperscript{27} give an $O(n^2 \log n)$ algorithm for computing the exact polyhedral distance from a single source to all other vertices of a triangulated surface. Surazhsky et al.\textsuperscript{20} demonstrate that this algorithm tends to run in sub-quadratic time in practice, and present an approximate $O(n \log n)$ version of the algorithm with guaranteed error bounds; Bonnes and Kobelt\textsuperscript{1} extend the algorithm to polygonal sources. Similar to fast marching, these algorithms propagate distance information in wavefront order using a priority queue, again making them difficult to parallelize. More importantly, the amortized cost of these algorithms (over many different source subsets $\gamma$) is substantially greater than for the heat method since they do not reuse information from one subset to the next. Finally, although\textsuperscript{30} greatly simplifies the original formulation, these algorithms remain challenging to implement and do not immediately generalize to domains other than triangle meshes.

Closest to our approach is the recent method of Rangarajan and Gurumoorthy,\textsuperscript{12} who do not appear to be aware of Varadhan’s formula—they instead derive an analogous relationship between the distance function and solutions $\psi$ to the time-independent Schrödinger equation; this derivation applies only in flat Euclidean space rather than general curved domains. Moreover, they compute solutions using the fast Fourier transform, which limits computation to regular grids.

A slight modification of the heat method allows us to compute a smoothed distance function, useful in contexts where sharp discontinuities can cause subsequent numerical difficulties. Previous smooth distance approximations provide this regularity at the cost of poor approximation of the true geometric length\textsuperscript{10, 14, 21, 33}, see Section 3.3 for a comparison.

Recently, the heat method has facilitated a variety of tasks in computational science and data analysis. For example, Huth et al.\textsuperscript{22} use fast distance queries to optimize a probabilistic model of cortical organization; van Pelt et al.\textsuperscript{41} use the heat method to assist cerebral aneurysm assessment; Zhou et al.\textsuperscript{27} use the heat method for efficient tool path planning; Solomon et al.\textsuperscript{38} leverage our approach to efficiently solve optimal transport problems on geometric domains; Lin et al.\textsuperscript{28} apply this approach to vector-valued data in the context of manifold learning. Figure 5 shows a real-world design application of the heat method based on differential growth. Various improvements have also been made to the original algorithm; for instance, de Goes et al.\textsuperscript{13} and Yang and Cohen\textsuperscript{14} describe two different ways to extend the method to accurate computation of anisotropic distance; it has also been adapted to voxelizations\textsuperscript{6}, $C^1$ finite elements,\textsuperscript{29} and subdivision surfaces.\textsuperscript{12} Finally, Belaev and Fayolle\textsuperscript{35} provide a variational interpretation of our method, observing that more accurate results can be obtained by either iterating the heat method, or by applying more sophisticated descent strategies.

### 3. THE HEAT METHOD

A useful feature of the heat method is that the basic algorithm can be described in the purely continuous setting (i.e., in terms of curved surfaces, or more generally, smooth manifolds) rather than in terms of discrete data structures and algorithms. In other words, at this point one should not imagine that we have chosen a particular data structure (triangle meshes, grids, point clouds, etc.) or even dimension (2D, 3D, etc.). Instead, we focus on a general principle that can be applied on many different domains in different dimensions. We will later explore several particular choices of spatial and temporal discretization (Sections 3.1 and 3.2); further alternatives have been explored in recent literature.\textsuperscript{13, 29, 45}

In general, the heat method can be applied in any setting where one has a gradient operator $\nabla$, divergence operator $\nabla$, and Laplace operator $\Delta = \nabla \cdot \nabla$—standard derivatives from vector calculus, possibly generalized to curved domains. Expressed in terms of these operators, the heat method consists of three basic steps:

#### Algorithm 1 The Heat Method

I. Integrate the heat flow $\dot{u} = \Delta u$ for some fixed time $t$.

II. Evaluate the vector field $X = -\nabla u_t / |\nabla u_t|$.

III. Solve the Poisson equation $\Delta \phi = \nabla \cdot X$.

The function $\phi$ approximates the distance to a given source set, approaching the true distance as $t$ goes to zero (Equation 1). For instance, to recover the distance to a single point $x$ we use initial conditions $u_0 = \delta(x)$, that is, a Dirac delta encoding an “infinite spike” of heat. More generally we can obtain the distance to any subset $\gamma$ by letting $u_{\gamma}$ be a generalized Dirac distribution\textsuperscript{42}—essentially an indicator function over $\gamma$; see Figures 3 and 7. Note that since the solution to (III) is determined only up to an additive constant, final values are shifted such that the smallest distance is zero.

