Geodesic Paths and Distances

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

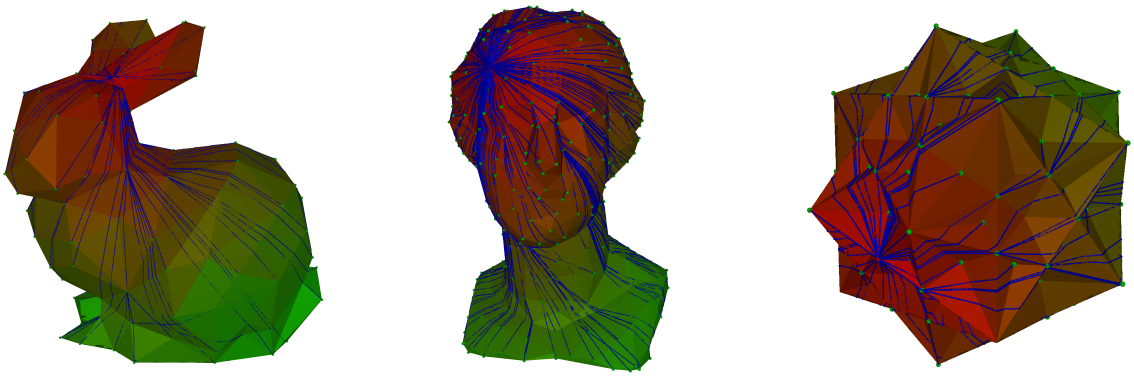

Figure 1: Geodesic paths on meshes, computed with our implementation of the Improved Chen-Han algorithm. Faces are colored from red to green according to the distance of their barycenter to the source point. The geodesic path to each vertex is drawn in blue.

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 three methods, namely the heat method, fast marching, 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, it becomes more challenging when considering curved surfaces.

TODO: mathematical problem formulation in general

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 two 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 $\mathbb{R}^2$. 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.

In our context, we will be given a 2-manifold already meshed[1], 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,

*Both authors contributed equally.

[1] [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.

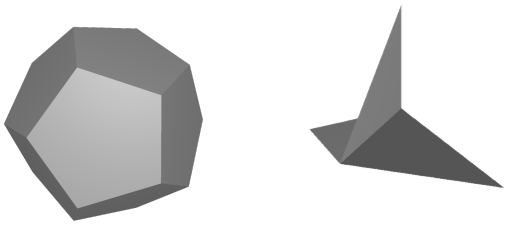

Figure 2: Examples of a 2-manifold (left) and non-manifold (right) mesh. (Images generated from our custom viewer.)

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.

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 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

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

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

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

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 piecewise-linear version of the continuous laplacian) as $\Delta = (\nabla \cdot) \circ \nabla : \mathbb{V} \to \mathbb{R}$ or simply

$$(\Delta u)_i = \frac{1}{2A_i} \sum_{e=(i,j)} \left(\cot \alpha_{i,j} + \cot \beta_{i,j}\right)\left(u_i - u_j\right), \tag{3}$$

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.

Heat Method. This method is based on the heat equation

$$\nabla u_t = \frac{\mathrm{d}}{\mathrm{d}t}u_0, \tag{4}$$

which models the evolution of temperature profiles u_t in time in a given material, which here will be our surface, from the initial profil u_0. It can be derived We discretize it as:

$$(\mathrm{id} - t\Delta)u_t = u_0. \tag{5}$$

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

$$\Delta u = u_0, \tag{6}$$

which can be derived, for example, from the Maxwell equations to compute the electrostatic potential along a charge distribution, or from the momentum equation to compute the pressure in an incompressible fluid given its velocity.

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 (4) and (6) 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

$$\begin{cases} |\nabla u|^2 = 1 & \text{on } M, \\ u = 0 & \text{on } \partial M, \end{cases} \tag{7}$$

which directly characterizes the distance to the boundary ∂M (which 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 (4) and (6), 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 1, 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 (3) by iterating on all faces to iterate on all edges and thus all vertices in the end.
- We discretize in time the equation with a single step of backwards Euler method and solve the induced linear system, as done in Equation (5).

For the heat equation, we can also retrieve approximate geodesics passing through vertices by solving in ϕ the Poisson equation $\Delta\phi = \div X$ on the normalized gradient X of the temperature profile. This is done by simply computing the divergence operator on the manifold from Equation 2, and iterating on the faces to fill the divergence matrix of the manifold and solving another linear system.

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 (7) is more difficult, as the equation is non-linear and thus easy system resolution techniques will not work.

To work around this issue, we implemented the fast marching method, 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.

