Discrete geodesic graphs

Fang, Zheng

2019

Fang, Z. (2019). Discrete geodesic graphs. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/90288>

<https://doi.org/10.32657/10220/48533>

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

DISCRETE GEODESIC GRAPHS

DISCRETE GEODESIC GRAPHS

FANG ZHENG

FANG ZHENG

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

2019

2019

DISCRETE GEODESIC GRAPHS

FANG ZHENG

FANG ZHENG

School of Computer Science and Engineering

A thesis submitted to the Nanyang Technological University
in fulfilment of the requirement for the degree of
Doctor of Philosophy

2019

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

A handwritten signature in black ink, reading "Fang Zheng". Below the signature is a horizontal dotted line.

.. 28 May 2019..
Date

..... Fang Zheng

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

.. . 28 May 2019 ..
Date

..... He Ying

Authorship Attribution Statement

This thesis contains material from 3 paper(s) published in the following peer-reviewed journal(s) / from papers accepted at conferences in which I am listed as an author.

Please amend the typical statements below to suit your circumstances if (B) is selected.

Chapter 3 is published as Wang, X.; Fang, Z.; Wu, J.; Xin, S.-Q. & He, Y.

Discrete Geodesic Graph (DGG) for Computing Geodesic Distances on Polyhedral Surfaces. Computer-Aided Geometric Design, 2017, 52, 262-284

The contributions of the co-authors are as follows:

- A/Prof He provided the initial project direction and edited the manuscript drafts.
- I prepared the manuscript drafts. The manuscript was revised by A/Prof Xin and Dr Wang.
- I co-designed the experiments with Dr Wang and Mr Wu and performed all the experimental work at the School of Computer Science and Engineering. I also analyzed the data.
- Dr Wang implemented the algorithm. I assisted in the implementation and did the coding for experiments.
- Dr Wu assisted in the algorithm design.
- A/Prof Xin assisted in amendment of paperwork.

Section 5.1 is conditionally accepted as Yohanes Yudhi Adikusuma, Zheng Fang, and Ying He. Fast Construction of Discrete Geodesic Graphs. ACM Transactions on Graphics (TOG).

The contributions of the co-authors are as follows:

- Mr Yohanes provided the initial project direction and prepared the manuscript drafts.
- I edited the manuscript drafts. The manuscript was revised by A/Prof He.
- I designed the experiments performed all the experimental work at the School of Computer Science and Engineering.

- I implemented the algorithm with Mr Yohanes. I assisted in the implementation and did the coding for experiments.

Section 5.2 is published as Wensong Wang, Zheng Fang, Shiqing Xin, Ying He, Yuanfeng Zhou, Shuangmin Chen. Tracing High-quality Isolines for Discrete Geodesic Distance Fields. Computational Visual Media Conference 2019.

The contributions of the co-authors are as follows:

- A/Prof Xin provided the initial project direction and edited the manuscript drafts.
- I prepared the manuscript drafts. The manuscript was revised by A/Prof He and A/Prof Xin.
- I co-designed the experiments with Ms Wang and performed all the experimental work at the School of Computer Science and Engineering.
- Ms Wang implemented the algorithm. I assisted in the implementation and did the coding for experiments.

... 28 May 2019 ...
Date

..... Fang Zheng

Abstract

The discrete geodesic problem aims to find the shortest path and distance between arbitrary points on discrete surfaces. It is significant in a variety of computational geometry applications, such as surface parameterization and distance-based shape descriptors for shape analysis. As discrete surfaces are not able to generalize like parametric surfaces, computing the distance metric requires complex models and provides with huge possibilities to algorithm design. With the growth of computation capabilities, geometry processing applications incorporate with larger scale data and demand higher runtime performance, space efficiency, scalability and robustness. Therefore, ever since the initial introduction of the discrete geodesic problem, there are plenty of research constantly focusing on and improving these issues. To improve efficiency, a possible direction is to enhance information reuse. With precomputed essential information, queries are more responsive using more efficient algorithms. It may introduce error to the exact discrete geodesics and is regarded as trade-offs between the performance aspects and the accuracy. In contrast to the window-based exact algorithms, these methods are also called approximation algorithms. In this thesis, we discuss the related scheme in detail and further consider several applications and improvements. We conclude our contribution in three aspects:

First, I propose a graph-based approximation method for computing discrete geodesics. The scheme aims at enhancing information reuse for multiple distance queries meanwhile keeping the classical discrete wavefront propagation scheme. The wavefront propagation sticks to the geometric nature of any discrete surface, such that both the distance metric, robustness and accuracy is under control. We prove that a DGG on triangle mesh M has $O(\frac{n}{\sqrt{\varepsilon}})$ edges and the distance produced from DGG has $O(\varepsilon)$ relative error. We observe that DGG significantly outperforms several approximation methods including the saddle vertex graph (SVG) in terms

of accuracy controlling, theoretical soundness, time and spatial consumption. Meanwhile DGG possesses the same features for arbitrary points on mesh surface.

Second, I present an algorithm which is capable of computing accurate and robust discrete Green's functions for arbitrary points using DGG. We discover the connection between geodesic distance function and the Green's function by assigning geodesics to the Green's third identity on closed triangle meshes. We build a harmonic B-splines formation equivalent to the Green's third identity. The computational domain shrinks to a local patch as the basis function decays to zero outside one-ring neighbors of the patch. I introduce an optimization scheme to solve the localized harmonic B-spline equation and compute the discrete Green's function quickly, which is a better simulation of ground truth on synthetic models according to experimental results. I also present a refinement procedure for PVG mesh solver - a vector mesh editing method providing with better visual effects on rough models.

Third, I explore the improvement and extension to DGG. I improve the complexities of DGG algorithms. I also extend the scheme based on discrete geodesic graphs to point clouds. The proposed algorithm is conceptually simple yet effective. It extends a method which approximates the intrinsic distance with the Euclidean distance in an offset band but allows user to control the error convergence rate and performs the geodesic computation in a Dijkstra-like sweep. In this thesis, I also explore the possibilities in geometric applications based on discrete geodesic graphs. In addition to classical manifold-distance related topics such as Voronoi tessellation, I discover the internal relationship between the discrete Green's functions and the discrete geodesics. Another application is tracing the iso-curve of geodesic distance fields.

DGG provides a fast preprocessing method to compute discrete geodesic distance. The graph nature of DGG enables instant distance query. With the feature of the classical discrete wave-front propagation scheme, I observe that DGG is significantly competitive to the state-of-art preprocessing algorithms. The results of major extensions and improvements of DGG is also illustrated in this thesis.

Acknowledgments

I would offer my genuine gratitude to my supervisor Prof. He Ying for his patient guidance helped me during all the time of research and writing of this thesis. I am deeply indebted to his professional advice and care for my life. It is my luck of lifetime to be one of his students.

Thanks to all my labmates in MICL for the inspiring discussions and all the fun we had, in particular Dr Lu Xuequan, Dr Ma Long, Ms Du Jie, Ms Fu Qian and Ms Yao Sidan.

I cannot forget the beginning years of my study with Dr Sun Qian, Dr Hou Fei, Dr Wang Xiaoning, Dr Le Tien Hung, Dr Zhang Minqi, Dr Ying Xiang, and Mr Wu Jiajun. They led me to start as an amatuer and also help open my door to the research field.

I also wish to thank Dr Xin Shiqing, Ms Wang Wensong, Mr Adikusuma Yohanes Yudhi and all the friends from past collaboration.

I would like to express my love to my parents, my grandma and all my family. I am deeply indebted to them for supporting me spiritually throughout my growing up. Last but not the least, words cannot express my gratitude to Sun Yidan, my beloved girlfriend, for her encouragement, care and love.

Publications

Published & Accepted:

Wang, X., **Fang, Z.**, Wu, J., Xin, S. Q., & He, Y. (2017). Discrete geodesic graph (DGG) for computing geodesic distances on polyhedral surfaces. *Computer Aided Geometric Design*, 52, 262-284.

Hou, F., Sun, Q., **Fang, Z.**, Liu, Y., Hu, S., Hao, A., Qin, H., & He, Y. (2018). Poisson Vector Graphics (PVG). *IEEE Transactions on Visualization and Computer Graphics*, DOI: 10.1109/TVCG.2018.2867478, 1-1

Yohanes Y., **Fang, Z.**, & He, Y. Fast Construction of Discrete Geodesic Graph. (Conditionally accepted by *ACM Transactions on Graphics*, joint first author).

Wang, W., **Fang, Z.**, Xin, S., & He, Y. Tracing High-quality Isolines for Discrete Geodesic Distance Fields. (Accepted by CVM 2019, to be published in *IEEE TVCG*)

To be submitted:

Fang, Z., Lu, X. & He, Y., Discrete Green's Functions On Triangular Meshes.

Fang, Z., Lu, X. Leung, Y. & He, Y., Convolutional Neural Networks for 3D Point Clouds with Discrete Geodesic Graphs.