The heat method can be motivated as follows. Consider an approximation $u_t$ of heat flow for a fixed time $t$. Unless $u_t$ exhibits precisely the right rate of decay, Varadhan’s
by considering the elliptic boundary value problem
\[ \Delta u (x) = 0 \text{ on } M \setminus \gamma, \]
\[ v_i = 1 \text{ on } \gamma. \]  
which for a point source yields a solution \( v_i \) equal to \( u_i \) up to a multiplicative constant. As established by Varadhan in his proof of Equation (1), \( v_i \) also has a close relationship with distance, namely
\[ \lim_{t \to 0} \frac{1}{2t} \log v_i = \phi. \]
This relationship ensures the validity of steps II and III since the transformation applied to \( v_i \) preserves the direction of the gradient.

3.2. Spatial discretization
Here we detail several possible implementations of the heat method on triangle meshes, polygon meshes, and point clouds. Note that the heat method can also be used on flat Euclidean domains of any dimension by simply applying standard finite differences on a regular grid; Belyaev and Fayolle outline an implementation on tetrahedral (3D) meshes.

**Triangle meshes.** Let \( u \in \mathbb{R}^{|V|} \) specify a piecewise linear function on a triangulated surface with vertices \( V \), edges \( E \), and faces \( F \). A standard discretization of the Laplacian at a vertex \( i \) is given by
\[ (L u)_i = \frac{1}{2 A_i} \sum_j (\cot \alpha_{ij} + \cot \beta_{ij})(u_j - u_i), \]
where \( A_i \) is one third the area of all triangles incident on vertex \( i \), the sum is taken over all neighboring vertices \( j \), and \( \alpha \) and \( \beta \) are the angles opposing the corresponding edge. We can express this operation via a matrix \( L = M^{-1} L_c \), where \( M \in \mathbb{R}^{|V| \times |V|} \) is a diagonal matrix containing the vertex areas and \( L_c \in \mathbb{R}^{|V| \times |V|} \) is the cotan operator representing the
remaining sum. Heat flow can then be computed by solving the symmetric positive-definite system

$$(M - tL_C)u = \delta_i,$$

where $\delta_i$ is a Kronecker delta (or indicator function) over $\gamma$. The gradient in a given triangle can be expressed succinctly as

$$\nabla u = \frac{1}{2A_t} \sum_i u_i (N \times e_i),$$

where $A_t$ is the area of the triangle, $N$ is its outward unit normal, $e_i$ is the $i$th edge vector (oriented counter-clockwise), and $u_i$ is the value of $u$ at the opposing vertex. The integrated divergence associated with vertex $i$ can be written as

$$\nabla \cdot X = \frac{1}{2} \sum_j \cot \theta_i (e_i \cdot X_j) + \cot \theta_j (e_j \cdot X_i),$$

where the sum is taken over incident triangles $j$ each with a vector $X_j$, $e_j$, and $e_i$ are the two edge vectors of triangle $i$ containing $i$, and $\theta_i$, $\theta_j$ are the opposing angles. If we let $b \in \mathbb{R}^n$ be the vector of (integrated) divergences of the normalized vector field $X$, then the final distance function is computed by solving the symmetric Poisson problem

$$L_c \phi = b.$$ 

As noted in Section 3.1, the solution to step I is a function that decays exponentially with distance. Fortunately, normalization of small values is not a problem because floating point division involves only arithmetic on integer exponents; likewise, the large range of magnitudes does not adversely affect accuracy because gradient calculation is local. For the calculation of $\phi$ itself we advocate the use of a direct (Cholesky) solver in double precision; empirically we observe roughly uniform pointwise relative error across the domain.

