Geodesics in Heat: A New Approach to Computing Distance Based on Heat Flow

KEENAN CRANE
Caltech
CLARISSE WEISCHEDEL, MAX WARDETZKY
University of Göttingen

We introduce the heat method for computing the geodesic distance to a specified subset (e.g., point or curve) of a given domain. The heat method is robust, efficient, and simple to implement since it is based on solving a pair of standard linear elliptic problems. The resulting systems can be prefactored once and subsequently solved in near-linear time. In practice, distance is updated in an order of magnitude faster than with state-of-the-art methods, while maintaining a comparable level of accuracy. The method requires only standard differential operators and can hence be applied on a wide variety of domains (grids, triangle meshes, point clouds, etc.). We provide numerical evidence 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 required.

Categories and Subject Descriptors: 1.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems

General Terms: Algorithms

Additional Key Words and Phrases: digital geometry processing, discrete differential geometry, geodesic distance, distance transform, heat method

ACM Reference Format:

1. INTRODUCTION

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 \( k_{t,x}(y) \) 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 [1967], which says that the geodesic distance \( \phi \) between any pair of points \( x, y \) on a Riemannian manifold can be recovered via a simple pointwise transformation of the heat kernel:

\[
\phi(x, y) = \lim_{t \to 0} \sqrt{-4t \log k_{t,x}(y)}. \tag{1}
\]

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. To date, however, this relationship has not been exploited by numerical algorithms that compute geodesic distance.

Why has 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 \( k_{t,x} \) does not yield the correct result, as illustrated in Figures 2 and 6. The heat method circumvents this issue by working with a broader class of inputs, namely any function whose gradient is parallel to geodesics. We can then separate computation into two stages: first find the gradient of the distance field, 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 even unstructured point clouds. Second, it involves only sparse linear systems, which can be prefactored once and rapidly re-solved many times. This feature makes the heat method particularly valuable for applications such as shape matching, path planning, and level set-based simulation (e.g., free-surface fluid flows), which require repeated distance queries on a fixed geometric domain. Moreover, because linear elliptic equations are widespread in scientific computing, the heat method 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

$$\left| \nabla \phi \right| = 1$$

subject to boundary conditions $\phi|_{\gamma} = 0$ over some subset $\gamma$ of the domain. 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 [Sethian 1996] and triangulated surfaces [Kimmel and Sethian 1998]. These algorithms can also be used on implicit surfaces [Memoli and Sapiro 2001], point clouds [Memoli and Sapiro 2005], and polygon soup [Campen and Kobbelt 2011], 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 a complex unfolding procedure to preserve 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 [Hysing and Turek 2005].

The main drawback of these methods is that they do not reuse information: the distance to different subsets $\gamma$ 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. Weber et al. [2008] address this issue by decomposing surfaces into regular grids, but this decomposition resamples the surface and requires a low-distortion parameterization over a small number of quadrilateral patches, which is difficult to obtain.

In a different development, Mitchell et al. [1987] 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. [2005] 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; Bommes and Kobbelt [2007] 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 [Surazhsky et al. 2005] 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 [2011], who do not appear to be aware of Varadhan’s formula – they instead derive an analogous relationship $\phi = -\sqrt{R \log \psi}$ between the distance function and solutions $\psi$ to the time-independent Schrödinger equation. We emphasize, however, that this derivation applies only in $\mathbb{R}^n$ where $\psi$ takes a special form – in this case it may be just as easy to analytically invert the Euclidean heat kernel $u_{t,x} = (4\pi t)^{-n/2}e^{-\phi(x,y)^2/4t}$. Moreover, they compute solutions using the fast Fourier transform, which limits computation to regular grids. To obtain accurate results their method requires either the use of arbitrary-precision arithmetic or a combination of multiple solutions for various values of $h$; no general guidance is provided for determining appropriate values of $h$.

Finally, there is a large literature on smoothed distances [Coifman and Lafon 2006; Fouss et al. 2007; Rustamov et al. 2009; Lipman et al. 2010], which are valuable in contexts where differentiability is required. However, existing smooth distances may not be appropriate in contexts where the geometry of the original domain is important, since they do not attempt to approximate the original metric and therefore substantially violate the unit-speed nature of geodesics (Figure 10). These distances also have an interpretation in terms of simple discretizations of heat flow – see Section 3.3 for further discussion.
The heat equation is used to model the diffusion of heat in a material. In the context of computing geodesic distances, the heat equation is applied to a manifold to diffuse a temperature field, which eventually corresponds to the geodesic distances.