Contents

Abstract	i
Acknowledgments	iii
Publications	iv
List of Algorithms	ix
List of Figures	x
List of Tables	xvii
1 Introduction	1
1.1 Background	1
1.1.1 Geodesic Distances on Manifold Triangle Meshes	1
1.1.2 Discrete Green's Functions	3
1.1.3 Connection between Geodesic Distances and Green's Functions	6
1.2 Objectives	6
1.2.1 More Accurate Approximate Geodesic Distances	7
1.2.2 Discrete Green's Functions	9
1.3 Thesis Organization	10
2 Related Work	11
2.1 Discrete Geodesics	11
2.1.1 Window Propagation Methods	12
2.1.2 Differential Equation Methods	13
2.1.3 $(1 + \varepsilon)$ -Approximation Algorithms	14
2.2 Discrete Green's Functions, Harmonic B-splines and Applications	16
2.2.1 Discrete Green's Functions	16

2.2.2	Harmonic B-splines	17
2.2.3	Vector Graphics	18
2.2.4	Smoothed Distance	21
2.3	Extension Applications	22
2.3.1	Geodesics on Point Clouds	22
2.3.2	Isoline Extraction	23
3	Discrete Geodesic Graphs (DGG)	25
3.1	Motivation	25
3.2	Preliminaries	26
3.2.1	Geodesic paths & Discrete Wavefronts	26
3.2.2	Graph-Based Methods	29
3.3	Discrete Geodesic Graphs	31
3.3.1	Motivation & Definition	31
3.3.2	DGG Construction Algorithm	32
3.3.3	Handling Anisotropic Meshes	35
3.3.4	Property and Time Complexity	38
3.4	Computing Discrete Geodesics using DGG	40
3.4.1	DGG-Tailored Label-Correcting Algorithm	40
3.5	Results & Discussions	42
3.5.1	Graph Construction	46
3.5.2	Space Complexity	48
3.5.3	Runtime Performance	51
3.5.4	Approximation Error	52
3.5.5	Comparison with Other Methods	53
3.5.6	Applications	55
3.6	Summary	56
4	Computing Discrete Green's Functions using DGG	62
4.1	Motivation	62
4.2	Preliminaries	63
4.2.1	Discrete Green's Third Identity	63

4.2.2	Harmonic B-spline	65
4.3	Localized Solver for Discrete Green's Functions	66
4.3.1	Construction of Discrete Geodesic Distance Fields Between Arbitrary Points	66
4.3.2	Localized Harmonic B-splines	67
4.3.3	Optimization	72
4.3.4	Evaluation	73
4.4	Applications	75
4.4.1	PVG Refinement	75
4.5	Summary	77
5	Improvements and Extensions	79
5.1	Fast Construction of Discrete Geodesic Graphs	79
5.1.1	Overview	79
5.1.2	Runtime Performance	81
5.1.3	Robustness to Triangulation	81
5.2	Discrete Geodesic Graphs on Point Clouds	83
5.2.1	Overview	83
5.2.2	Accuracy	89
5.2.3	Robustness	90
5.2.4	Efficiency	90
5.3	Tracing Iso-distance Curve	95
5.3.1	Overview	95
5.3.2	Performance	97
5.3.3	Approximation Error	98
5.3.4	Extension to PDE methods	99
6	Conclusion and Future Work	103
6.1	Conclusion	103
6.2	Future Work	104
6.2.1	Closed-form Solver for Poisson Equations and Non-homogeneous Bi- Laplace Equations	104

6.2.2	Discrete Green's Functions for Bi-Laplace Operator	104
6.2.3	Unified Framework for Computing Discrete Distances	105
References		107

List of Algorithms

1	Constructing Discrete Geodesic Graph on Manifold Triangle Meshes	33
2	DGG-Tailored Label-Correcting Algorithm	43
3	Computing Geodesic Distance between Arbitrary Points	68
4	Refining PVG mesh	77

List of Figures

1.1	The single-source all destination geodesic distance fields associated with the Armadillo model. Results of DGG, SVG and HM-DM are compared to the exact geodesic distance from [67]. Compare to the fast computation (see timings in captions) of methods adopting preprocessing, it takes [67] 6.20 seconds to compute the distance. Due to the sensitivity to triangulation, the heat method uses Delaunay meshes [37] as input such that the result(illustrated as HM-DM) is more accurate. For clearer illustration of the difference between distances, I apply different texture color for distance iso-curves. The exact distance field is painted in green while the other comparison results are in red.	2
1.2	Three basis functions constructed by three columns of the pseudo-inverse matrix of the Laplacian matrix, according to [61]	4
1.3	The discrete Green's functions corresponding to the columns, according to [61]	4
1.4	The localization is based on harmonic B-spline. The figures illustrated by [25] show the components of biharmonic B-spline. Harmonic B-spline has the same components. Red color represents high function value while blue represents low value.	7

3.1	An illustration for basic primitives in discrete geodesics. (a) shows the edge $e_{ij} = (v_i, v_j)$ and the attached window $w = (e_{ij}, b_0, b_1, d_0, d_1, d)$ as well as the parameterization of w . (b) shows a geodesic path $\omega(p, q)$ passing through a saddle vertex s on a real world model. The surrounding angles of s is greater than 2π meaning that s is a saddle vertex and the path is indirect. (c) demonstrates the face sequence $\mathcal{F}(\omega) \subseteq F$ and its associated geodesic path. They are both unfolded to an Euclidean plane. (d) shows the geodesic distance wavefronts in continuous case and (e) shows the wavefronts in discrete case calculated by the MMP algorithm. Windows are rendered with red or yellow color for clear visualization.	28
3.2	(a) The tangent vector on a smooth manifold. Transportation along a geodesic maintain the parallel status. (b) The orange geodesic path is a direct one since there is no intermediate points. The long geodesic path $\omega(s, t)$ can be segmented into four direct short paths $\omega(s, v_1) \cup \omega(v_1, v_2) \cup \omega(v_2, v_3) \cup \omega(v_3, t)$. Each segment (v_i, v_{i+1}) simulates a part of $\omega(s, t)$	32
3.3	(a) The exact geodesic algorithms propagate from source vertex s all the way until w_1 "touches" the target vertex t and compute the direct geodesic path $\omega(s, t)$. (b) Our DGG window w_2 on the contrary terminates its propagation until an end point or a vertex u is able to estimate the remaining path within the given error bound ε	34
3.4	An illustration for edge pruning and pseudo edges. (a) shows the all the edges from the output of computing candidate edges. The redundant edges are rendered in orange and the edges in green are preserved after edge pruning. (b) demonstrates a typical redundant case $\omega(s, t)$ (rendered in orange). $\omega(s, u)$ and $\omega(u, t)$ approximate the long edge instead (rendered in green). (c)-(d) Inserting the blue pseudo edges of length l_ε . The pseudo edges ensure the accuracy. The angle between any two edges is less than $\sqrt{\varepsilon}$. (e) u is a non-vertex point on the mesh surface. After creating the $\omega(s, u)$ as a pseudo edge, starting from u , we build the red direct paths shorter than l_ε and insert all the new edges into the existing graph.	37

3.5	The label correcting method only allow distance update to occur inside the fan-shape region associated with each DGG edge. (a) shows a vertex p at the center of all the incident DGG edges. In (b), the fan-shape region is displayed along with the two respective edges $\omega(p, q_1)$ and $\omega(p, q_2)$	44
3.6	The DGG-tailored label-correcting has an empirical $O(n)$ time complexity on moderate-scale models. The timing increases flatly with decreasing user-specified accuracy ε	44
3.7	The DGG construction algorithm runs in $O(\frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$ empirically. (a) shows the construction time T_p with fixed n and varying ε . (b) and (c) show T_p with fixed ε but varying n . (d) Profiling of the DGG construction algorithm. The vertical axis is the ratio $\frac{T_p}{T_p+T_c}$, where T_p is running time of edge pruning and T_c is the time of computing candidate edges. Timing was obtained using single thread on an Intel Xeon E5-2609 v2 2.50GHz CPU.	45
3.8	Geodesics on models with sharp geometry. The accuracy control for DGG is $\varepsilon = 1\%$. The accuracy is not affected by geometric sharp features but the quality of mesh tessellation.	47
3.9	The average degree of DGG as well as SVG converges to an upper limit when the error is small enough or K is large enough.	48
3.10	Relation between the graph size or average degree of SVG and DGG and the actual relative mean error. DGG is a more compact graph than SVG under almost all error scenarios. With fixed average degrees, we observe a significant advantage of DGG that the accuracy is 2-5 times better than SVG.	49
3.11	The average degree of SVG is significantly affected by noise, since noise produces saddle vertices. In contrast, DGG is more robust to noise in terms of graph size.	50
3.12	Different type of vertex and the degrees statistics on the Bimba model with 74.5K vertices. DGG has smaller vertex degree variance than SVG.	50
3.13	The adding pseudo edge scheme is effective on improving the overall accuracy. τ measures the anisotropy degree for mesh triangulation. “w/” means with pseudo primitives and “w/o” means without those primitives. T_g is for distance querying time. $\frac{ V_p }{ V }$ is pseudo vertex ratio.	51