**Polygon meshes.** Curved surfaces are often described by polygons that are neither planar nor convex; although such polygons can of course be triangulated, doing so can adversely affect an existing computational pipeline. We instead leverage the polygonal Laplacian of Alexa and Wardetzky\(^4\) to implement the heat method directly on polygonal meshes—the only challenge in this setting is that for nonplanar polygons the gradient vector no longer has a clear geometric meaning. This issue is resolved by noting that we need only the magnitude $|\nabla u|$ of the gradient; see Crane et al.,\(^1\) Appendix A. Therefore, on a fixed mesh decreasing the value of $t$ does not necessarily improve accuracy, even in exact arithmetic—to improve accuracy we must simultaneously refine the mesh and decrease $t$ accordingly. Moreover, very large values of $t$ produce an over-smoothed approximation of geodesic distance (Section 3.3). For a fixed mesh, we therefore seek an optimal time step $t^*$ that is neither too large nor too small.

An optimal value of $t^*$ is difficult to obtain due to the complexity of analysis involving the cut locus.\(^28\) We instead use a simple estimate that works well in practice, namely $t = mh^2$ where $h$ is the mean spacing between adjacent nodes and $m > 0$ is a constant. This estimate is motivated by the fact that $h^2 \Delta$ is invariant with respect to scale and refinement; numerical experiments suggest that $m = 1$ yields near-optimal accuracy for a wide variety of problems. In this paper the time step

$$t = h^2$$

is therefore used uniformly throughout all tests and examples, except where we explicitly seek a smoothed approximation of distance, as in Section 3.3. For highly nonuniform meshes one could set $h$ to the maximum spacing, providing a more conservative estimate. Numerical underflow could theoretically occur for extremely small $t$, though we do not encounter this issue in practice.

**Numerics.** As demonstrated in Figures 10, 18, and 19, one does not need a particularly nice mesh or point cloud to get a reasonable distance function. However, as with any numerical
method, accuracy and other properties of the solution may be influenced by the quality of the mesh. For instance, in some applications one may wish to avoid “spurious minima,” that is, local maxima or minima that do not appear in the true (smooth) distance function. At present, there is no numerical scheme that guarantees the absence of spurious minima on arbitrary meshes, including exact polyhedral schemes.\(^{17}\) Empirically, however, we observe that the heat method produces fewer spurious minima than either fast marching or the biharmonic distance (see Figure 20), in part due to regularization from the Hodge step (step III). In cases where one wishes to avoid spurious minima altogether, we advocate the use of Delaunay meshes.

### 3.3. Smoothed distance

Geodesic distance fails to be smooth at points in the cut locus, that is, points at which there is no unique shortest path to the source—these points appear as sharp cusps in the level lines of the distance function. Non-smoothness can result in numerical difficulty for applications which need to take derivatives of the distance function \(\phi\) (e.g., level set methods), or may simply be undesirable aesthetically.

Several distances have been designed with smoothness in mind, including diffusion distance,\(^{10}\) commute-time distance,\(^{14}\) and biharmonic distance\(^{21}\) (see the last reference for a more detailed discussion). These distances satisfy a number of important properties (smoothness, isometry-invariance, etc.), but are poor approximations of true geodesic distance, as indicated by uneven spacing of isolines (see Figure 12, middle). They can also be expensive to evaluate, requiring one to either solve a linear system for each vertex, or compute a large number of eigenvectors of the Laplace matrix (~150 to 200 in practice).

In contrast, one can rapidly construct smoothed approximations of geodesic distance by simply applying the heat method for large values of \(t\) (Figure 11). The computational cost remains the same, and isolines are evenly spaced for any value of \(t\) due to normalization (step II); the solution is isometry invariant since it depends only on intrinsic operators. For a time step \(t = mh^2\), meaningful values of \(m\) are found in the range \(1 - 10^6\)—past this point the term \(t\Delta\) dominates, resulting in little visible change.

### 3.4. Boundary conditions

To solve the equations in steps I and II, we must define the behavior of derivatives near the boundary. Intuitively, the behavior of our distance approximation should not be significantly influenced by the shape of the boundary (Figure 13)—for instance, cutting off a corner of a convex domain should not affect the distance at the points that remain. For exact distance computation, we can apply standard zero-Neumann or zero-Dirichlet boundary conditions, since this choice does not affect the behavior of the smooth limit.
solution (see Renesse\textsuperscript{44} Corollary 2 and Norri,\textsuperscript{30} Theorem 1.1, respectively). Boundary conditions do however alter the behavior of the smoothed distance. Although there is no well-defined “correct” behavior for this smoothed function, we advocate the use of boundary conditions obtained by taking the mean of the Neumann solution $u_N$ and the Dirichlet solution $u_D$, that is, $u = \frac{1}{2}(u_N + u_D)$. The intuition behind this behavior again stems from a random walker interpretation: zero Dirichlet conditions absorb heat, causing walkers to “fall off” the edge of the domain. Neumann conditions prevent heat from flowing out of the domain, effectively “reflecting” random walkers. Averaged conditions mimic the behavior of a domain without boundary: the number of walkers leaving equals the number of walkers returning. Figure 14 shows how boundary conditions affect the behavior of geodesics in a path-planning scenario.