**Algorithm 1** The Heat Method

1. Integrate the heat flow $\dot{u} = \Delta u$ for some fixed time $t$.
2. Evaluate the vector field $X = -\nabla u / |\nabla u|$.
3. Solve the Poisson equation $\Delta \phi = \nabla \cdot X$.

The function $\phi$ approximates geodesic distance, approaching the true distance as $t$ goes to zero (Eq. (1)). Note that the solution to step III is unique only up to an additive constant — final values simply need to be shifted such that the smallest distance is zero. Initial conditions $u_0 = \delta(x)$ (i.e., a Dirac delta) recover the distance to a single source point $x \in M$ as in Figure 1, but in general we can compute the distance to any piecewise submanifold $\gamma$ by setting $u_0$ to a generalized Dirac [Villa 2006] over $\gamma$ (see Figures 3 and 4).

The heat method can be motivated as follows. Consider an approximation $v_t$ of heat flow for a fixed time $t$. Unless $v_t$ exhibits precisely the right rate of decay, Varadhan’s transformation $u_t \mapsto e^{-t}\log u_t$ will yield a poor approximation of the true geodesic distance $\phi$ because it is highly sensitive to errors in magnitude (see Figures 2 and 6). The heat method asks for something different: it asks only that the gradient $\nabla u_t$ points in the right direction, i.e., parallel to $\nabla \phi$. Magnitude can safely be ignored since we know (from the eikonal equation) that the gradient of the true distance function has unit length. We therefore compute the normalized gradient field $X = -\nabla u / |\nabla u|$ and find the closest scalar potential $\phi$ by minimizing $\int_M |\nabla \phi - X|^2$, or equivalently, by solving the corresponding Euler-Lagrange equations $\Delta \phi = \nabla \cdot X$ [Schwarz 1995]. The overall procedure is depicted in Figure 5.

**3.1 Time Discretization**

We discretize the heat equation from step I of Algorithm 1 in time using a single backward Euler step for some fixed time $t$. In practice, this means we simply solve the linear equation

$$\left(\text{id} - t\Delta\right) u_t = u_0 \quad (3)$$

over the entire domain $M$, where $\text{id}$ is the identity (here we still consider a smooth manifold; spatial discretization is discussed in Sections 3.1 and 3.2, respectively. Let $\Delta$ be the negative-definite Laplace–Beltrami operator acting on (weakly) differentiable real-valued functions over a Riemannian manifold $(M, g)$. The heat method consists of three basic steps:

**Algorithm 1 The Heat Method**

1. Integrate the heat flow $\dot{u} = \Delta u$ for some fixed time $t$.
2. Evaluate the vector field $X = -\nabla u / |\nabla u|$.
3. Solve the Poisson equation $\Delta \phi = \nabla \cdot X$.

The function $\phi$ approximates geodesic distance, approaching the true distance as $t$ goes to zero (Eq. (1)). Note that the solution to step III is unique only up to an additive constant — final values simply need to be shifted such that the smallest distance is zero. Initial conditions $u_0 = \delta(x)$ (i.e., a Dirac delta) recover the distance to a single source point $x \in M$ as in Figure 1, but in general we can compute the distance to any piecewise submanifold $\gamma$ by setting $u_0$ to a generalized Dirac [Villa 2006] over $\gamma$ (see Figures 3 and 4).

The heat method can be motivated as follows. Consider an approximation $v_t$ of heat flow for a fixed time $t$. Unless $v_t$ exhibits precisely the right rate of decay, Varadhan’s transformation $u_t \mapsto e^{-t}\log u_t$ will yield a poor approximation of the true geodesic distance $\phi$ because it is highly sensitive to errors in magnitude (see Figures 2 and 6). The heat method asks for something different: it asks only that the gradient $\nabla u_t$ points in the right direction, i.e., parallel to $\nabla \phi$. Magnitude can safely be ignored since we know (from the eikonal equation) that the gradient of the true distance function has unit length. We therefore compute the normalized gradient field $X = -\nabla u / |\nabla u|$ and find the closest scalar potential $\phi$ by minimizing $\int_M |\nabla \phi - X|^2$, or equivalently, by solving the corresponding Euler-Lagrange equations $\Delta \phi = \nabla \cdot X$ [Schwarz 1995]. The overall procedure is depicted in Figure 5.