3.14	Runtime performance and error distribution. (a) shows the wavefronts resulted from FWP-MMP. From (b) to (f) is the error map of different methods rendered with colors. The precision of the heat method is improved by Delaunay meshes (DM). We adopt varied scales in the horizontal axes of error histograms for clearer visualization.	57
3.15	The experimental error $\bar{\varepsilon}$ and the user-specified tolerance ε . The empirical ratio $\bar{\varepsilon}/\varepsilon$ on the two representative models is between 0.3 and 0.6.	58
3.16	Differential equation methods show unstable performance in terms of accuracy with different level of geometric details.	58
3.17	The modified data structure for polyhedral path tracing.	58
3.18	Geodesics on a terrain model. Each path represented by gray tubes is calculated within 0.5 seconds. The model has 66,049 vertices.	59
3.19	Approximate geodesic loops simulate the tape and rope. A heuristic controls the smoothness of the ring when two shortest path meets at the same vertex. . .	60
3.20	The bijective boundaries on a hand-shaped model. The yellow dots represent the traced intersection point on mesh edges based on the modified data structure featured intersection information. The boundaries are then computed by connecting neighbor intersection points according to the unfolding sequence. .	61
4.1	The localization is based on harmonic B-spline. Harmonic B-spline relies on Green's functions. The figures showing the discrete Green's function on a triangle patch are illustrated by [15].	63
4.2	DGG is capable of computing geodesic distance between arbitrary points. (a) p is located in face $f \in F$. $p \in M \setminus V$ is an arbitrary non-vertex point. (b) The blue curves represent the new DGG edges emanating from p after re-triangulate f . (c) We compute the with good accuracy source p from the augmented graph. The results is comparable to the exact distance in (d).	67
4.3	The DGG result between arbitrary non-vertex points.	68

4.4	Farthest point sampling (FPS) on a model with $ V = 2.5K$. The FPS computes a new sample using the distance between existing ones. The feature of FMM and HM only allows them to compute distances between vertices. Given bad triangulation, the distance is not accurate and produces low quality sampling results. Red points indicate the final samples while the blue points are filtered out by FMM and HM. Our DGG with augmented graph produces good quality sampling compared to vertex-only methods FMM and HM.	69
4.5	Visualization of normalized Green's function calculated by different approaches. (a) Ground truth (b) Lipman's method with tessellation (c) Lipman's method with tessellation removing maximum value (d) Linear interpolation of neighboring vertices' GF using Lipman's method (e) Our method	74
4.6	Visualization of normalized Green's function on sphere calculated by different approaches for arbitrary points. The arbitrary points are samples in 100×100 pixels around the central point (a vertex). ε represents the normalized function error. For our method, distance from each pixel to the central vertex is computed first. The DGF is then induced by the distance. (a) Ground truth. (b) Lipman's method with subdivision surface. (d) Linear interpolation of DGF on neighboring vertices using Lipman's method. (e) Our method.	74
4.7	converging rates with random sampled source point within 0.2 radius (average edge-length and precision)	75
5.1	Runtime profiling on three representative models, Bunny $ V = 72K$, Armadillo $ V = 140K$, Gargoyle $ V = 350K$ (from left to right). Since the mesh quality is good, Wang et al.'s method skips the step of adding pseudo vertices and edges. W_d and W_i are the number of windows for our direct method and Wang et al.'s indirect method, and T_d and T_i are the run time. I observe that the performance improvement T_d/T_i is roughly proportional to the window ratio W_d/W_i	81
5.2	Comparing DGG with HM and its variants. For each row on the left is the results of HM and on the right is our results. From the error map we can see that, our method control the accuracy well.	82

5.3	Results on the subset of the Thingi10K shape repository, consisting of 4,439 models, each of which is connected, manifold triangle mesh. Among them, I further selected 590 models with low anisotropy measure $\tau \leq 3$ so that Wang et al.'s method can be applied. The gray planes denote the user specified tolerance ε for Wang et al.'s method and the accuracy parameter ε for our method. Each point corresponds to a 3D model. I color the models passing the test ($\bar{\varepsilon} \leq \varepsilon$) in green, and the ones failing the test ($\bar{\varepsilon} > \varepsilon$) in red. See the supplementary material for the detailed results.	84
5.4	Geodesic path on point cloud. The path (dashed brown) is more accurate than the path in blue, because some distances from $f(g_i)$ to G is large (in red). . . .	86
5.5	An example of searching neighbors for point χ_p . The points in green and yellow are directly connected to χ_p inside Ω^h . But only yellow points will be discarded if the distance is larger than K	87
5.6	Geodesic map on models. Top row: Exact geodesic computation. Middle row: approximated geodesic by our method. Bottom row: approximated geodesic by [39]. The mean relative error are reported in table 5.1.	91
5.7	Example of point clouds with holes on potential surface.	92
5.8	Example of noisy point clouds.	93
5.9	Centroidal Voronoi Tessellation constructed by (a) RVD-based method and (b) our method with 50 generators.	93
5.10	Relative error changes with different value of α . I plot the distribution of the distance error for the variable of points. The geodesic distance is computed from a single source point.	94
5.11	Yoga model with different value of h and α	94
5.12	The cell of P_1 is S_1 . The Apollonius vertices are shown in yellow, and the Apollonius edge are shown in green.	96
5.13	Algorithmic pipeline. (a) shows the input model. (b) shows a mesh triangle and a geodesic path (in red) and uses cubic function interpolation. (c) shows a set of sites (in red) in triangle. (d) builds Apollonius diagram for those sites. (e) builds the Apollonius vertices and Steiner point triangulation. (f) shows the isolines of the geodesic distance field.	96

5.14	Our method is able to compute high-quality isolines of geodesic distance fields on triangle meshes, whereas the simple linear interpolation fails to capture their geometric and topological features. To visualize the quality, I overlay the results with the exact solution. The red, black and blue-gray colors indicate the results of our method, the piecewise linear method and the exact solution. . . .	97
5.15	The memory and time of computing the isolines on the Buddha and Gargoyle model with 2,000 to 20,000 vertices and 0, 2, 10, 20, 50 Steiner points. . . .	98
5.16	The comparison of mean error between our method and the piecewise linear method. Our method (in lighter orange) significantly reduces the mean error compared with the piecewise linear method (in dark orange).	100
5.17	Visualization of our method with $k = 2$. The isolines of our method are rendered in black color and those of the piecewise linear method in red. Top Left: Beetle $ V = 5K$, $\varepsilon = 0.019\%$, 2 sources. Top Right: Buddha $ V = 5K$, $\varepsilon = 0.028\%$, 1 source. Bottom Left: Bottle $ V = 1K$, $\varepsilon = 0.096\%$, 3 sources. Bottom Right: Fertility $ V = 1.2K$, $\varepsilon = 0.074\%$, 2 sources.	101
5.18	Top: the Fast Marching Method. Bottom: the Heat Method. The isolines of our method are rendered in black and the piecewise linear method in red. . . .	102
6.1	Biharmonic distance field visualization from [35]. (a) Biharmonic distance from a source point (darkest blue) (b) Isolines follow the shape indifferently to the missing parts and holes (c) Earth-mover distances under different parameter conditions (d) Comparison of geodesic distances	106

List of Tables

3.1	Statistics of edge pruning.	48
3.2	The FIFO-queue statistics. Numbers are the account of elements entering the queue.	52
3.3	Typical discrete geodesic algorithms comparison. We assume that the comparison model has n vertices.	53
3.4	Mesh complexity and runtime performance. We select K for SVG in order to compare with DGG under close relative mean error (represented by $\bar{\varepsilon}$) for fair comparison. We name T_p for the preprocessing time and T_g for a single distance query time. Timings are measured in seconds. The experiments are conducted on an Intel Xeon 2609 v2 2.50 GHz CPU using a single thread with 32GB memory. τ is the anisotropy degree. $\bar{D}$ measures the average degree for SVG and DGG.	54
4.1	Nomenclature	64
5.1	Comparing the mean relative error of our methods with [39] on the same offset band resolution 512^3	91
5.2	Experiments on uniformly and non-uniformly distributed point clouds	91
5.3	Comparisons between different models in offset, β , ε , graph size and time factors. T_p represents pre-computation time in minutes. T_d represents Dijkstra time in seconds.	92

5.4 Runtime performance on 5 low resolution models and 5 higher resolution models. On each model there are $|V|$ vertices and k Steiner points on each edge. t and RAM show the running time and the peak memory usage of our algorithm. Timing unit is one second. The experiments are done on a workstation with an Intel Xeon 2.50 GHz CPU and 8 GB RAM. ε is the mean error of our algorithm and $\varepsilon - Linear$ is the mean error of the piecewise linear method. . . 99

Chapter 1

Introduction

1.1 Background

1.1.1 Geodesic Distances on Manifold Triangle Meshes

Measuring geodesic distances on polyhedral surfaces is a critical step in many geometry processing applications, such as shape analysis, parameterization, segmentation, matching and retrieval. From applications' point of view, a geodesic algorithm is evaluated in terms of the following criteria: (1) *Performance*: it is efficient to compute all types of geodesic distances, i.e., single-source, multiple-sources, and all-pairs distances. (2) *Accuracy*: it computes either exact distance or approximate solution with guaranteed error bound. (3) *Memory efficiency & scalability*: it is memory efficient and works well for large-scale models. (4) *Robustness*: it is insensitive to mesh tessellation and can also tolerate a certain degree of defects, such as noise, holes and gaps. (5) *Metric*: the computed distances are true distance metric, i.e., symmetric, satisfying triangle inequality and non-negative. (6) *Applications*: it is able to compute both geodesic distances and paths to feed various applications.

