Geodesic Paths and Distances on Meshes: PDE and Computational Geometry Approaches

Report on *A Survey of Algorithms for Geodesic Paths and Distances*

Matthieu Pierre Boyer*
École Normale Supérieure
Paris, France
matthieu.boyer@ens.fr

Antoine Groudiev*
École Normale Supérieure
Paris, France
antoine.groudiev@ens.fr

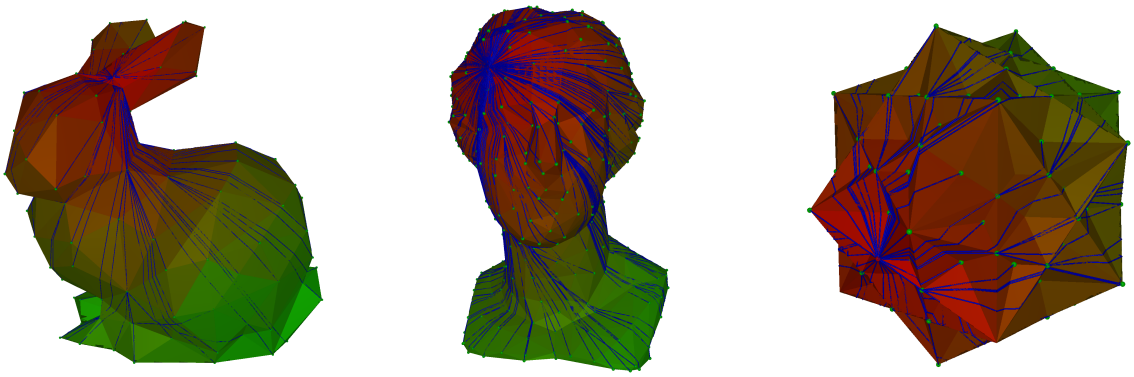


Figure 1: Geodesic paths on meshes, computed with our implementation of the Improved Chen-Han algorithm. Faces are colored from red to green according to the distance of their barycenter to the source point. The geodesic path to each vertex is drawn in blue.

ABSTRACT

In this report, we investigate different methods to compute shortest-paths on meshed 2-manifolds embedded in $\mathbb{R}^3$, based on [CLPQ20]. We will most notably compare different types of methods, either coming from the resolution of PDEs on the manifold, or through the unfolding of the embedding to $\mathbb{R}^2$. We also provide from-scratch implementations of five classes of methods, namely the heat method, the fast marching algorithm, the spectral embedding resistance distance transform, the Poisson equation border transform, and the Improved Chen-Han algorithm, and compare their performances on several datasets.

INTRODUCTION

Computing the distance between two points and retrieving a shortest path is, in general, a fundamental problem in computer science. While the problem is well understood in flat, Euclidean spaces, as well as on graphs, it becomes more challenging when considering curved surfaces. Indeed, the geodesic distance between two points on a surface is defined as the length of the shortest path that lies entirely on the surface. Computing such distances and paths is a non-trivial task, even when the surface is represented as a discrete mesh.

Such a problem arises in various fields, including computer graphics, robotics, medical imaging, and computer vision. In computer graphics, geodesic distances are used for texture mapping [OTC+10]; in robotics, they form the cornerstone of path planning algorithms on complex terrains [GMB13]; in medical imaging, they assist in analyzing anatomical structures [MTY06]; and in computer vision, they improve object recognition and shape analysis [SK17].

The ability to efficiently compute geodesic distances and paths is therefore crucial for many applications in which the size of the problem is large, and where real-time performance is required. In the polyhedral setting, where the surface is represented as a mesh of vertices, edges, and faces, the number of vertices is derived from the desired accuracy of the representation, and can thus be very large. Algorithms with a low theoretical and practical complexity are therefore essential to handle such large-scale problems.

After brief mathematical reminders in section 1, we present four classes of methods based on the resolution of Partial Differential Equations (PDEs) in section 2. Then, in section 3, we discuss computational geometry methods, focusing on the Improved Chen-Han algorithm. Finally, we present experiments and results in section 4, and conclude in the final section.

1 MATHEMATICAL REMINDERS

Formally, a 2-manifold (without boundary) is a topological space in which all points have neighborhoods homeomorphic to disks (without boundary) in $\mathbb{R}^2$. Such a homeomorphism is called a chart. Intuitively, this means that zooming enough on any point of the manifold will make it look like a flat surface. In Figure 2, we can see that some meshes are not manifolds, as some points are locally non-flat.

*Both authors contributed equally.

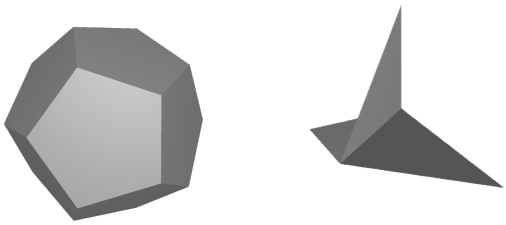


Figure 2: Examples of a 2-manifold (left) and non-manifold (right) mesh. (Images generated from our custom viewer.)