**3.2 Spatial Discretization**

In principle the heat method can be applied to any domain with a discrete gradient ($\nabla$), divergence ($\nabla \cdot$) and Laplace operator ($\Delta$). Note that these operators are highly local and hence do not exhibit significant cancellation error despite large global variation in $u_t$.

**3.2.1 Simplicial Meshes.** Let $u \in \mathbb{R}^{|V|}$ specify a piecewise linear function on a triangulated surface. A standard discretization of the Laplacian at a vertex $i$ is given by

$$(Lu)_i = \frac{1}{2A_i} \sum_j \left( \cot \alpha_{ij} + \cot \beta_{ij} \right) (u_j - u_i),$$

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 $\alpha_{ij}, \beta_{ij}$ are the angles opposing the corresponding edge [Mac-Neal 1949]. We can express this operation via a matrix $L = A^{-1}L_C$, where $A \in \mathbb{R}^{|V|\times|V|}$ is a diagonal matrix containing the vertex areas and $L_C \in \mathbb{R}^{|V|\times|V|}$ is a sparse matrix containing the edge weights.
the cotan operator representing the remaining sum. Heat flow can then be computed by solving the symmetric positive-definite system

\[(A - tL_C)u = \delta,\]

where \(\delta_i\) is a Kronecker delta over \(\gamma\) (the mass matrix \(A\) need not appear on the right-hand side – a Kronecker delta already gives the integrated value of a Dirac delta). The gradient in a given triangle can be expressed succinctly as

\[\nabla u = \frac{1}{2A_f} \sum_i u_i (N \times e_i)\]

where \(A_f\) is the area of the face, \(N\) is its unit normal, \(e_i\) is the \(i\)th edge vector (oriented counterclockwise), and \(u_i\) is the value of \(u\) at the opposing vertex. The integrated divergence associated with vertex \(i\) can be written as

\[\nabla \cdot X = \frac{1}{2} \sum_j \cot \theta_1(e_i \cdot X_j) + \cot \theta_2(e_2 \cdot X_j)\]

where the sum is taken over incident triangles \(j\) each with a vector \(X_j\), \(e_1\) and \(e_2\) are the two edge vectors of triangle \(j\) containing \(i\), and \(\theta_1, \theta_2\) are the opposing angles. If we let \(b \in R^{|V|}\) 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

\[L_C \phi = b.\]

Conveniently, this discretization easily generalizes to higher dimensions (e.g., tetrahedral meshes) using well-established discrete operators; see for instance [Desbrun et al. 2008].

3.2.2 Polygonal Surfaces. For a mesh with (not necessarily planar) polygonal faces, we use the polygonal Laplacian defined by Alexa and Wardetzky [2011]. The only difference in this setting is that the gradient of the heat kernel is expressed as a discrete 1-form associated with half edges, hence we cannot directly evaluate the magnitude of the gradient \(|\nabla u|\) needed for the normalization step (Algorithm 1, step II). To resolve this issue we assume that \(\nabla u\) is constant over each face, implying that

\[u_f^T L_f u_f = \int_M |\nabla u|^2 dA = |\nabla u|^2 A_f,\]

where \(u_f\) is the vector of heat values in face \(f\), \(A_f\) is the magnitude of the area vector, and \(L_f\) is the local (weak) Laplacian. We can therefore approximate the magnitude of the gradient as

\[|\nabla u|_f = \sqrt{\frac{u_f^T L_f u_f}{A_f}}.\]

This quantity is used to normalize the 1-form values associated with half edges in the corresponding face. The integrated divergence is then given by \(d^T M \alpha\) where \(\alpha\) is the normalized gradient, \(d\) is the coboundary operator and \(M\) is the mass matrix for 1-forms (see [Alexa and Wardetzky 2011] for details). These operators are applied in steps I-III as usual. Figure 7 demonstrates distance computed on an irregular polygonal mesh.