There are many approaches to compute geodesic distances on various discrete domains. Each method has its own advantages and drawbacks that may be suited to a particular application, but none of them satisfies all of the aforementioned criteria. The MMP and CH algorithms

Figure 1.1: The single-source all destination geodesic distance fields associated with the Armadillo model. Results of DGG, SVG and HM-DM are compared to the exact geodesic distance from [67]. Compare to the fast computation (see timings in captions) of methods adopting preprocessing, it takes [67] 6.20 seconds to compute the distance. Due to the sensitivity to triangulation, the heat method uses Delaunay meshes [37] as input such that the result (illustrated as HM-DM) is more accurate. For clearer illustration of the difference between distances, I apply different texture color for distance iso-curves. The exact distance field is painted in green while the other comparison results are in red.

compute exact geodesic distances if numerical operations are exact and can guarantee true distance metric, however, they are computationally expensive and work only for triangle meshes. The fast marching method and the heat method are efficient and work for both meshes and point clouds. But they are sensitive to mesh tessellation/resolution and lack of error bound. Moreover, they are mainly designed to compute geodesic distances rather than geodesic paths. It is known that tracing the gradient of geodesic distances on meshes, as a linear approximation, may produce geodesic paths of poor quality, since the distance field is highly non-linear. SVG is efficient, scales well, and can compute both geodesic distances and paths on triangle meshes. Unfortunately, its space complexity and runtime performance highly depend on the number of saddle vertices ¹ and their distribution. As a result, its performance can be compromised significantly if the input meshes have few saddle vertices or the saddle vertices are distributed non-uniformly. Furthermore, SVG does not have an error bound, although it works pretty well on common mesh models.

1.1.2 Discrete Green's Functions

Green's function has been used in solving differential equations and representing specific metrics in closed-form. For various geometric problems, Green's function has its application. On triangle meshes, current methods basically calculate the Green's function defined on the vertices. Originated from Yau's theory [9], the Green's function with no boundary value on a connected infinite graph τ is defined as

$$G \triangle (k, j) = I(k, j) - \frac{d_j}{vol} \quad (\text{Eq. 1.1})$$

where G is the Green's function and I is the identity matrix. d_j is the degree of vertex j and vol is the sum of degrees of τ . Similarly, on a closed 2-manifold $\mathcal{M}$, Lipman et al. [35] proves

¹The surrounding angles of a saddle vertex is greater than 2π .

Figure 1.2: Three basis functions constructed by three columns of the pseudo-inverse matrix of the Laplacian matrix, according to [61]

Figure 1.3: The discrete Green's functions corresponding to the columns, according to [61]

that the discrete Green's function g_d can be calculated by

$$L_c x = I - \frac{1}{N} \mathbf{1} \mathbf{1}^t \quad (\text{Eq. 1.2})$$

$$(g_d)_j = x - \frac{\mathbf{1}^t x}{\mathbf{1}^t \mathbf{1}} \mathbf{1} \quad (\text{Eq. 1.3})$$

where L_c is the conformal Laplacian matrix, N is number of columns and $(g_d)_j$ is the j_{th} column of g_d .

Lipman's method can effectively solve the Green's function on the vertices of a mesh. To solve the Green's function on arbitrary point domain still remains challenging. For applications like diffusion editing on meshes, although diffusion texture keeps the resolution independent and some other features of vector graphics [54], its texture mapping technique may cause undesired distortion. Users have to carefully configure the texture coordinates manually in order to achieve good visual effects. On 2-Manifold, a diffusion process is usually defined as

$$\triangle f = g \quad (\text{Eq. 1.4})$$

where g is a given function $g : \mathcal{V} \rightarrow \mathbb{R}^d$, $\mathcal{V}$ represents the vertices of a mesh. Solving such Poisson equations and find the value of f is the key to diffusion process. If Green's function G for Laplacian on $\mathcal{V}$ can be determined, the target function f will become a closed-form solution

$$f = Gg \quad (\text{Eq. 1.5})$$

If I define Green's function at random point on meshes, a potential application is, for example, utilizing the Green's function kernel inspired by diffusion textures, we can find an implicit representation of mesh color and edit intuitively on meshes. Moreover, for other applications such as earth-mover distance [52] and biharmonic distance [35] we may extend both distance to continuous region, which has significant meaning to the description of various types of distance fields.

1.1.3 Connection between Geodesic Distances and Green's Functions

We can find the underlying connection between discrete Green's functions and geodesic distance metric on compact 2-Manifold from Green's third identity [18]. Green's third identity in continuous manner is defined as

$$\begin{aligned}\phi_s(x) &= \iint_{\Omega} g(x, y) \triangle \phi_s(y) d\sigma \\ &+ \oint_{\partial\Omega} (\phi_s(y) \frac{\partial g(x, y)}{\partial n} - g(x, y) \frac{\partial \phi_s(y)}{\partial n})\end{aligned}\quad (\text{Eq. 1.6})$$

On a compact smooth 2-manifold $\mathcal{M}$, ignore the latter part

$$\phi_s(x) = \iint_{\Omega} g(x, y) \triangle \phi_s(y) d\sigma \quad (\text{Eq. 1.7})$$

Discretizing it on a closed triangle mesh $M = (V, E, F)$

$$\begin{aligned}\phi_s(x) &= c + \sum_y g(x, y) \triangle \phi_s(y) A \\ &= c + \sum_y g(x, y) A^{-1} L_c \phi_s(y) A \\ &= c + \sum_y g(x, y) L_c \phi_s(y)\end{aligned}\quad (\text{Eq. 1.8})$$

If we let function $\phi_s(x)$ be the geodesic distance between a source point $s \in V$ and an arbitrary point x , we may further calculate the Green's function defined on these two points. Combining the localization approach based on harmonic B-spline which is explained in Chapter 4, we can calculate arbitrary point-based Green's functions using geodesic distances.

1.2 Objectives

The aim is to build an algorithm for faster and more accurate geodesic distance computation in this thesis for further calculating Green's function and extend to more applications. The

Figure 1.4: The localization is based on harmonic B-spline. The figures illustrated by [25] show the components of biharmonic B-spline. Harmonic B-spline has the same components. Red color represents high function value while blue represents low value.

two key topics are geodesic distance and Green's function. The objectives include two parts: one is more accurate algorithms for geodesic distances, and the other is to efficiently compute discrete Green's function for arbitrary points based on the geodesics between arbitrary points.

1.2.1 More Accurate Approximate Geodesic Distances

To overcome the issue in SVG which intensely utilizes the distribution of saddle vertices, in this thesis I propose another graph based technique, namely discrete geodesic graphs (DGG). DGG keeps the majority of the ideal properties for geodesics. It can be adaptably developed utilizing any sorts of vertices on any polyhedral surfaces. DGG approximates a long geodesic path τ by a progression of *short* geodesic paths nearby τ , according to the parallel transport theory on smooth manifolds. This apparently minor change enables us to straightforwardly indicate the estimate error with a novel distinctive graph method. I demonstrate that the DGG related to M has $O(\frac{n}{\sqrt{\varepsilon}})$ edges, given a 2-manifold n -vertex triangle mesh M and precision control parameter $\varepsilon > 0$. There is also a firm $O(\varepsilon)$ error bound for the estimated geodesic distances on the graph. Computational outcomes demonstrate that the genuine error does not surpass 0.6ε on regular mesh models.

Exploiting DGG's extraordinary highlights, for fast distance querying, a DGG-tailored label correcting algorithm is designed for figuring geodesic distances. The speed is multiple times

faster than SVG. In spite of the fact that DGG is based upon mesh vertices, considering the nature of local window propagation scheme used by DGG, we can build a local editable augmented graph to register geodesics from arbitrary points without much of a stretch. As window propagation method maintains the information along the same face sequence, the lately inserted graph edge preserves the accuracy and other properties. I assess DGG's multifaceted nature and execution performance on a large group of commonly used triangle meshes. DGG fundamentally outflanks SVG as far as execution timing and memory usage, exactness control and theoretical complexities.

Following the previous discussion, I exceptionally focus on the geodesics between arbitrary points. In addition that the discrete geodesic graphs presented in the principle content can process all-pair geodesic problem, it is also a faster method for large-scale geodesics between non-vertex surface points. It is exceptionally required by a few applications to register geodesic distances for arbitrary surface points. Let $p_1, p_2 \in M \setminus V$ be two non-vertex primitives on $f_1, f_2 \in F$. One may just retrieve inexact $d_g(p_1, p_2)$ from bilinear interpolation, using the accessible geodesic distances among the face vertices of f_1 and f_2 . But on poorly tessellated meshes, bilinear interpolation creates poor outcome due to the non-linear feature of geodesic distances.

The geodesic triangle unfolding (GTU) method [66] creates preferred outcomes over bilinear interpolation. In any case, the estimated $d_g(p_1, p_2)$ has only first-order continuity.

I likewise present a viable strategy to approximate geodesic distances for arbitrary points with better precision. The idea is simply constructing the DGG edges from every source point p_i and combining with the existing graph covering the whole mesh. Given that the inserted geodesic paths from any point are *exact*, we then acquire $d_g(p_1, p_2)$ with the same estimation rate $1 + O(\varepsilon)$.