Figure 13. Effect of Neumann (top-left), Dirichlet (top-right) and averaged (bottom-left) boundary conditions on smoothed distance. Averaged boundary conditions mimic the behavior of the same surface without boundary.

Figure 14. For path planning, the behavior of geodesics can be controlled via boundary conditions and the time step $t$. Top-left: Neumann conditions encourage boundary adhesion. Top-right: Dirichlet conditions encourage avoidance. Bottom-left: small values of $t$ yield standard straight-line geodesics. Bottom-right: large values of $t$ yield more natural trajectories.

Figure 15. Meshes used in Table 1. Left to right: Bunny, Isis, Horse, Bimba, Aphrodite, Lion, Ramses.

4. EVALUATION

4.1. Performance

A key advantage of the heat method is that the linear systems in steps I and III can be prefactored. Our implementation uses sparse Cholesky factorization,\textsuperscript{9} which for Poisson-type problems has guaranteed sub-quadratic complexity but in practice scales much better\textsuperscript{5}; moreover there is strong evidence to suggest that sparse systems arising from elliptic PDEs can be solved in very close to linear time.\textsuperscript{34,39}

Independent of these issues, the amortized cost for problems with a large number of right-hand sides is roughly linear, since back substitution can be applied in essentially linear time. See inset for a breakdown of relative costs in our
implementation; Potential is the time taken to compute the
right hand side in step III.

In practice, a number of factors affect the run time of
the heat method including the choice of spatial discreti-
zation, discrete Laplacian, and geometric data structures.
As a typical example, we compared the scheme from
Triangle meshes section to the first-order fast march-
ing method of Kimmel and Sethian\cite{19} and the exact algo-
rithm of Mitchell et al.,\cite{27} using the state-of-the-art fast
marching implementation of Peyré and Cohen\cite{31} and the
exact implementation of Kirsanov.\cite{40} The heat method
was implemented in ANSI C in double precision using a
vertex-face adjacency list. Single-threaded performance
was measured on a 2.4 GHz Intel Core 2 Duo (Table 1).
Note that even for a single distance computation the heat
method outperforms fast marching; more importantly,
updating distance for new subsets $\gamma$ is consistently an
order of magnitude faster (or more) than both fast march-
ing and the exact algorithm.

4.2. Accuracy
We examined errors in the heat method, fast marching,\cite{19}
and the polyhedral distance,\cite{27} relative to mean edge length
$h$ on triangulated surfaces. Both fast marching and the
heat method appear to exhibit linear convergence; it is
interesting to note that even the exact polyhedral distance
provides only quadratic convergence. Keeping this fact in
mind, Table 1 uses the polyhedral distance as a baseline
for comparison on more complicated geometries—Max is
the maximum error as a percentage of mesh diameter and
Mean is the mean relative error at each vertex. Note that
fast marching tends to achieve a smaller maximum error,
whereas the heat method does better on average. Figure 16
gives a visual comparison of accuracy; the only notable dis-
crepancy is a slight smoothing at sharp cusps, which may
explain the larger maximum error. Figure 17 indicates that
smoothing does not interfere with the extraction of the
cut locus—here we visualize values of $|\Delta \phi|$ above a fixed
threshold. Overall, the heat method exhibits errors of the
same order and magnitude as fast marching (at lower com-
putational cost) and is therefore suitable in applications
where fast marching is presently used; see Crane et al.\cite{14}
for more extensive comparisons.