When considering a 2-manifold M, a (smooth) curve is a (smooth) application $\gamma : I \to M$ for an interval I. Two curves γ_1, γ_2 on the same interval $(-\epsilon, \epsilon)$ based at $q = \gamma_1(0) = \gamma_2(0)$ are equivalent if, in some chart, they have same first Taylor expansion around 0. Its tangent vector $\dot{\gamma}(0)$ at $q = \gamma(0)$ is the equivalence class[1] of γ in the space of smooth curves in M. The tangent space at $q \in M$ is the set T_qM of all tangent vectors of smooth curves based at q. It has the structure of a 2-dimensional vector space. A Riemannian tensor g on M is an application which to $q \in M$ associates a positive definite symmetric endomorphism of T_qM. A 2-manifold equipped with a Riemannian tensor g is called a Riemannian 2-manifold. We can define the length of a curve $\gamma : [a, b] \to M$ on such a manifold as

$$L(\gamma) = \int_a^b \sqrt{g_{\gamma(t)}\,(\dot{\gamma}(t), \dot{\gamma}(t))}\mathrm{d}t. \tag{1}$$

The geodesic (curve) between p and q is the curve γ verifying

$$\gamma = \arg \min_{\gamma:[0,1]\to M, \gamma(0)=p, \gamma(1)=q} L(\gamma).$$

The length of the geodesic between p and q is the called the geodesic distance. As presented above, computing the geodesic curves and the geodesic distances has multiple applications, and this is the problem we will be tackling in this report.

In our context, we will be given a 2-manifold already meshed[2], that is, a finite set $\mathbb{V} \subseteq \mathbb{R}^3$ of vertices (of cardinal $n_{\mathbb{V}}$) and a finite set $\mathcal{F} \subseteq [\![1, n_{\mathcal{F}}]\!]^3$ of faces, given by the indices of the associated vertices. The edges E of the manifold are given by any subsets of size two of a face $f \in \mathcal{F}$. Because continuous functions on the manifold cannot be fully represented in memory, and especially by a finite number of points on the mesh, functions on the manifold will be approximated by functions on $\mathbb{V}$, on E or on $\mathcal{F}$. As such, we will take any function and interpolate it linearly on each face, giving us a piecewise-linear function.

A path on a piecewise-linear manifold can then be understood as interpolating on pieces of the manifold, or directly computing the curves on the mesh. The quality of the approximation by the mesh of the 2-manifold will never be taken into account in the quality results. Finding the best approximation of a curved surface by a mesh is a whole other problem that we will not address in this report.

We now present two classes of methods to compute such geodesic paths and distances on a mesh.

[1] This allows us to define the (covariant) derivative of a vector field on a manifold, as well its gradient and Laplacian, but the mathematics behind these definitions are behind the scope of this report.

[2] [CLPQ20] gives a few methods to create such a meshing. In particular, our implementations of the methods are restricted without loss of generality to triangular faces; more general meshes can be considered through triangulation techniques.

2 PDE-BASED METHODS

In this section we will study methods inspired by Partial Differential Equations (PDE) that arise from models of physical phenomena. Indeed, many physical phenomena propagate along the surfaces over time, and dissipate over space, thus allowing us to retrieve geodesics from solutions to the equations.

2.1 General Theory

Consider the parabolic heat equation

$$\frac{\mathrm{d}}{\mathrm{d}t}u_t = \Delta u_t.$$

Here, u_t is the temperature profile at time t and Δ is the Laplacian operator (or divergence of the gradient operator). However, on a piecewise-linear (PL) manifold, because functions on vertices are interpolated linearly to become functions on the manifold, the gradient is piecewise-constant and can be computed explicitly from the values at each vertex. Consider a face $f \in \mathcal{F}$ with vertices $p_i, p_j, p_k \in \mathbb{R}^3$. Let $e_1 = p_j - p_i$ and $e_2 = p_k - p_i$. The face normal is

$$\overrightarrow{n_f} = \frac{e_1 \times e_2}{|e_1 \times e_2|},$$

where $\times$ is the cross product in $\mathbb{R}^3$ and thus the gradient, being perpendicular to level curves is

$$\nabla u_{|_f} = \frac{1}{2A_f} \sum_{l \in f} u_l (\overrightarrow{n_f} \times e_l), \tag{2}$$

where $A_f = \frac{1}{2}\,|e_1 \times e_2|$ is the area of face f and e_l is the **opposite** to vertex l. Note that we can represent the gradient as a matrix $G \in \mathbb{R}^{n_{\mathbb{V}} \times 3n_{\mathcal{F}}}$, although this representation is really inefficient for practical computation. The definition of the divergence then comes from the Gauss-Ostrogradski theorem by intregration by parts

$$(\nabla \cdot X)_i = \frac{1}{A_i} \sum_{f \ni i} \sum_{e \in f} \frac{1}{2} \cot\left(\alpha_e^f\right) \langle X_f, e \rangle, \tag{3}$$

where A_i is the Voronoi area associated with vertex i and α_e^f is the angle at the vertex opposite to edge e in f.

Finally, we can define the Laplace-Beltrami operator (the PL version of the continuous Laplacian) as $\Delta = (\nabla\cdot) \circ \nabla : \mathbb{V} \to \mathbb{R}$ or simply

$$(\Delta u)_i = \frac{1}{2A_i} \sum_{e=(i,j)} \left(\cot \alpha_{i,j} + \cot \beta_{i,j}\right)\left(u_i - u_j\right), \tag{4}$$

where $\alpha_{i,j}, \beta_{i,j}$ are the two angles opposite edge (i, j). One could then see the Laplace-Beltrami operator as a matrix $L \in \mathbb{R}^{n_{\mathbb{V}} \times n_{\mathbb{V}}}$.

After the spatial discretization of the laplace operator we just described, operating a time discretization in a single backward Euler step for some fixed time t will give us approximate solutions to the equation. If we want to find the distance maps from some set $X \subseteq \mathbb{V}$, we simply solve the linear equation associated to the continuous equation we need to solve.

2.2 Physical Equations

In our implementation, we compared multiple methods based on physical phenomena allowing to trace geodesics, or at least geodesic-like curves. Indeed, not all of those compute the true geodesic distance, but some sense of distance that can be drawn and integrated to find shortest paths.