1.2.2 Discrete Green's Functions

In this thesis, I extend the domain of discrete Green's functions to pairs of vertex and any arbitrary point. I propose a novel method for calculation using the more accurate geodesic distances given by the first part. As a result I can provide closed-form solution of the Poisson equations on arbitrary points. As is described in the previous section, I discretize Green's third identity equation and use geodesic distance as left hand side function.

$$\phi_s(x) = c + \sum_y g(x, y) L_c \phi_s(y)$$

But directly solving $g(x, y)$ for all-pair is almost infeasible. With the growth of the model size, the dense matrix on the right hand side becomes too big to solve. As a consequence it is essential to simplify the computation cost. Harmonic B-spline is a perfect tool for the simplification. Harmonic B-spline function $\phi_s : \mathcal{M} \rightarrow \mathbb{R}^d$ has the form

$$\phi_s(x) = \sum_{i=1}^m \lambda_i \psi_i(x), \quad (\text{Eq. 1.9})$$

where $\lambda_i \in \mathbb{R}^d$ is the control point, and m is the number of control points.

The basis function for Vertices $V_i \in V$ is defined as

$$\psi_i(x) = a_i + \sum_j n_{ij} g(x, j), \quad (\text{Eq. 1.10})$$

where $\{V_j\}$ are one-ring neighbors of V_i .

Then I need to build the connection between $\phi_s(x)$ of Green's third identity and $\phi_s(x)$ of harmonic B-spline. Feng et al. [15] proved that harmonic B-spline function $\psi_j(rx)$ decays to zero quickly, which guarantees the local feature. Utilizing such feature of harmonic B-spline, I further narrow the calculation down to a local region, inside which I gather all the arbitrary point-to-vertex pairs (x, j) , and finally I find the solution of $g(x, j)$.

1.3 Thesis Organization

The thesis is organized into following structure: The related literatures and work are covered in Chapter 2 including discrete geodesics, discrete Green's functions and the related applications. Then I introduce two algorithms for geodesic: discrete geodesic graphs (DGG) for computing geodesic distances in Chapter 3 and discrete green's function in Chapter 4. Chapter 5 presents the improvement of DGG and extend the methods to point clouds and iso-curve tracing. Chapter 6 concludes all the algorithms, applications and extensions. Several future work is also propose in that chapter.

Chapter 2

Related Work

This chapter mainly surveys some related work in discrete geodesic distances, discrete Green's functions and two extension applications for geodesics: geodesics on point clouds and iso-curve tracing. As the concept of harmonic B-splines is the key to our localization technique and computational cost, we also include related work in harmonic B-splines.

2.1 Discrete Geodesics

The discrete geodesics solution can be generalized in two categories: window propagation methods and differential equation(DE or PDE) methods. From the perspective of distance accuracy there are exact polyhedral and approximation algorithms. The former one contains plenty of window propagation methods, while the latter one include most DE methods and a part of window propagation methods as well. Regarding the ability of controlling the error, literatures have proposed the specific $(1 + \varepsilon)$ -approximation algorithms among all the approximation algorithms. This section reviews the two brand categories and one closely related approximation approach. In Section 2.1.1 and Section 2.1.2, we list the publications of window propagation and DE methods respectively, which we adopt as comparison in the experiments.

Our method belongs to the $(1 + \varepsilon)$ -approximation algorithms a specific category among the approximation algorithms, which is covered in Section 2.1.3.

2.1.1 Window Propagation Methods

A window is a special kind of data structure that stores and describes the sweeping wavefront on mesh edges. Windows are propagated along the edge sequence from source vertices to others. From the 2D local coordinates stored in windows, we can directly compute the distance from source points to any point inside the windows. The first window propagation method was proposed by Mitchell et al. [41] namely the MMP algorithm. They build a priority queue for windows and maintain the order by distance. A new window nearest to the source point is propagated in each iteration. Another major algorithm is CH algorithm [8]. Unlike the priority queue used by MMP algorithm, CH adopts a tree structure and windows are updated using breadth first search. CH algorithm outperforms MMP algorithm in terms of time complexity, with $O(n^2)$ against $O(n^2 \log n)$. However as observed by Mitchell et al., on a mesh with n faces the most windows are $O(n^2)$ and therefore the theoretical time complexity remains $O(n^2)$ in the worst case. Surazhsky et al. [55] first implemented the MMP algorithm and proposed an approximate MMP algorithm by merging abundant windows. Their method has $O(n^{1.5} \log n)$ time complexity according to experiments on non-synthetic models. Similar to the idea of merging windows to reduce complexities, Xin et al. [65] presented a significant improvement to CH algorithm and there have been literatures improving the time performance by changing window merging strategy and parallelization [55] [65] [36] [38] [67] [46] [69]. Liu et al. [38] also discussed the window propagation method on degenerated models.

For special case like convex polytope and partial concave models such as uncrowded polyhedrons and terrains, Schreiber et al. [49] [48] proposed improvement work having time complexities as fast as $O(n \log(n))$. Due to their special computation domain and non-trivial implementation, the application is limited.

There remains discussion on improving the runtime performance for the window propagation methods. One possible direction was proposed by Ying et al. [68], their saddle vertex graph (SVG) allows empirical $O(n)$ for geodesic distance query. SVG propagates windows in local regions connecting source points and saddle vertices. The short geodesic paths are then combined as a large sparse graph covering the mesh surface. These procedures can be treated as preprocessing so that there are no propagation process in distance queries.

2.1.2 Differential Equation Methods

The geodesic distance satisfy the Eikonal equation $\|\nabla u(x)\| = 1$, therefore there are plenty of literatures on solving Eikonal equation to obtain the geodesics. Window propagation methods are usually computationally inefficient. On the contrary the differential equation methods run fast with light-weight implementation.

The fast marching method (FMM) proposed by Sethian et al. [50] uses the Dijkstra propagation strategy and finite difference gradient approximation. FMM computes the distance field on regular grids and solves the Eikonal equation with an $O(n \log n)$ time complexity. Tsitsiklis et al. [57] also proposed another discretized form of Eikonal equation. Based on Sethian et al.'s method, FMM was generalized to unstructured meshes [51], implicit surfaces [40], arbitrary triangulated surfaces [51], broken meshes [7] and parametric surfaces [53].

Ever since first proposal of FMM, there have been discussion on improving the runtime performance including parallelization. As the priority-queue based propagation scheme requires strict ordering with dependencies on previous queue elements, the paralleled implementation is difficult. Weber et al. [63] achieved timing improvement up to 4 orders of magnitude to the original FMM based on GPU computing. The method take geometry images as input. Using raster scanning on regular scatters, the parallel FMM reduces time complexity to $O(n)$. Other methods including [70], [20] and [13] also solve the Eikonal equation in a parallel manner.

In addition to the direct solver for Eikonal equations, there are also *indirect* solving methods. Xin et al. [64] proposed an iterative scheme that improved the distances from their normalized gradients with Euclidean distance initialization. The method lacks theoretical proof of convergence although empirically working well on different scenarios. Crane et al. [11] introduced the heat equation to the problem and presented a novel direction to the DE methods. Their idea is based on Varadhan’s formula [59], equalizing the normalized heat function gradients the geodesics gradients. The heat diffusion is calculated fast by solving a Laplacian equation. It then solves a Poisson equation with the divergence of normalized heat flow on the right hand side. The heat method works on various domains such as triangle meshes, non-planar polygonal meshes and point clouds. It is also a preprocessing method since the Laplacian matrix can be pre-factored, the distance query is in linear time using the factorization.

Belyaev and Fayolle in [3] minimized certain energy functionals and proposed an iterative variational approach to approximate the PDE-based distance computation. They also discussed on the extension to iteratively approximate other types of distance such as L_p -distance, p -Laplacian distance and Poisson distance.

The application domain of differential equation methods relies on the discretization of the differential operators, which provides possibilities on a variety of structures. The disadvantage of these methods on triangular meshes is that they are sensitive to triangulation qualities. The accuracy to exact polyhedral distance can be very low with bad mesh tessellations.

2.1.3 $(1 + \varepsilon)$ -Approximation Algorithms

As referenced above, estimating *exact* geodesic distances is costly. Despite the fact that the current PDE strategies and the SVG technique are effective, they need thorough analysis on accuracy. An ε -approximate path in computational geometry is defined as follows. Consider

the error $\varepsilon > 0$, if the length of a path between two points is at most $(1 + \varepsilon)$ times the exact geodesic distance, the path is ε -approximated.

Hershberger and Suri [23] exhibited the method to register a 2-approximate path on a *convex* polyhedron in $O(n)$ time. Be that as it may, their calculation does not appear to stretch out to yield better accuracy. Agarwal et al. [1] introduced a calculation that processes a ε -estimated path on a curved polyhedron in $O(n \log 1/\varepsilon + 1/\varepsilon^3)$ time for any endorsed $\varepsilon > 0$.

On polyhedral surfaces, the $(1 + \varepsilon)$ -approximations is proposed by Har-Peled et al. in [22]. Their algorithm firstly compute a Voronoi tessellation on polyhedral surface P , using the Voronoi diagrams they then subdivide P with a size of $O(n/\varepsilon \log 1/\varepsilon)$. Solving the Voronoi and subdivision is preprocessing in this algorithm with a time complexity of $O(n^2 \log n + n/\varepsilon \log(1/\varepsilon) \log(n/\varepsilon))$. The overall time complexity for single source all destination problem is $O(n \log(n/\varepsilon))$.