3.2.3 Point Clouds. For a discrete point sample \(P \subset R^n\) of \(M\) with no connectivity information, we solve the heat equation (step I) using the point cloud Laplacian recently introduced by Liu et al. [2012], which extends previous work of Belkin et al. [2009a]. In this formulation, the Laplacian is represented by \(A_C^{-1} L_{PC}\), where \(A_C\) is a diagonal matrix of Voronoi areas and \(L_{PC}\) is symmetric positive semidefinite (see [Liu et al. 2012], Section 3.4, for details).

To compute the vector field \(X = -\nabla u / |\nabla u|\) (step II), we represent the function \(u : P \rightarrow R\) as a height function over approximate tangent planes \(T_p\) at each point \(p \in P\) and evaluate the gradient of a weighted least squares (WLS) approximation of \(u\) [Nealen 2004]. This approximation provides a discrete gradient operator \(D \in R^{(P)\times(P)}\). To compute tangent planes, we use a moving least squares (MLS) approximation for simplicity, although other choices are possible (see Liu et al.). To find the best-fit scalar potential \(\phi\) (step III), we solve the linear, positive-semidefinite Poisson equation \(L_{PC} \phi = D^T A_C X\). Figure 8 shows two examples.

Other discretizations are certainly possible (see for instance [Luo et al. 2009]); we picked one that was simple to implement in any dimension. Note that the computational cost of the heat method depends primarily on the intrinsic dimension \(n\) of \(M\), whereas methods based on fast marching require a grid of the same dimension \(m\) as the ambient space [Memoli and Sapiro 2001] – this distinction is especially important in contexts like machine learning where \(m\) may be significantly larger than \(n\).
3.2.4 Choice of Time Step. Accuracy of the heat method relies in part on the time step \( t \). In the smooth setting, Eq. (5) suggests that smaller values of \( t \) yield better approximations of geodesic distance. In the discrete setting we instead discover that the limit solution to Eq. (3) is purely a function of the combinatorial distance, independent of how we discretize the Laplacian (see Appendix A). Therefore, on a fixed mesh decreasing the value of \( t \) does not necessarily improve accuracy, even in exact arithmetic. (Of course, we can always improve accuracy by refining the mesh and decreasing \( t \) accordingly.) Moreover, large values of \( t \) produce a smoothed approximation of geodesic distance (Section 3.3). We therefore seek an optimal time step \( t^* \) that is neither too large nor too small.

Determining a provably optimal expression for \( t^* \) is difficult due to the complexity of analysis involving the cut locus [Neel and Stroock 2004]. We instead use a simple estimate that works remarkably 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; experiments on a regular grid (Figure 18) suggest that \( m = 1 \) is the smallest parameter value that recovers the \( \ell_2 \) distance, and indeed this value yields near-optimal accuracy for a wide variety of irregularly triangulated surfaces, as demonstrated in Figure 20. In this paper the time step

\[
\Delta t = h^2
\]

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.

3.3 Smoothed Distance

Geodesic distance fails to be smooth at points in the cut locus, i.e., 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 [Coifman and Lafon 2006], commute-time distance [Fouss et al. 2007], and biharmonic distance [Lipman et al. 2010] (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 10, middle). They can also be expensive to evaluate, requiring either a large number of Laplacian eigenvectors (~150 – 200 in practice) or the solution to a linear system at each vertex.

In contrast, one can rapidly construct smoothed approximations of geodesic distance by simply applying the heat method for large values of \( t \) (Figure 9). The computational cost remains the same, and isolines are evenly spaced for any value of \( t \) due to normalization (step II); the solution is isometrically invariant since it depends only on intrinsic differential operators. For a time step \( t = mh^2 \), meaningful values of \( m \) are found in the range \( 1 - 10^5 \); past this point the term \( t \Delta \) dominates, resulting in little visible change.

Existing smooth distance functions can also be understood in terms of time-discrete heat flow. In particular, the commute-time distance \( d_C \) and biharmonic distance \( d_B \) can be expressed in terms of the harmonic and biharmonic Green’s functions \( g_C \) and \( g_B \):

\[
\begin{align*}
d_C(x, y)^2 &= g_C(x, y) - 2g_C(x, y) + g_C(y, y), \\
d_B(x, y)^2 &= g_B(x, y) - 2g_B(x, y) + g_B(y, y).
\end{align*}
\]

On a manifold of constant sectional curvature the sum \( g(x, x) + g(y, y) \) is constant. Locally, then, commute-time and biharmonic distance look like the harmonic and biharmonic Green’s functions (respectively), which can be expressed via one- and two-step backward Euler approximations of heat flow:

\[
\begin{align*}
g_C &= \lim_{t \to \infty} (\text{id} - t \Delta)^{1/2}, \\
g_B &= \lim_{t \to \infty} (\text{id} - 2t \Delta + t^2 \Delta^2)^{1/2}.
\end{align*}
\]

(Here \( \dagger \) denotes the pseudoinverse.)