The following equations all rely on the Laplace-Beltrami operator (or Laplacian operator Δ) which models the sum of second-order derivatives along each axis of a function. It is not difficult to understand how such an operator can mimic the propagation of a phenomenon in space. Indeed, the Laplacian of u at x measures the average extent of the deviation from the value of u near x to the value of u at x. As such, when we consider systems in which some quantity at a point is influenced by the value of the same quantity at nearby points (possibly with a time deviation), the Laplacian naturally occurs. Adding this to the symmetry at the core of the Laplacian's definition (in a local coordinate frame), when the laws of deviation are symmetric, which is often the case in physics, the Laplacian is bound to appear. From a more mathematical point of view, the Laplacian is the first scalar linear differential operator being covariant[3], and is fundamental as such.

Heat Method. This method is based on the heat equation

$$\nabla u_t = \frac{\mathrm{d}}{\mathrm{d}t} u_0, \tag{5}$$

which models the evolution of temperature profiles u_t in time in a given material, which here will be our surface, from the initial profile u_0. It can be derived from the first principle of thermodynamics on infinitesimal slices of material and describes the fact that spatial variations of temperature spread quadratically with regard to time. This means that spreading temperature from point A to point B will take of the order of magnitude of the square of the distance between A and B.

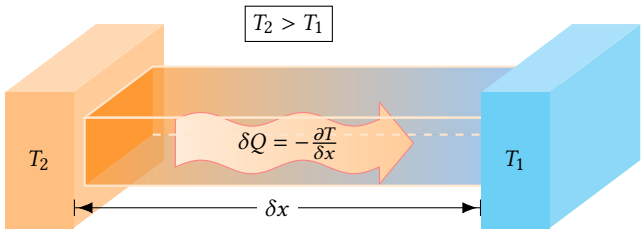


Figure 3: Illustration of the heat propagation between two slices of material of temperatures T_2 and T_1.

We can discretize the heat equation in time as

$$(\mathrm{id} - t\Delta) u_t = u_0. \tag{6}$$

Retrieving the true geodesic distance can then be done by first normalizing the gradient $X = -\nabla u_t / |\nabla u_t|$ of the solution of the above equation that points along geodesics, then solving the Poisson equation $\Delta\phi = \nabla \cdot X$ to retrieve the true distance function. This fact comes from the Varadhan formula $\phi(x, y) = \lim_{t\to 0} \sqrt{-4t \log k_{t,x}(y)}$.

[CWW17] suggest that the proper value of t to use for computations here is around h^2 with h the mean spacing between adjacent nodes, as $h\Delta$ is invariant with respect to scale.

Poisson Equations. The equations in this paragraph all allow to draw geodesics-like curves, although they do not give the actual metric on the manifold. They are all based on the Poisson equation

$$\Delta u = u_0, \tag{7}$$

where u_0 is distribution in space. This equation can be derived, for example, from the Maxwell equations to compute the electrostatic potential along a charge distribution, the equivalent theorems about gravitation, or from the momentum equation to compute the pressure in an incompressible fluid given its velocity.

The Poisson equations we consider are the screened Poisson equation and the border Poisson equation

$$\begin{cases} u & = 1 \text{ on } \partial M \\ \Delta u - \frac{1}{\rho^2} u & = 0 \text{ on } M \end{cases} \quad \text{and} \quad \begin{cases} u & = 0 \text{ on } \partial M \\ \Delta u & = -1 \text{ on } M \end{cases}$$

which both characterize a sense of distance u to the border conditions ∂M which in our case are the sources, the points we wish to consider distances to. This type of distance we call a border distance transform.

Spectral Methods. Like the previous class of equations, the spectral methods we will present below only allow to compute a sense of distance on the manifold, which is not the geodesic distance on the embedding in $\mathbb{R}^3$. However, unlike the previous methods, it will allow us to compute distances between every pair of points, and not simply from a set of sources.

The idea is to use the eigenvectors and eigenvalues of the Laplace-Beltrami operator Δ to create an embedding of our vertices. If we consider eigenspaces defined by

$$L\psi_i = \lambda_i \psi_i, \tag{8}$$

which we order by increasing eigenvalues λ_i, removing the vector associated with $\lambda_0 = 0$, since eigenspaces map (almost surely) the whole space, there are $n_{\mathbb{V}} - 1$ such vectors $\psi_1, \ldots, \psi_{n_{\mathbb{V}}-1}$. We consider an embedding size k (fixed in advance) and, because line i of the Laplace-Beltrami operator is concerned with the function at vertex i, we embed vertex i to the vector of the i-th coordinates of $\psi_1, \ldots, \psi_k$.

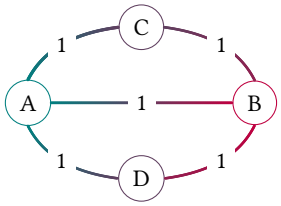
[BN03] argue that this embedding minimizes the ℓ^2 norm of the distortion when considering embeddings with eigenvectors of the Laplacian. We then simply need to define the notion of distance we want to consider on this embedding. We implemented multiple proposals to compute $\phi^2(x, y)$, which are basically all the sum of some transform of the eigenvalues λ_i multiplied by the squared ℓ^2 difference of the embeddings $(\psi_i(x) - \psi_i(y))^2$:

$$\begin{aligned} \text{(eigenmap)} \quad & \phi_E^2(x,y) = \textstyle\sum_{i=1}^k |\psi_i(x) - \psi_i(y)|^2 \\ \text{(commute time)} \quad & \phi_C^2(x,y) = \textstyle\sum_{i=1}^k \frac{1}{\lambda_i} |\psi_i(x) - \psi_i(y)|^2 \\ \text{(biharmonic)} \quad & \phi_B^2(x,y) = \textstyle\sum_{i=1}^k \frac{1}{\lambda_i^2} |\psi_i(x) - \psi_i(y)|^2 \\ \text{(diffusion)} \quad & \phi_D^2(x,y) = \textstyle\sum_{i=1}^k e^{-2t\lambda_k} |\psi_i(x) - \psi_i(y)|^2 \end{aligned} \tag{9}$$

[3] Considering the linear category of differential operators between tensor fields, we can prove that the Laplacian is generated by the covariant derivative ; which in Riemannian geometry can be thought as the derivative along a vector adjusted to respect the Riemannian tensor metric of the manifold.