Two graph-based methods were proposed by Varadarajan et al. in [59] to compute the geodesics on non-convex polyhedrons. To construct the graphs a polyhedron mesh surface is first divided into small patches. Each patch is comprised less than r faces. A mesh with n vertices has $O(n/r)$ patches in total. A local graph is built on each patch connecting the center vertex with a set of Steiner points sampling from the boundary of the patch. The density of Steiner points is related with an accuracy control parameter ε . At last, the solitary graphs for local patches are then merged into one large-scale graph. Varadarajan et al. first proposed the primary algorithm with an $O(n^{5/3} \log^{5/3} n)$ time complexity. This algorithm incline its focus on accuracy and it can produce $(1 + \varepsilon)$ -approximation with ratio $7(1 + \varepsilon)$. Another algorithm was proposed later that reduces calculation time to $O(n^{8/5} \log^{8/5} n)$, yet increases the error to $15(1 + \varepsilon)$. These graph-based methods breakthrough the quadratic barrier for geodesic computation. Due to their low accuracy performance, there are opportunities for further constructive improvements.

2.2 Discrete Green's Functions, Harmonic B-splines and Applications

The text and results of the presented work in sub-section 2.2.3 have been published in [26].

2.2.1 Discrete Green's Functions

George Green [21] first introduced the concept of Green's functions. Since then Green's functions have been referred in many literatures. The calculation of Green's functions is a special topic in various scenarios. One of the most relative articles is the Green's functions on finite graphs by Fan et al. [9]. They proposed a sparse linear system for solving discrete Green's functions for graphs with or without boundaries. Feng et al. [15] discussed discrete Green's functions for laplacian and extend it to bi-laplacian. These Green's functions are usually integrations around the local area, as the value at center position of Green's functions domain is infinite. Lipman et al. [35] besides their work on closed-form biharmonic distances, they calculated such integration over local Voronoi cells assuming the area of each Voronoi cell is equal.

Green's functions have also been adopted by many recent geometric applications. Lipman et al. [34] proposed Green Coordinates and provided shape-preserving mappings for mesh deformation. They defined Green's Coordinates as cage-based integration of Green's functions. D'Eon and Irving [12] applied Green's function on a quantized-diffusion model for light scattering in translucent materials. Farbman et al. [14] built a typical diffusion model for images with the image gradients and the Poisson equations. The solution is the convolution of planar Green's functions and the divergence of the image gradients. Discrete Green's functions play particularly significant roles in defining distance metrics. Lipman et al. [35] introduced the concept of biharmonic distances. Based on the formation of Green's functions on graphs, they calculate discrete Green's functions defined on vertices of meshes simply by solving a sparse

linear system. Then they present a closed-form biharmonic distances representation by the integral of those Green's functions. Solomon et al. [52] overcame the drawbacks of geodesic and spectral distances by introducing the hybrid earth mover's distances. They provide a closed-form representation using the integral of the gradient of Green's functions.

Similar ideas giving closed-form solutions in other areas include diffusion textures by Sun et al. [54]. They adapt the Green's functions for the Laplacian in 2D space to model and render diffusion curve images, for which the underlying Poisson equations can be accurately modeled by Green's function kernels defined on control curves. They also derive an analytic solution for integrating the 2D Green's functions kernel contributions over a rectangular region and present a method to compute this in real time.

2.2.2 Harmonic B-splines

Harmonic B-spline can be used to define various kind of function values on triangle meshes. There have been literatures on harmonic B-splines and biharmonic B-splines. Besides discrete Green's functions for laplacian and bi-laplacian, Feng et al. [15] also mentioned the characteristic of harmonic B-splines. They prove that for discrete laplacian as a boundary operator which has linear precision, fix $x \neq 0$ and vary r towards infinity, the rate at which the harmonic B-splines function $\psi_j(rx)$ decays to zero is $O(\frac{1}{r^2})$. Based on the concept of biharmonic B-splines on planar fields, Hou et al. [25] extended the biharmonic B-splines to triangle meshes, and propose an algorithm to optimize the position of each knots, which enhances the performance of data fitting on the meshes. The control points only have influence over the local area of each Voronoi cell due to the local properties of biharmonic B-splines. Harmonic B-splines also have such properties. Our work transfer Green's third identity to equivalent form of harmonic B-splines and achieve local computation of discrete Green's functions.

2.2.3 Vector Graphics

Early vector graphics supports only linear or radial color gradients, which do not support smooth propagation of colors. Orzan et al. [43] pioneered diffusion curve images, which are curves with colors defined on either side. By diffusing these colors over the image, the final result includes sharp boundaries along the curves with smoothly shaded regions between them. Due to its compact nature and the ability of producing smoothly shaded images, diffusion curves quickly gain popularity in the graphics field and motivate recent work on improving its runtime performance and numerical stability, extending its expressiveness with more user control, and generalization to 3D and non-Euclidean domains, to name a few.

Solver. To rasterize a DCI, one needs to solve a Laplace equation defined on the entire image plane. Since a direct solver is expensive, Orzan et al. [43] designed a multigrid solver, which uses a coarse version of the domain to efficiently solve for the low frequency components of the solution, and a fine version of the domain to refine the high frequency components. Although being fast, this solver suffers from aliasing and flickering artifacts due to the rasterization of the curves over a discrete multi-scale pixel grid. Initializing the color at each pixel with the color of the closest curve, Jeschke et al. [29] used finite differences with variable step size to accelerate the convergence rate of Jacobi iterations. Their solver can guarantee the convergence to the right solution. However, it does not support zooming-in of arbitrary resolution, and the Laplace equation has to be fully solved even for local editing of curves and simple operations such as translation, rotation and scaling of the entire drawing. To overcome these limitations, Boyé et al. [6] developed a finite element method (FEM) based biharmonic equation solver, which dynamically converts a DCI into a high-order mesh-based representation that is automatically adapted to the complexity of the input curves. Such an intermediate triangulation, hidden from the user, is updated only when diffusion curves are edited. Moreover, their solver directly deals with gradient constraints.

Motivated by the parallel between diffusion curve rendering and final gathering in global illumination, Bowers et al. [5] developed a stochastic ray tracing method that allows trivial parallelism by using shaders and provides a unified treatment of diffusion curves with classic vector and raster graphics. However, it densely computes values even in smooth regions and sacrifices support for instancing and layering. Later, Prevost et al. [45] improved the ray tracing method by using an intermediate triangular representation with cubic patches to synthesize smooth images faithful to the per-pixel solution.

Another family of methods for rasterizing DCIs is to use boundary element method (BEM), which rephrases Laplace equation as a boundary integral along the control curves. Sun et al. [54] formulated the solution as a sum of Green’s functions of Laplacian. Thanks to the closed-form formula, this approach is quite fast and enables integrating the solution over any rectangular region, which allows anti-aliasing. However, it requires pre-calculating the weights of the Green’s function kernels, which depends on normal derivatives along the control curves. As a result, it can only take a DCI with fixed geometry and color constraints as input, and is not suitable for interactive authoring. Ilbery et al. [27] proposed BEM based solver for rendering biharmonic diffusion curves in a line-by-line manner. Their method is able to achieve real-time performance for rasterization and texture-mapping DCIs of up to millions of curves.

Most DCI solvers require GPU acceleration to achieve real-time performance. Aiming at efficiently rendering DCIs on devices with only a CPU, Pang et al. [44] developed a mesh-based approach, which sets the diffusion curves as constraints in triangulation and employs mean value coordinates interpolants to estimate mesh vertices’ colors. Their algorithm supports random access evaluation, but the produced DCI is only an approximation and it may suffer from the aliasing issue.

Gradient control. Laplacian diffusion does not natively support manipulation of the color gradient and gradient control can be achieved by higher-order interpolation. Finch et al. in [16]

improved the diffusion curves using biharmonic interpolation with a set of color constraints. Their method provides smooth interpolation for DC. Biharmonic DCIs allow more user control and also produce smoother results than DCI, however, they often produce unwanted extremals (hereby unpredicted effects) out of the user-specified regions due to violation of the maximal principle of harmonic equation. The shading curves proposed by Lieng et al. [33] introduce the concept of curve shading profiles. Users can manipulate the shading profiles that control the color gradient along the geometric primitives. However, the colors produced by shading curves are not as vivid as diffusion curves. Recently, Jeschke [28] proposed generalized diffusion curve images (GDCIs), which spatially blend multiple conventional DCIs. Thanks to more degrees of freedom provided by all DCIs, GDCI is able to provide a similar expressive power of color control as the biharmonic model and its solver is efficient and numerically stable. Since the blending functions are highly non-linear, which are obtained by solving a global Poisson equation and minimizing an energy functional, GDCI can only be used for reverse engineering, i.e., image vectorization, rather than design.