3.4 Boundary Conditions

When computing the exact distance, either vanishing Neumann or Dirichlet conditions suffice since this choice does not affect the behavior of the smooth limit solution (see [von Renesse 2004], Corollary 2 and [Norris 1997], Theorem 1.1, respectively). Boundary conditions do however alter the behavior of our smoothed distance. Although there is no well-defined “correct” behavior for this smoothed function, we advocate the use of averaged boundary con-
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 [Chen et al. 2008], which for Poisson-type problems has guaranteed sub-quadratic complexity but in practice scales even better [Botsch et al. 2005]; moreover there is strong evidence to suggest that sparse systems arising from elliptic PDEs can be solved in very close to linear time [Schmitz and Ying 2012; Spielman and Teng 2004].

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.

In terms of absolute performance, a number of factors affect the run time of the heat method including the spatial discretization, choice of discrete Laplacian, geometric data structures, and so forth. As a typical example, we compared our simplicial implementation (Section 3.2.1) to the first-order fast marching method of Kimmel & Sethian [1998] and the exact algorithm of Mitchell et al. [1987] as described by Surazhsky et al. [2005]. In particular we used the state-of-the-art fast marching implementation of Peyré and Cohen [2005] and the exact implementation of Kirsanov [Surazhsky et al. 2005]. The heat method was implemented in ANSI C in double precision using a simple vertex-face adjacency list. Performance was measured using a single core of a 2.4 GHz Intel Core 2 Duo (Table I). 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 marching and the exact algorithm.

4.2 Accuracy

We examined errors in the heat method, fast marching [Kimmel and Sethian 1998], and the polyhedral distance [Mitchell et al. 1987], relative to mean edge length \( h \) on triangulated surfaces. Figures 21 and 22 illustrate convergence on simple geometries where the exact distance can be easily obtained. 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 – \( \text{MAX} \) is the maximum error as a percentage of mesh diameter and \( \text{MIN} \) is the mean relative error at each vertex (a convention introduced in [Surazhsky et al. 2005]). Note that fast marching tends to achieve a smaller maximum error, whereas the heat method does better on average. Figure 14 gives a visual comparison of accuracy; the only notable discrepancy is a slight smoothing at sharp cusps, which may explain the slightly larger maximum error exhibited by the heat method. Figure 15 indicates that this phenomenon does not interfere with the extraction of the cut locus – here we simply visualize values of \( \Delta \phi \) above a fixed threshold. Figure 23 plots the maximum violation of metric properties – both the heat method and fast marching exhibit small approximation errors that vanish under refinement. Even for smoothed distance (\( m \gg 1 \)) the triangle inequality is violated only for highly degenerate geodesic triangles, \( i.e., \) all three points on a common geodesic. In contrast, smoothed distances discussed in Section 2 satisfy metric properties exactly.
Fig. 13. Meshes used in Table I. Left to right: Bunny, Isis, Horse, Bimba, Aphrodite, Lion, Ramses1.

Table I. Comparison with fast marching and exact polyhedral distance. Best speed/accuracy in **bold**; speedup in *orange*.