To understand why these transforms of the eigenvalues which might seem a bit random do give a physical sense of distance between the two points, one can note that the commute time distance suggested above is actually equivalent to the effective resistance between x and y in an electrical network. This might be a more consistent distance model when modeling for example the most efficient path taking traffic into account, as it allows for multiple-route distance diminishment, as shown in [KR93] for example.

In this graph, the resistance distance between A and B is 0.5, while the geodesic distance is 1.



Similarly, the biharmonic distance arises from the equivalent definition when considering the bi-Laplacian Δ^2 which has the same exact eigenvectors but with squared eigenvalues. Those two distances can also be defined using a diffusion process (similarly to the equations in the previous paragraphs), as the expected value of the first arrival at y of a random walk on the mesh starting at x. For the diffusion distance, the discrete diffusion process used in the definition only considers the amount of information that has been diffused from x to y and from y to x at time t, and is as such directly tied to the heat equation.

As said, these methods do not contain the geodesics, but may be used as good approximations of the geodesic distance, or as regularized versions of the geodesic distance. Moreover, when considering k much smaller than $n_{\mathbb{V}}$, it is easy to approximate the distance quite well (see Figure Figure 11 below).

Wavefront Propagation. The hyperbolic wavefront propagation equation (or d'Alembert equation)

$$\frac{\mathrm{d}^2}{\mathrm{d}t^2} u_t = c^2 \Delta u_t, \text{or}, \Box u = 0$$

where $\Box$ is the d'Alembert operator, models the propagation of a wave in a material, which will again be our surface here. It arises for example the response of the surface to some elastic deformation u (say, a stone dropping in a pond) considering the stress tensor $T = E\nabla u$ with E the homogenous modulus of elasticity (here, the surface tension of the water) and considering the inertial force $\rho \frac{\partial^2 u}{\partial t^2}$ caused by the local acceleration (the stone moving the water down on impact).

However, while retrieving distance information from Equations (5) and (7) is not too difficult and doable by solving a simple linear system, retrieving distance from propagating wavefronts is more easily done from the non-linear hyperbolic Eikonal equation

$$\begin{cases} |\nabla u|^2 = 1 & \text{on } M, \\ u = 0 & \text{on } \partial M, \end{cases} \tag{10}$$

which directly characterizes the distance to the boundary ∂M (which again in our case will be the set we're computing the distance to). The Eikonal equation arises as the fundamental equation specifying the light paths in space given the refractive index function of the media encountered. In our case, there is a unique medium to be considered, and light sources ∂M that emit the light.

2.3 Implementation Details

Our input manifold is given as a triangular mesh, that is a set $\mathbb{V}$ of vertices of $\mathbb{R}^3$ and a set $\mathcal{F} \subseteq [\![1, |\mathbb{V}|]\!]^3$ of faces passed by indices. We are also given a set ∂M of sources or boundaries which will be points we compute the distance to (or from).

2.3.1 Linear equations. In the case of linear equations such as Equations (5) and (7), we only have to construct and solve a linear system of equations from the manifold and the set of sources:

- We compute the gradient for the manifold and a function u using the formula of Equation Equation 2, as the dynamic array of the face vectors.
- We also compute the Laplacian operator for the manifold as the $n_{\mathbb{V}} \times n_{\mathbb{V}}$ matrix using the cotangent formula of Equation (4) by iterating on all faces to iterate on all edges and thus all vertices in the end.
- If needed, we discretize in time the equation with a single step of backwards Euler method as done in Equation (6).
- We solve the induced linear system.

For the heat equation, we can also retrieve the actual geodesics passing through vertices by solving in ϕ the Poisson equation $\Delta\phi = \mathrm{div}X$ on the normalized gradient X of the temperature profile. This is done by simply computing the divergence operator on the manifold from Equation Equation 3, and iterating on the faces to fill the divergence matrix of the manifold and solving another linear system. For the Poisson method, a similar transform called the Spalding-Tucker transform exists to better approximate the geodesics. It is computed by $u \mapsto \sqrt{|\nabla u|^2 + 2u} - |\nabla u|$.

For the spectral methods, we compute the Laplacian matrix in the same fashion as above, and compute its eigenspaces (as eigenvalues and eigenvectors). We then sort those by increasing eigenvalues, skip the first one and compute the approximate geodesic distance for a given value of k.

2.3.2 Fast Marching Method. While the simple linear system solving works directly for the heat equation, solving for the geodesic paths arising from the Eikonal equation (10) is more difficult, as the equation is non-linear, and thus straightforward system resolution techniques will not work.

To work around this issue, we implemented the fast marching algorithm, based on [KS98]. The method initializes the distance at the sources ∂M to 0 and iterates using a priority queue on the neighbours of the already seen points. The distances are not interated according to paths along edges, but so that the Eikonal equation is verified. If we know values ϕ_i, ϕ_j at two corners, we pick ϕ_k so that the slope of the triangle (i, j, k) for ϕ (so the norm of the gradient of ϕ, $\|\nabla\phi\|$) is 1 as presented in Figure 4. A fun complement to the fast marching algorithm is that when the manifold is flat, i.e. when it is locally isometric to the Euclidean space, we return to the usual Dijkstra's algorithm.