Extension. Diffusion curves have been extended to 3D volumes and non-Euclidean domains. Jeschke et al. [30] mapped diffusion curves to parameterized surfaces as textures, displacement maps and geometry images. Their algorithms for view dependent texture space warping and dynamic feature reconstruction allow sharp, high quality renderings of diffusion curve textures with a low base resolution. Formulating the diffusion process in terms of Green's functions, Sun et al. [54] showed that the image value at any given point can be directly evaluated. Such an explicit representation of DCI is ideal for rendering anti-aliased diffusion curve textures on 3D surfaces. The FEM based vectorial solver [6] also supports 2.5D images and 3D mapping. Takayama et al. [56] generalized diffusion curves to diffusion surfaces for volumetric modeling. Since solving a 3D Poisson equation is expensive, they adopted mean value coordinates to interpolate colors only locally at user-defined cross-sections. Since it avoids volumetric meshing, their algorithm is efficient for interactive modeling.

Recently Hou et al. [26] introduced a closed-form solver for Poisson equations and apply on the vector graphics. Their method allow more intuitive control on highlight and shadow using the new primitives namely Poisson Curve and Poisson Region. The closed-form solver is then extended to mesh domain [19]. The PVG mesh solver deploys the 2D primitives as color and displacement information associated with vertices on mesh. The mesh solver computes the discrete Green’s functions as preprocessing and therefore relies on the quality of mesh tessellation.

2.2.4 Smoothed Distance

Geodesic distance satisfies a metric on manifolds, but is sensitive to topology and is not smooth enough. To overcome these disadvantages, there were articles proposing various kinds of smoothed distance. Fouss et al. [17] introduced commute-time distance on graphs using the harmonic Green’s functions. It is a metric and shows smooth features. However it cannot be defined on surface since there is singularity at the source point. Its topological invariant characteristics brings difficulty in distinguish different shapes. Coifman et al. [10] proposed diffusion distance on surface based on heuristic parameters. Diffusion distance keeps the metric feature by adopting more eigenvalues and eigenvectors, but it still lacks of shape awareness. The biharmonic distance in [35] makes a balance between local and global features, letting it possess the most number of properties of a distance among all the smoothed distance methods. These methods rely on the discrete Green’s functions and have limited computation domains. Solomon et al. [52] introduced the concept of earth mover’s distances (EMD) on meshes considering probability distributions. A novel discretization using finite elements (FEM) is used to compute the differential formulation of EMD. The method provides flexibility to compute smoothed distances from the geodesic distance to bi-harmonic distance.

2.3 Extension Applications

2.3.1 Geodesics on Point Clouds

The algorithms mentioned in Section 2.1 are heavily dependent on the underlying mesh structure and hence unlikely to be adaptable to unstructured geometry such as point clouds. This section briefly reviews the algorithms that compute the geodesics on points.

Memoli and Sapiro [40] showed that the *intrinsic* distance functions on hypersurfaces (codimension one submanifolds of $\mathbb{R}^d$) without boundary can be accurately approximated by *extrinsic* Euclidean ones computed inside a thin offset band surrounding the manifold. Later, they extended the result to general codimensions and dealt with manifolds with or without boundary [39]. Based on the theoretical results, they developed an algorithm for computing geodesic distances on submanifolds of $\mathbb{R}^d$ given by point clouds. Their method avoids the need to explicitly find the underlying manifold, hereby skipping the manifold reconstruction step. Ruggeri et al. [47] proposed a variational approach for computing geodesic paths on point set surfaces, represented by surfels, i.e., oriented circular or elliptical disks. The energy, consisting of path length and closeness to the surface, is nonlinear. To avoid local minimal, the algorithm is initialized by a good piecewise linear path, computed by Dijkstra’s algorithm. Thanks to the optimization nature, their algorithm can compute a robust geodesic path with respect to certain degree of noise and outliers. However, it fails to compute the single-source geodesic problem due to its high computational cost. Hofer and Pottmann [24] computed a minimized energy based on a MLS surface of point cloud. The minimization of energy functional generates the optimized geodesic curves on the MLS surface. This method is sensitive to noise. Natali et al. [42] introduces Point Cloud Graph, a skeletal representation that generalizes the definition of the Reeb graph to arbitrary point clouds sampled from m -dimensional manifolds embedded in the d -dimensional space. The method is robust to non-uniform point distributions.

2.3.2 Isoline Extraction

The term “isoline” is used to denote a curve consisting of points at the same distance from a given point. Isolines are very useful in modeling levels of pollution, air pressure, and density of a mineral or any other feature. As a special case, when the real-valued function is the Z-axis component, the isoline map is able to characterize the change of height. Generally speaking, no explicit form exists for an arbitrary input scalar function. Therefore, the commonly used technique is to discretize the given domain into a set of simplices or small units and then extract isolines for each element.

When the real-valued function becomes the distance function, the isolines are equally spaced, which is an interesting property. In the field of computer graphics, geodesic is a useful tool for geometry analysis. In fact, geodesic is able to measure the distance between two points on a given surface, just like that straight lines are used to define the distance in Euclidean spaces. Various discrete geodesic approaches are proposed in recent decades to deal with the problems in digital geometry processing occasions. Isolines can serve as an intuitive means to visualize whether a geodesic approach is accurate or not. An important criterion is to check if the space between two successive isolines remains constant. Furthermore, geodesic isolines can also reflect the variations of the outer surface of a 3D model.

The most popular way to extract geodesic isolines consists of two steps [58]. First, we run a geodesic algorithm and keep the distance values at each mesh vertex. Second, for each mesh face, we trace the isoline, associated with a distance value d , constrained in that face f . In fact, the real distance distribution in f is unknown and therefore one assumes that the distance function linearly changes in f . Two typical situations may occur: (1) f doesn't contain a d -isoline, and (2) f contains a straight-line segment.

In spite of the resemblance between straight-line distances and geodesic distances, the distance changes are much more complicated in a curved space than in the Euclidean space. Geodesic

isolines may have sharp turns or topological changes due to geodesic “ridge” points that have two or more geodesic paths to the source point. We found that linear interpolation cannot well reconstruct the sharp features of isolines. A simple trick is to subdivide the input mesh surface to a very high resolution. However, those key points of isolines still cannot be accurately captured except the size of each triangle is as small as negligible. Furthermore, we need to re-compute the geodesic distance values for those newly inserted points, which requires a huge computational cost. We predict the change of geodesic distances along a mesh by a cubic interpolation technique and infer the sharp turns or topological changes of isolines via Apollonius diagrams.

Chapter 3

Discrete Geodesic Graphs (DGG)

The results of the presented work in this chapter and the text in Section 3.4, Section 3.5 and Section 3.6 have been published in [62].

3.1 Motivation

The objective of DGG in this chapter is to improve the shortcomings of other approximate algorithms such as SVG. The disadvantage of SVG is as follows. The graph is built upon existing saddle vertices from a mesh, which highly relies on the distribution of saddle vertices. For anisotropic meshes or non-uniform meshes the usage of SVG is inevitably restricted. A few models may have not many saddle vertices or unequal dissemination of saddle vertices. It is specially difficult to characterize saddle vertices on point clouds.

Based on the parallel transport theory (see Figure 3.2(a)) that while tangent vectors transport along a geodesic they stay parallel, the discrete geodesic graph segments a *long* geodesic path ω into many *short* geodesic paths which are close enough to ω . The angle between the short path and the exact long one quantitatively represents the "closeness". See Figure 3.2(b). DGG simulates a long geodesic path ω by a series of *short* geodesic paths whose endpoints are close

to ω . Exploiting DGG’s remarkable highlights, we build up a DGG-tailored label correcting algorithm for processing geodesic distances, which has 2 to multiple times speedup to SVG.

This chapter is organized into the following sections: Section 3.2 reviews the mathematical background. Then Section 3.3 introduces the discrete geodesic graph for 2-manifold meshes, analyzes its complexity and proves the error bound. Next, Section 3.4 presents the DGG-tailored label-correcting method for instant distance querying. Section 3.5 thoroughly evaluates DGG and SVG in terms of graph construction, runtime performance, error control and space complexity, and it also compares DGG with other approximate methods.

3.2 Preliminaries

3.2.1 Geodesic paths & Discrete Wavefronts

The following discussions are conducted based on the manifold triangular mesh M . V , E , F represent the vertex, edge and face sets on mesh M separately. According to the sum of surrounding angles of vertices, they can be categorized into Euclidean, saddle or spherical if the sum is equivalent to, greater than or less than 2π respectively. The geodesic path with ending nodes of any two point p and q is described as $\omega(p, q)$. It is possible that there are multiple geodesic paths over the surface. Therefore only the overall shortest path is referred to as $\omega(p, q)$. Mitchell et al. defined discrete geodesic paths as follows.

Theorem 1 [41]. *”A geodesic path on a mesh is an alternating sequence of vertices and (possibly empty) edge sequences such that the unfolded image of the path along any edge sequence is a straight line segment and the angle of ω passing through a vertex is greater than or equal to π .”*

$\mathcal{F}(\omega(p, q)) \subseteq F$ denotes the face sequence or a face set that is crossed by geodesic path $\omega(p, q)$. We use the hat symbol $\hat{x} \in \mathbb{R}^2$ to denote the element 2D unfolded image $x \in M$ throughout the paper. $\hat{v} \in \mathbb{R}^2$ represents the $\mathbb{R}^2$ unfolded image of vertex $v \in \mathcal{F}(\omega(p, q))$ on the face set. Likewise If it is unfolded to 2D faces containing $\hat{\omega}(p, q)$, the face sequence $\mathcal{F}(\omega(p, q))$ transfers to $\hat{\mathcal{F}}(\omega(p, q)) \subset \mathbb{R}^2$.