<table>
<thead>
<tr>
<th>Model</th>
<th>Triangles</th>
<th>Heat method</th>
<th>Fast marching</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Precompute (s)</td>
<td>Solve (s)</td>
</tr>
<tr>
<td>Bunny</td>
<td>28k</td>
<td>0.21</td>
<td>0.01s (28x)</td>
</tr>
<tr>
<td>Isis</td>
<td>93k</td>
<td>0.73</td>
<td>0.05s (21x)</td>
</tr>
<tr>
<td>Horse</td>
<td>96k</td>
<td>0.74</td>
<td>0.05s (20x)</td>
</tr>
<tr>
<td>Kitten</td>
<td>106k</td>
<td>1.13</td>
<td>0.06s (22x)</td>
</tr>
<tr>
<td>Bimba</td>
<td>149k</td>
<td>1.79</td>
<td>0.09s (29x)</td>
</tr>
<tr>
<td>Aphrodite</td>
<td>205k</td>
<td>2.66</td>
<td>0.12s (47x)</td>
</tr>
<tr>
<td>Lion</td>
<td>353k</td>
<td>5.25</td>
<td>0.24s (24x)</td>
</tr>
<tr>
<td>Ramses</td>
<td>1.6M</td>
<td>63.4</td>
<td>1.45s (68x)</td>
</tr>
</tbody>
</table>

Best speed/accuracy in bold; speedup in orange.

Figure 16. Visual comparison of accuracy. Left: exact polyhedral
distance. Using default parameters, the heat method (middle) and
fast marching (right) both produce results of comparable accuracy,
here within less than 1% of the polyhedral distance—see Table 1 for a
more detailed comparison.

Figure 17. Medial axis of the hiragana letter “a” extracted by
thresholding second derivatives of the distance to the boundary. Left:
fast marching. Right: heat method.

More recent implementations of the heat method improve
accuracy by using a different spatial discretization,\cite{29}
or by iteratively updating the solution.\cite{3} The accuracy of fast
marching schemes is determined by the choice of update
since it tends to produce results more accurate than fast marching at a similar computational cost. However, accuracy is measured relative to the polyhedral distance rather than the smooth geodesic distance of the approximated surface. Like fast marching, Surazhsky’s method does not take advantage of precomputation and therefore exhibits a significantly higher amortized cost than the heat method; it is also limited to triangle meshes.

4.3. Robustness

Two factors contribute to the robustness of the heat method, namely (1) the use of an unconditionally stable time discretization and (2) an elliptic rather than hyperbolic formulation (i.e., relatively stable local averaging vs. more sensitive global wavefront propagation). Figure 19 verifies that the heat method continues to work well even on meshes that are poorly discretized or corrupted by a large amount of noise (here modeled as uniform Gaussian noise applied to the vertex coordinates). In this case we use a moderately large value of $t$ to investigate the behavior of our smoothed distance; similar behavior is observed for small $t$ values. Figure 18 illustrates the robustness of the method on a surface with many small holes as well as long sliver triangles.

5. CONCLUSION

The heat method is a simple, general method that can be easily incorporated into a broad class of algorithms. However, a number of highly accurate rules have been developed for regular grids (e.g., HJ WENO\textsuperscript{18}), but fewer options are available on irregular domains such as triangle meshes, the predominant choice being the first-order update of Kimmel and Sethian.\textsuperscript{19} Finally, the approximate algorithm of Surazhsky et al.\textsuperscript{40} provides an interesting comparison...
great deal remains to be explored, including further investigation of alternative spatial discretizations, and formal analysis of convergence under refinement. Further exploration of the parameter $t$ also provides an avenue for future work (especially in the case of variable spacing), though one should note that the existing estimate already outperforms fast marching in terms of mean error (Table 1). Another natural question is whether a similar transformation can be applied to a larger class of Hamilton-Jacobi equations; it is likewise enticing to apply a similar principle to distance computation on domains that do not immediately resemble a continuous domain (such as a weighted graph).

Acknowledgments
This work was funded by a Google PhD Fellowship and a grant from the Fraunhofer Gesellschaft. Thanks to Michael Herrmann for inspiring discussions. Meshes are provided courtesy of the Stanford Computer Graphics Laboratory, the AIM@Shape Repository, Luxology LLC, and Jotero Gbr (http://www.evolution-of-genius.de/).

References

Keenan Crane (kmcrane@cs.cmu.edu), Carnegie Mellon University.
Clariisse Weischedel and Max Wardetzky (Clariisse.Weischedel@darmstadt.ihk.de, wardetzky@math.uni-goettingen.de), University of Göttingen.