2.3.3 Limitations.

Linear PDEs. The main issue that arises with the linear PDEs we consider is numerical stability. Indeed, the Laplacian of a manifold will usually be singular, so we need to add regularization terms at every linear system resolution. This yields a lot of numerical artifacts when the discretization steps are too big for the precision

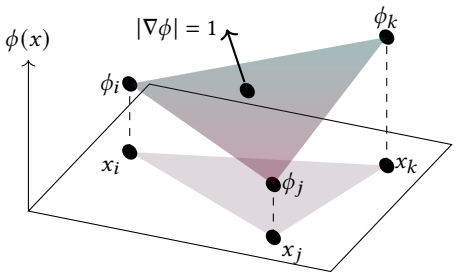


Figure 4: Representation of the Eikonal equation (10) $|\nabla\phi| = 1$ **on a triangular face** (x_i, x_j, x_k) **of a mesh.**

we can compute. Moreover, while the method is polynomial, it scales quite poorly as the number of vertices increases, the algorithm being in $O(n_{\mathbb{V}}^3)$.

Spectral and Poisson Methods. Some of the issues that arise with those methods are similar to those described above, as they are all based on the linear system solving, a slow process. However, the main issue with those methods is that they only allow to describe some distance transform from sources or some distance transform on the step. Those methods are thus not as well adapted to the problem of computing geodesics, although they solve a similar class of problems.

Fast Marching. For the fast marching algorithm, the method is much shorter, as it runs in (theoretical but impractical) $O(n_{\mathbb{V}} \log n_{\mathbb{V}})$, but this comes at the cost of stability when considering angles that are too small. Indeed, the order in which vertices are updated might violate the causality property of the propagation. If triangles have obtuse angles, there might exist a point i that is closer to the source than another point j, but the distance in j might be finalized before the distance at i. A solution to this exists but might not terminate due to its non-locality as suggested by [KS98], so in general one has to remesh the manifold or apply an iterative strategy as suggested by [BR06]. Moreover, this algorithm cannot be parallelized, unlike the above methods.

3 COMPUTATIONAL GEOMETRY METHODS

While PDE-based methods provide a general framework to compute geodesic distances on arbitrary, possibly non-meshed manifolds, we can leverage the fact that our manifold is meshed to design more efficient algorithms. A whole class of such algorithms, arising from computational geometry, can be classified according to two main caracteristics.

Global vs. local. Global methods compute globally optimal paths, up to numerical precision. Local methods start from an initial non-geodesic path and iteratively improve it until convergence to a local minimum.

Exact vs. approximate. Exact methods compute the exact geodesic distance, while some approximate methods trade accuracy for speed, and compute only an approximation of the geodesic distance within some guaranteed error bound.

In the following, we focus on a global and exact method, the Improved Chen-Han algorithm [CH90, XW09].

3.1 Continuous Dijkstra

The Improved Chen-Han (ICH) algorithm is based on the continuous Dijkstra paradigm introduced by Mitchell et al. for the Mitchell-Mount-Papadimitriou (MMP) algorithm [MMP87]. The main idea is to apply Dijkstra's algorithm in the continuous setting of a piecewise-linear manifold.

A first natural approach could be to consider the graph formed by the vertices and edges of the mesh, and run Dijkstra's algorithm on it. However, this would only yield paths that follow the edges of the mesh, which can be arbitrarily longer than the true geodesic paths on the surface (see Figure 5).

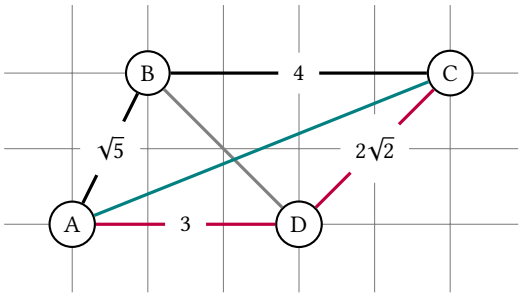


Figure 5: Example of suboptimality of applying Dijkstra's algorithm on the graph formed by the vertices and edges of the mesh. The red path follows the edges of the mesh, while the green path is the true geodesic path on the 2D surface, which is shorter.

Instead, we would like to consider each point on the surface as a potential node in the graph. However, since the shortest path on a single face is always a straight line, it is sufficient to only consider points on the edges of the mesh as potential nodes. This still yields an infinite number of potential nodes; a key concept of the MMP algorithm is to use instead a data structure called *windows*, which encapsulate a continuous interval of points on an edge, along with the information needed to propagate the shortest path from these points to adjacent faces.

Formally, a window

$$w = (s, p, e, b_0, b_1, d_0, d_1, \sigma)$$

is defined on edge e, between points b_0 and b_1 (see Figure 6). The window represents shortest paths originating from a "pseudosource" p, at a distance σ from the actual source s. Distances from p to the endpoints b_0 and b_1 of the window, respectively d_0 and d_1, are also stored.

3.2 Window management and pruning

While all revolving around the same core idea of continuous Dijkstra, different algorithms differ in how they manage and propagate windows [CLPQ20]. All *propagate* windows, that is, given a window on an edge, either (a) the window goes through a vertex v, and two new windows are created on the two edges adjacent to v in the face opposite to e; or (b) the window is contained in the opposite edge,

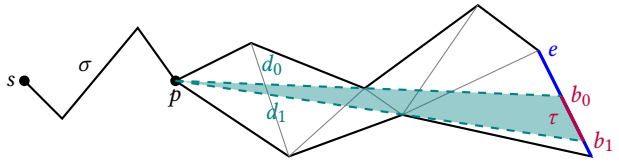


Figure 6: Illustration of a window $w = (s, p, e, b_0, b_1, d_0, d_1, \sigma)$.

and a single window is kept. However, the different algorithms differ in how they prune and prioritize windows to achieve better theoretical and practical complexity.