<table>
<thead>
<tr>
<th>MODEL</th>
<th>TRIANGLES</th>
<th>HEAT METHOD</th>
<th>PRECOMPUTE</th>
<th>SOLVE</th>
<th>MAX ERROR</th>
<th>MEAN ERROR</th>
<th>TIME</th>
<th>MAX ERROR</th>
<th>MEAN ERROR</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bunny</td>
<td>28k</td>
<td>Fast Marching</td>
<td>0.21s</td>
<td>0.01s</td>
<td>(28x)</td>
<td>3.22%</td>
<td>1.12%</td>
<td>0.28s</td>
<td>0.16%</td>
<td>1.15%</td>
</tr>
<tr>
<td>Isis</td>
<td>93k</td>
<td>Fast Marching</td>
<td>0.73s</td>
<td>0.05s</td>
<td>(21x)</td>
<td>1.19%</td>
<td>0.55%</td>
<td>1.06s</td>
<td>0.60%</td>
<td>0.76%</td>
</tr>
<tr>
<td>Horse</td>
<td>96k</td>
<td>Fast Marching</td>
<td>0.74s</td>
<td>0.05s</td>
<td>(20x)</td>
<td>1.18%</td>
<td>0.42%</td>
<td>1.00s</td>
<td>0.74%</td>
<td>0.66%</td>
</tr>
<tr>
<td>Kitten</td>
<td>106k</td>
<td>Fast Marching</td>
<td>1.13s</td>
<td>0.06s</td>
<td>(22x)</td>
<td>0.78%</td>
<td>0.43%</td>
<td>1.29s</td>
<td>0.47%</td>
<td>0.55%</td>
</tr>
<tr>
<td>Bimba</td>
<td>149k</td>
<td>Fast Marching</td>
<td>1.79s</td>
<td>0.09s</td>
<td>(29x)</td>
<td>1.92%</td>
<td>0.73%</td>
<td>2.62s</td>
<td>0.63%</td>
<td>0.69%</td>
</tr>
<tr>
<td>Aphrodite</td>
<td>205k</td>
<td>Fast Marching</td>
<td>2.66s</td>
<td>0.12s</td>
<td>(47x)</td>
<td>1.20%</td>
<td>0.46%</td>
<td>5.58s</td>
<td>0.58%</td>
<td>0.59%</td>
</tr>
<tr>
<td>Lion</td>
<td>353k</td>
<td>Fast Marching</td>
<td>5.25s</td>
<td>0.24s</td>
<td>(42x)</td>
<td>1.92%</td>
<td>0.84%</td>
<td>10.92s</td>
<td>0.68%</td>
<td>0.67%</td>
</tr>
<tr>
<td>Ramses</td>
<td>1.6M</td>
<td>Fast Marching</td>
<td>63.4s</td>
<td>1.45s</td>
<td>(68x)</td>
<td>0.49%</td>
<td>0.24%</td>
<td>98.11s</td>
<td>0.29%</td>
<td>0.35%</td>
</tr>
</tbody>
</table>

But cannot be used to obtain the true geometric distance. Overall, the heat method exhibits errors of the same order and magnitude as fast marching (at lower computational cost) and is therefore suitable in applications where fast marching is presently used.

The accuracy of the heat method depends on the particular choice of spatial discretization, and might be further improved by considering an alternative discrete Laplacian (see for instance [Belkin et al. 2009b; Hildebrandt and Polthier 2011]). In the case of fast marching, accuracy is determined by the choice of update rule. A number of highly accurate update rules have been developed for regular grids (e.g., HJ WENO [Jiang and Peng 1997]), but fewer options are available on irregular domains such as triangle meshes, the predominant choice being the first-order update rule of Kimmel and Sethian [1998]. Finally, the approximate algorithm of Surazhsky et al. provides an interesting comparison since it tends to produce results more accurate than fast marching at a similar computational cost. However, one should be careful to note that accuracy is measured relative to the polyhedral distance rather than the smooth geodesic distance of the approximated surface (see [Surazhsky et al. 2005], Table 1). Similar to 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.

Fig. 14. Visual comparison of accuracy. **Left**: exact geodesic 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 exact distance – see Table I for a more detailed comparison.

1 Bunny mesh courtesy Stanford Computer Graphics Laboratory. Isis, Horse, Bimba, Lion, and Ramses meshes courtesy AIM@Shape. Aphrodite mesh courtesy Jotero GbR.
4.3 Robustness

Two factors contribute to the robustness of the heat method, namely (1) the use of an unconditionally stable implicit time-integration scheme and (2) formulation in terms of elliptic PDEs. Figure 16 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 17 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 great deal remains to be explored, including an investigation of alternative spatial discretizations. Further optimization 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 I). Another obvious question is whether a similar transformation can be applied to a larger class of Hamilton-Jacobi equations – for instance, a variable speed function might be incorporated by locally rescaling the metric. Finally, weighted distance computation might be achieved by simply rescaling the source data.