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 artifact when the discretization steps are too big for the precision we can compute. Moreover, while the method is polynomial, it scales quite poorly as the number of vertices increases, the algorithm being in $O(n_{\mathbb{V}}^3)$.

Fast Marching. For the fast marching algorithm, the method is much shorter, as it runs in $O(n_{\mathbb{V}} \log n_{\mathbb{V}})$, but this comes at the cost of stability when considering too small angles. Indeed, the order in which vertices are updated might violate the causality property of the propagation. Indeed, if triangles have obtuse angles, there might be point i being 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.

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 3).

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

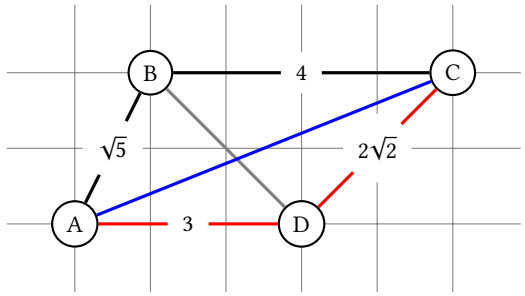

Figure 3: Example of suboptimality of applying Dijkstra's algorithm on the graph formed by the vertices and edges of the mesh. The red path follows the edges of the mesh, while the blue path is the true geodesic path on the 2D surface, which is shorter.

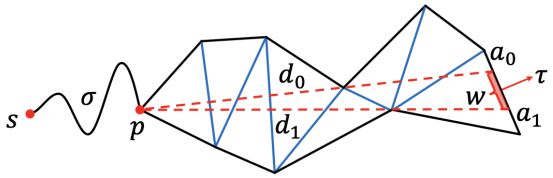

Figure 4: TODO: tikzfy. Illustration of a window $w = (s, p, e, b_0, b_1, d_0, d_1, \sigma)$.

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 4). 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].

The original MMP algorithm does not perform aggressive pruning of windows, but trim overlapping windows on the same edge to avoid redundancy later on. 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.

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; however, overlapping windows are not trimmed.

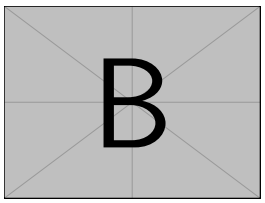

Figure 5: Window trimming in the MMP algorithm.

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. The pruning strategy is extended to trim overlapping windows on the same edge as well.

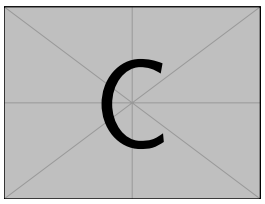

Figure 6: Extended "one angle, one split" pruning rule in the Improved Chen-Han algorithm.

3.3 Algorithm outline and example

We know give a high-level outline of the Improved Chen-Han algorithm, and illustrate it on a toy example.

TODO

3.4 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.

3.4.1 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 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.

3.4.2 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

TODO: do experiments, provide results, compare methods.

TODO: highlight the limitations of each method, and the trade-offs.

4.1 Methodology

We collected a set of TODO: number meshes of various sizes, ranging from TODO: min to TODO: max 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 TODO: number times with different source vertices, the points in Figure 7 corresponding to the average runtime over these runs.

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 Runtime of linear PDE methods

4.3 Runtime of Fast Marching

4.4 Runtime of the ICH algorithm

Theoretically, the Improved Chen-Han algorithm runs in $O(n^2 \log n)$ time on a mesh with n vertices, which is worse than the $O(n^2)$ 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 7.

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

CONCLUSION

TODO

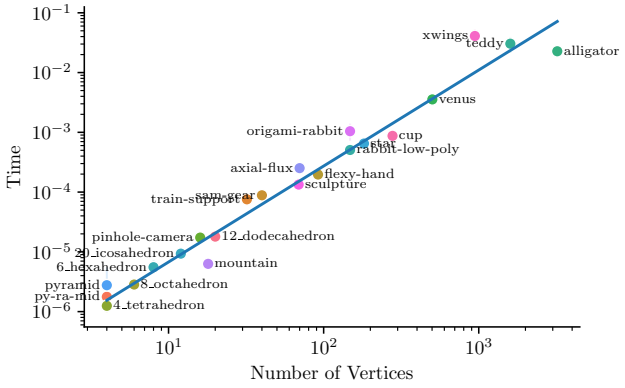

Figure 7: TODO: consider putting this in two columns Benchmark of the time complexity of the ICH algorithm on various 3D meshes, as a function of the number of vertices. Slope: 1.60, r^2**:** 0.96**.**

REFERENCES

[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.

[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.