The original MMP algorithm does not perform aggressive pruning of windows, but trim overlapping windows on the same edge to avoid redundancy later on; this simple trimming strategy is illustrated in Figure 7. It uses a priority queue to always propagate the window with the smallest minimum distance first, ensuring both global optimality and early trimming of redundant windows.

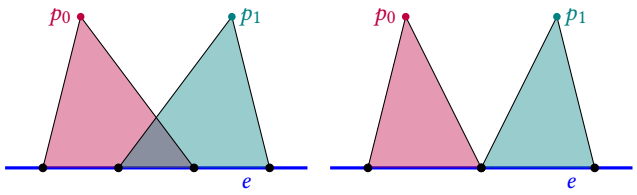


Figure 7: Window trimming in the MMP algorithm: overlapping windows on the same edge e (left) are trimmed to avoid redundancy (right).

To achieve a better theoretical complexity, the Chen-Han algorithm substitutes the priority queue with a First-In-First-Out (FIFO) queue, propagating windows in the order they were created. To speed up the algorithm in practice, a pruning rule called "one angle, one split" is introduced, which we illustrated in Figure 8; however, overlapping windows are not trimmed.

The Improved Chen-Han algorithm combines both approaches, using a priority queue to always propagate the window with the smallest minimum distance first, while also employing the "one angle, one split" pruning rule from Chen-Han. Another filtering strategy, illustrated in Figure 9, is introduced to discard windows that cannot lead to shorter paths than already explored window going through the same vertices.

We refer to [XW09] or to our implementation for details on the algorithm.

3.3 Implementation details

We implemented the Improved Chen-Hand algorithm from scratch in Rust, following the description from [XW09]. We provide here a short description of some implementation details, data structures used, as well as some difficulties we encountered and limitations of our implementation.

3.3.1 Data structures. As other methods, the input of the algorithm is a 2-manifold represented as a triangle mesh, that is a set of vertices $\mathbb{V}$ and a set of faces (3-tuples) $\mathcal{F}$. However, we also need to represent

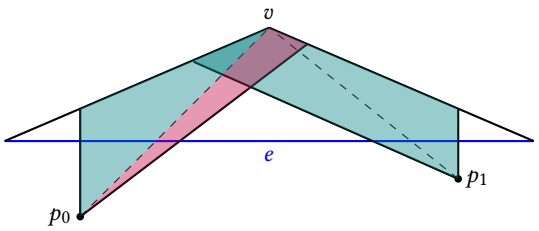


Figure 8: Illustration of the "one angle, one split" pruning rule: if two windows on the same edge e split on the same edge v, only three out of the four resulting windows are kept, as the one in red cannot lead to a shorter path.

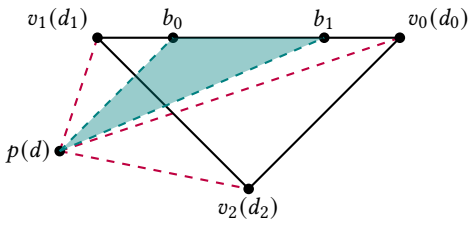


Figure 9: Illustration of the window filtering in ICH. For each vertex v_i, we denote by d_i the shortest distance to v_i found so far. If one the following inequalities holds, the window w can be safely discarded: $d + \|pb_0\| > d_0 + \|v_0b_1\|$, $d + \|pb_1\| > d_1 + \|v_1b_1\|$, or $d + \|pb_0\| > d_2 + \|v_2b_0\|$.

edges E explicitly, as windows are defined on edges. We therefore employ an edge data structure, storing for each edge both geometric information (length) and topological information (adjacent faces, vertices, …). Notably, we precompute the twin edges for each edge, which is repeatedly invoked during window propagation.

The main algorithm relies on a priority queue of windows, ordered by the minimum distance from the source to any point in the window. We implement this priority queue as a binary heap, using Rust's standard library implementation. Windows are represented as structs containing all the necessary information for propagation, as described above. Information about the vertices, such as the edge from which the shortest path to the vertex originates, is also stored during the main loop of the algorithm to enable path reconstruction later on.

3.3.2 Limitations. Two main limitations of our implementation are worth mentioning: paths along the edges fail to be computed successfully, and some vertices may not receive a distance value at the end of the algorithm when the geometry is particularly complex.

4 EXPERIMENTS AND RESULTS

In this section, we present experiments and results obtained from our implementations of the heat method, the spectral embedding distance transforms, the fast marching algorithm, and the Improved Chen-Han algorithm. All methods being implemented from scratch in the same programming language (Rust), and optimized up to a

similar level, we believe the comparison to be fairly representative of the practical performances of each algorithm.

The code for all implementations, as well as the scripts to reproduce the experiments, is available at:

`https://github.com/agroudiev/Geodesic-Paths`

4.1 Methodology

We collected a set of 27 meshes of various sizes, ranging from 4 to 125'066 vertices. Mesh come from various sources, including the Thingi10k dataset [ZJ16], filtered to only contain manifolds with a single connected component. Each mesh was processed by our implementation of each method, starting from a single source vertex chosen arbitrarily. For each mesh, each algorithm is run 10 times with different source vertices, each point corresponding to the average runtime over these runs. All experiments were conducted on a MacBook Pro with a 3.22 GHz Apple M1 Pro chip and 32GB of RAM. Plots are presented on a log-log scale, and when appropriate, a linear regression is fitted to the data points to estimate the practical time complexity of each method.

Care should be taken when interpreting the following results, as the runtime depends on the specific geometry of the meshes used in the benchmark, and may not generalize to all possible meshes. The number of vertices is not always a fair indicator of the complexity of the geometry, and other factors such as the distribution of vertex valences and curvature may also impact the practical runtime of the algorithm.