REFERENCES


### APPENDIX

#### A. A VARADHAN FORMULA FOR GRAPHS

**Lemma 1.** Let $G = (V, E)$ be the graph induced by nonzeros in any real symmetric matrix $A$, and consider the linear system $(I - tA)u_t = \delta$

where $I$ is the identity, $\delta$ is a Kronecker delta at a source vertex $u \in V$, and $t > 0$ is a real parameter. Then generically

$$\phi = \lim_{t \to 0} \frac{\log ||u_t||}{\log t}$$

where $\phi \in [0, \infty]$ is the graph distance (i.e., number of edges) between each vertex $v \in V$ and the source vertex $u$.

**Proof.** Let $\sigma$ be the operator norm of $A$. Then for $t < 1/\sigma$ the matrix $B := I - tA$ has an inverse and the solution $u_t$ is given by the convergent Neumann series $\sum_{k=0}^{\infty} t^k A^k \delta$. Let $v \in V$ be a vertex $n$ edges away from $u$, and consider the ratio $r_t := ||s||/||s_0||$ where $s_0 := (t^n A^n \delta)_v$ is the first nonzero term in the sum and $s := \sum_{k=n+1}^{\infty} t^k A^k \delta_v$ is the sum of all remaining terms. Noting that $||s|| \leq \sum_{k=n+1}^{\infty} t^k A^k \delta_v$ we get

$$r_t \leq \frac{t^{n+1} \sigma^{n+1} \sum_{k=n+1}^{\infty} t^k \sigma^k}{t^n (A^n \delta)_v} = c \frac{t}{1 - t \sigma},$$

where the constant $c := \sigma^{n+1}/(A^n \delta)_v$ does not depend on $t$. We therefore have $\lim_{t \to 0} r_t = 0$, i.e., only the first term $s_0$ is significant if $t$ goes to zero. But $\log s_0 = n \log t + \log (A^n \delta)_v$ is dominated by the first term as $t$ goes to zero, hence $\log (u_1)_v/\log t$ approaches the number of edges $n$. □

Numerical experiments such as those depicted in Figure 18 agree with this analysis. See also [Chebotarev 2011].

![Fig. 18. Isolines of $\log u_t/\log t$ computed in exact arithmetic on a regular grid with unit spacing ($h = 1$). As predicted by Lemma 1, the solution approaches the combinatorial distance as $t$ goes to zero.](image-url)
Fig. 19. Smoothed geodesic distance \((m = 1000)\) using “averaged” boundary conditions. Notice that increasing geodesic curvature along the boundary does not strongly influence the behavior of the solution.

Fig. 20. Mean percent error as a function of \(m\), where \(t = mh^2\). Each curve corresponds to a data set from Table I. Notice that in most examples \(m = 1\) (dashed line) is close to the optimal parameter value (blue dots) and yields mean error below 1%.

Fig. 21. \(L^\infty\) convergence of distance functions on the unit sphere with respect to mean edge length. As a baseline for comparison, we use the exact distance function \(\phi(x, y) = \cos^{-1}(x \cdot y)\). Linear and quadratic convergence are plotted as dashed lines for reference; note that even the exact polyhedral distance converges only quadratically.

ACKNOWLEDGMENTS

This work was funded by a Google PhD Fellowship and a grant from the Fraunhofer Gesellschaft. Thanks to Michael Herrmann for inspiring discussions, and Ulrich Pinkall for discussions about Lemma 1. 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/).

Received September 2012; accepted March 2013.

Fig. 22. Convergence of geodesic distance on the torus at four different test points. Error is the absolute value of the difference between the numerical value and the exact (smooth) distance; linear and quadratic convergence are plotted as dashed lines for reference. Right: test points visualized on the torus; dark blue lines are geodesic circles computed via Clairaut’s relation.

Fig. 23. Fast marching and the heat method both exhibit small violations of metric properties such as symmetry (top left) and the triangle inequality (top right) that vanish under refinement – we plot the worst violation among all pairs or triples of vertices (respectively) as a percent of mesh diameter. Dashed lines plot linear convergence. Bottom right: the triangle inequality is violated only for vertices along a geodesic between two distinguished points (in red), since the corresponding geodesic triangles are nearly degenerate. Bottom left: percent of red vertices as a function of \(h\) – each curve represents a different value of \(m\) sampled from the range \([1, 100]\).

Bunny mesh courtesy Stanford Computer Graphics Laboratory.