4.2 Benchmarking of linear PDE methods

Runtime of the Heat Method. In Figure 10, we present the results of our benchmark of the practical runtime of the heat method on various meshes. In theory, the method computes the laplacian in $O(n_{\mathbb{V}}^2)$ then solves two linear systems in $O(n^3)$.However, we do not observe a clear linear trend in log-log scale as a function of the number of vertices on the selected range of sizes. We believe this is due to the overhead of face-gradient, divergence and laplacian operators computation for small meshes, which takes a lot of added operations, and that the asymptotic behavior would be more visible on larger meshes.

Precision of Spectral Method. In Figure 11 we present how well the approximation using k eigenvectors for the spectral embedding described in section 2.2 and the different distance computations presented in Equation (9) compares to the full precision using the full $n-1$ eigenvectors non-associated to 0 as an eigenvalue.

The different colours represent the different methods and the different branches for each method are created from different models.

In terms of runtime, the duration is similar to the heat method as most of the computation time is spent computing the laplacian of the manifold, as well as its eigenspace decomposition which is, in theory, equivalent in time to the computation of a matrix multiplication (and thus near the resolution of a linear system).

4.3 Runtime of Fast Marching

The fast marching algorithm works really similarly to Dijkstra's algorithm by computing locally shortest paths. As such, it only

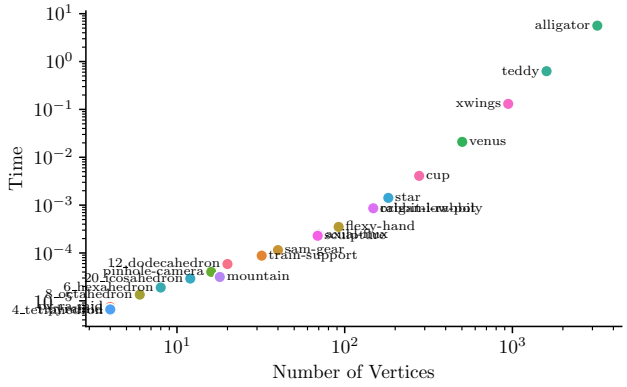


Figure 10: Benchmark of the time complexity of the *Heat method* on various 3D meshes, as a function of the number of vertices. TODO: add line to fit the second part

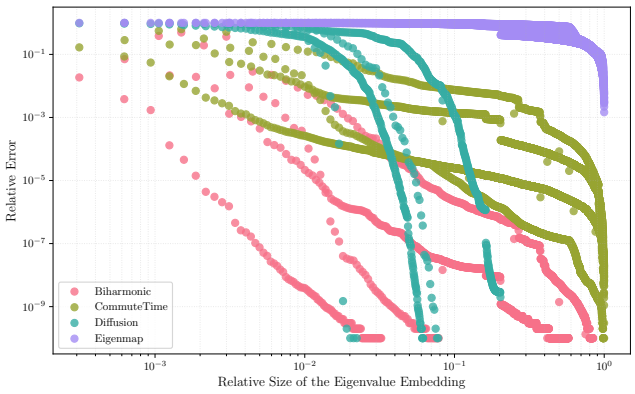


Figure 11: ℓ^2 error of the spectral embedding on k vectors to the full spectral embedding. Average computed on most of our manifold examples, as a function of the percentage of the dimension $n_{\mathbb{V}}$ used for the embedding size. TODO: increase font size in figure (decrease figsize in matplotlib), decrease points size

approximates geodesics (from the approximation on the mesh), but does so by performing at most one $O(1)$ operation on the minimum element of a priority queue on the vertices, which can be one vertex at most once. As such, in theory the algorithm runs in $O(n_{\mathbb{V}} \log n_{\mathbb{V}})$ when using a fully optimal priority queue. Figure 12 shows that the default implementation in Rust of the priority queues is actually a bit slower[4] and the algorithm thus runs in $O(n_{\mathbb{V}}\sqrt{n_{\mathbb{V}}})$ which is still sub-quadratic.

4.4 Runtime of the ICH algorithm

Theoretically, the Improved Chen-Han algorithm's time complexity is $O(n^2 \log n)$ on a mesh with n vertices, which is worse than the

[4]This is because the `BinaryHeap` data structure can be dynamically allowed and is as such a bit slower than the theoretical optimum speed.

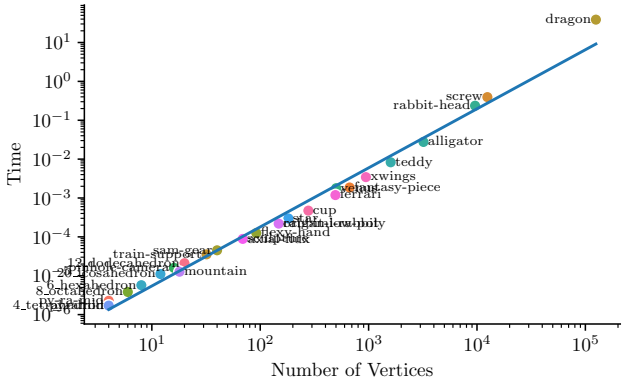


Figure 12: Benchmark of the time complexity of the *Fast Marching algorithm* on various 3D meshes, as a function of the number of vertices. Slope: 1.51, r^2**:** 0.98**.**

$O(n^2)$ previously obtained by the Chen-Han algorithm, the $\log n$ factor coming from the priority queue management.

[CLPQ20] reports that in practice, the ICH algorithm runs in sub-quadratic time on average, and dramatically improves the performance of the original Chen-Han algorithm, without supporting such a claim with a precise benchmark. We therefore conducted experiments to benchmark the practical runtime of our implementation of the ICH algorithm on various meshes, and report the results in Figure 13.

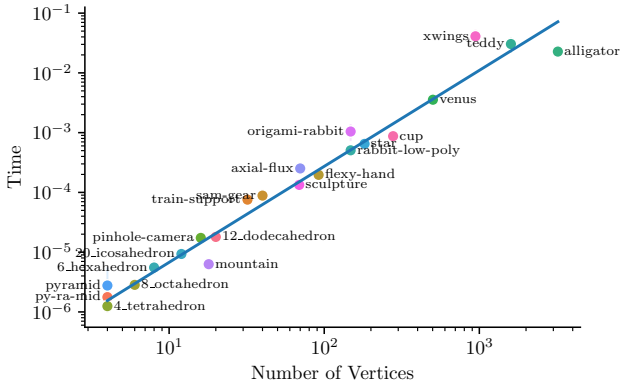


Figure 13: Benchmark of the time complexity of the *ICH algorithm* on various 3D meshes, as a function of the number of vertices. Slope: 1.60, r^2**:** 0.96**.**

The results confirm the sub-quadratic practical runtime of the ICH algorithm, with a fitted slope of 1.60 on a log-log scale, corresponding to a time complexity of approximately $O(n^{1.60})$ in practice.

4.5 Comparison and discussion

In Table 1 we present a short summary of all the methods we discussed in this report, as well as their limitations and time complexities.

Method	Theoretical Runtime	Actual Runtime	Precision	Limitations
PDE-based methods				
Heat	$O(n_{\mathbb{V}}^3)$		Approx.	Stability, runtime, fixed sources, no paths
Poisson	$O(n_{\mathbb{V}}^3)$		Approx.	Stability, not true geodesics, runtime, fixed sources, no paths
Spectral	$O(n_{\mathbb{V}}^3)$		Approx.	Stability, not true geodesics, runtime, no paths
Fast Marching	$O(n_{\mathbb{V}} \log n_{\mathbb{V}})$	$O(n_{\mathbb{V}}^{1.5})$	Approx.	Fixed sources, limited to meshes, no paths
Computational geometry methods				
ICH	$O(n_{\mathbb{V}}^2 \log n_{\mathbb{V}})$	$O(n_{\mathbb{V}}^{1.6})$	Exact	Limited to meshes, fixed (but possibly non-edge) sources

Table 1: Summary table comparing the different methods, their limitations and time complexity

We see in the table that two methods stand out: the Fast-Marching algorithm and the Improved Chen-Han algorithm, both running in around $O(n_{\mathbb{V}}^{1.5})$ time in practice. However, while the fast marching algorithm only allows to compute the distance from a fixed set of sources, the Improved Chen-Han algorithm computes all distances with better precision at the cost of a higher theoretical runtime, which in practice does not matter when pruning is done properly. It is also worth noting that ICH is the only algorithm to natively support path reconstruction; this can be done for some PDE-based methods at the cost of additional work.

matthieu tu peux préciser ça si besoin

CONCLUSION

In this report, we investigated different methods to compute shortest-paths on meshed 2-manifolds embedded in $\mathbb{R}^3$. We presented two classes of methods: PDE-based methods, which solve physical equations on the manifold to retrieve geodesic distances, and computational geometry methods, which leverage the meshed structure of the manifold to design efficient algorithms. We provided from-scratch implementations of three methods: the heat method, fast marching, and the Improved Chen-Han algorithm, and compared their practical performances on various datasets of meshes. Our experiments brought to light the practical runtime of each method, as well as their respective strengths and limitations.

REFERENCES

[BN03] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 06 2003.

[BR06] Folkmar Bornemann and Christian Rasch. Finite-element discretization of static hamilton-jacobi equations based on a local variational principle. *Computing and Visualization in Science*, 9(2):57–69, June 2006.

[CH90] Jindong Chen and Yijie Han. Shortest paths on a polyhedron. In *Proceedings of the sixth annual symposium on Computational geometry*, pages 360–369, 1990.

[CLPQ20] Keenan Crane, Marco Livesu, Enrico Puppo, and Yipeng Qin. A survey of algorithms for geodesic paths and distances, 2020.

[CWW17] Keenan Crane, Clarisse Weischedel, and Max Wardetzky. The heat method for distance computation. *Commun. ACM*, 60(11):90–99, October 2017.

[GMB13] Santiago Garrido, María Malfaz, and Dolores Blanco. Application of the fast marching method for outdoor motion planning in robotics. *Robotics and Autonomous Systems*, 61(2):106–114, 2013.

[KR93] Douglas Klein and Milan Randic. Resistance distance. *Journal of Mathematical Chemistry*, 12:81–95, 12 1993.

[KS98] Ron Kimmel and JA Sethian. Computing geodesic paths on manifolds. *Proceedings of the National Academy of Sciences of the United States of America*, 95:8431–5, 08 1998.

[MMP87] Joseph SB Mitchell, David M Mount, and Christos H Papadimitriou. The discrete geodesic problem. *SIAM Journal on Computing*, 16(4):647–668, 1987.

[MTY06] Michael I Miller, Alain Trouvé, and Laurent Younes. Geodesic shooting for computational anatomy. *Journal of mathematical imaging and vision*, 24(2):209–228, 2006.

[OTC+10] Guilherme N Oliveira, Rafael P Torchelsen, Joao LD Comba, Marcelo Walter, and Rui Bastos. Geotextures: a multi-source geodesic distance field approach for procedural texturing of complex meshes. In *2010 23rd SIBGRAPI Conference on Graphics, Patterns and Images*, pages 126–133. IEEE, 2010.

[SK17] Gil Shamai and Ron Kimmel. Geodesic distance descriptors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6410–6418, 2017.

[XW09] Shi-Qing Xin and Guo-Jin Wang. Improving chen and han's algorithm on the discrete geodesic problem. *ACM Transactions on Graphics (TOG)*, 28(4):1–8, 2009.

[ZJ16] Qingnan Zhou and Alec Jacobson. Thingi10k: A dataset of 10,000 3d-printing models, 2016.