Here we discuss the discrete wavefront and wavefront propagation method. Given a source vertex s in vertices V ,

the combination of all the points with the same distance is called a wavefront. Due to the existence of saddle vertices, on a 2-manifold triangular surface the wavefronts do not share the same center in the source s , which only happens in the $\mathbb{R}^2$ Euclidean domain. Each wavefront segment has either a saddle vertex or s as the concentric center. The fundamental work completed by [41] and [8] maintains a discrete wavefront, a simplified version for the continuous case.

If we regard s as a point of light, all the emanating rays spreading across the virtual tangent plane of s would be geodesic paths. On discrete surface, the rays transmitted to different directions share the same origin information can be traced back using the same form of parameterization. The discrete wavefront propagation aims at gathering all such rays like intervals. Surazhsky et al. generalized [55] these intervals on the edges as windows. A window, as the word literally means, is a segment on an edge such that the geodesic path to any point on the segment trespasses the same sequence of faces and vertices. Figure 3.2.1(a) illustrates the concept of discrete geodesics. Different geodesic path may share one nearest original source point other than s . We call such point a *pseudo-source* point.

We quote the mathematical definition of window from [62] below. "A window w associated to an oriented edge $e_{ij} = (v_i, v_j)$ is a 6-tuple $(e_{ij}, b_0, b_1, d_0, d_1, d)$, where"

- " b_0, b_1 are the endpoints of w ;"

- “ d_0, d_1 are the distances from the pseudo-source to v_i and v_j ;
- “ d is the distance from the source to the pseudo-source.”

If $d = 0$ the window is *direct*, because source s directly light the window.

Figure 3.1: An illustration for basic primitives in discrete geodesics. (a) shows the edge $e_{ij} = (v_i, v_j)$ and the attached window $w = (e_{ij}, b_0, b_1, d_0, d_1, d)$ as well as the parameterization of w . (b) shows a geodesic path $\omega(p, q)$ passing through a saddle vertex s on a real world model. The surrounding angles of s is greater than 2π meaning that s is a saddle vertex and the path is indirect. (c) demonstrates the face sequence $\mathcal{F}(\omega) \subseteq F$ and its associated geodesic path. They are both unfolded to an Euclidean plane. (d) shows the geodesic distance wavefronts in continuous case and (e) shows the wavefronts in discrete case calculated by the MMP algorithm. Windows are rendered with red or yellow color for clear visualization.

Regarding the propagation of windows, a typical strategy is sweeping like Dijkstra’s algorithm. For example, the MMP algorithm maintains a priority queue $\mathcal{Q}$ storing all the windows and ordering the windows based on the calculated distance. A window from $\mathcal{Q}$ would extend to its one-ring neighbor edges and flip on the opposite side generating new windows. Once the newly created windows intersect with previous one, the algorithm recalculate the distance parameter

for the new window, and then insert back into the queue again. The above procedure loops itself until there is no elements in $\mathcal{Q}$. According to Mitchell et al. [41], for each edge there are at most $O(n)$ windows. The total space complexity is $O(n^2)$. Time complexity of the priority queue for a single insert operation is $O(\log n)$, thus the overall time complexity for MMP is $O(n^2 \log n)$.

3.2.2 Graph-Based Methods

The idea of graph-based methods is to describe the underlying surface geometrical information with graph. It is effective for geodesic problems since the graph edges share common properties with the geodesic paths. The geodesic distance on mesh M is denoted by $d_g(\cdot, \cdot)$ in this thesis. Given a graph G comprised of geodesic paths, the shortest distance on the graph is indicated by $d_s(\cdot, \cdot)$. Given two point u and v , the Euclidean distance is represented as $\|uv\|$.

It takes $O(n)$ time for a single source all destination distance query on a weighted complete graph $G_c = (V, E_c)$ with the edge set $E_c = \{e_{ij} | w_{ij} = d_g(v_i, v_j), \forall v_i, v_j \in V\}$. But the computation of constructing G_c is costly and not practical. As a consequence of saving computational expense and enhancing the precision, the graph based methods would extract G_c 's subgraph.

Aleksandrov et al. [2] introduced the graph method for $(1 + \varepsilon)$ -approximation geodesic distance. The algorithm designed a certain weighted graph on polyhedra $\mathcal{P}$. It computes the multiples of the travelled distance and the assigned weight on each passing face. They introduced Steiner points and placed them on each edge of $\mathcal{P}$. The Steiner points become the nodes of the graph, meanwhile the edges are simply all-pair connections among all the Steiner points. These nodes stick to the surface of $\mathcal{P}$. The placement of Steiner points follows a certain logarithmic rule and the geometry of each edge. Assume there are m Steiner points in total, their algorithm achieved $O(mn \log(mn) + nm^2)$ time complexity.

The GTU method [64] is a special type of graph-based methods. Given a mesh M , the algorithm use a user specified number (indicated by k) of sampling points $S = \{s_1, \dots, s_m\}$, distributed uniformly over the surface. It then connects the samples as edges of Delaunay triangles. Obviously each edge is geodesics. We illustrate the definition of GTU's graph $G_g = (V_g, E_g)$ here: The node set V_g of the graph is composed of sample points as well as vertices. There are two tiers in the edge set $E_g = E_{g1} \cup E_{g2}$, The edge set E_g contains two tiers. The first tier in the edge set is a interconnected graph of all sample points, $E_{g1} = \{e_{ij} | w_{ij} = d_g(s_i, s_j), \forall s_i, s_j \in S\}$, and each sample point is connected by mesh vertices as nearest neighbors, which is the second tier graph. Given the three nodes $s_{i_1} s_{i_2} s_{i_3}$ of the geodesic triangle, the second tier graph E_{g2} contains $\omega(v, s_{i_j})$. It takes $O(k^2 + n)$ space and $O(kn^2 \log n)$ time to construct the graph G_G . For any pair of vertices or non-vertex points p, q , After flattening the geodesic triangles from the curved surface to $\mathbb{R}^2$ domain using the edge length of E_{g1} , $d_g(p, q)$ is calculated simply by the Euclidean distance between the $\mathbb{R}^2$ images. Computing the local unfolded Euclidean distance takes $O(1)$ time for any single pair query. Therefore the overall time complexity of solving the single-source-all-destination (SSAD) geodesic problem is $O(n)$. The GTU algorithm controls the error of approximation using the scale of Delaunay triangles. The error bound is associated with $O(\frac{1}{\sqrt{k}})$. However despite its instant solution time, it has a quadratic space cost and hence has limited practice for large k or high accuracy. Along geodesic paths in addition to Euclidean vertices there are also saddle vertices.

Observations [68] show that on real world models a long geodesic path has a good chance to be a combination of shorter segments. Ying et al. [68] introduced a divide-and-conquer graph-based method for fast construction and solution utilizing the above feature. Represented by $G_s = (V, E_s)$ a saddle vertex graph nodes cover all the mesh vertices. E_s has three classes: the first class is for all direct saddle-vertex-only geodesic paths; the second class includes the direct non-saddle-vertex-to-saddle-vertex geodesic paths; and the third class contains the direct

non-saddle-vertex-only geodesic paths. The saddle vertex graph introduces a sparse geodesic graph G_s whereas G_g of GTU is a complete graph.

Nevertheless a genuine SVG consists of exact geodesic paths is not practical due to its high computational consumption. Ying et al. [68] introduced a heuristic value which control both the scale of the graph as well as the accuracy accordingly. The degree of nodes in G_s can not exceed K , which is a user provided parameter. SVG computes geodesics to K neighbor vertices around each vertex $v \in V$ and remove all the indirect geodesic paths. The size of G_s is simply $O(Kn)$. The time complexity for constructing G_s is $O(nK^2 \log K)$. Sometimes due to a non-uniform distribution of saddle vertices, the actual error may deviate under fixed K . In such case, it is better to adopt a more intuitive accuracy control scheme.

3.3 Discrete Geodesic Graphs

3.3.1 Motivation & Definition

According to [62], the definition of discrete geodesic graph is as follows.

Definition. "Given a manifold triangle mesh $M = (V, E, F)$ and the desired accuracy $\varepsilon > 0$, an undirected graph $G_\varepsilon = (V_\varepsilon, E_\varepsilon)$ is called discrete geodesic graph, if"

- (i) "the vertex set V_ε includes all mesh vertices and possibly pseudo vertices, i.e., $V_\varepsilon \supseteq V$;"
- (ii) "each edge in E_ε is a direct geodesic path; and"
- (iii) "the shortest path distances on G_ε are $(1 + \varepsilon)$ -approximation of the geodesic distances on M ."

The very basic distinction between DGG and SVG is that DGG uses every type of points on the surface as intermediate relay points. Relay points also refer to the pseudo points. The subtle adjust gives birth to a novel graph-based geodesic methods with completely distinctive features

to SVG. The relaxation in relay points endues rigorous error bound and complexity analysis and graph construction flexibility.

From a perspective of intuitive motivation, DGG should contain the direct geodesic path subset which is able to make $(1 + \varepsilon)$ -approximation of the geodesic distances. **For approximation error and space complexity analysis, an assumption of uniform vertex distribution on M is made. The assumption enables every point on the surface to be one of the relay points.** For meshes whose vertices are distributed highly non-uniformly, the technical assumption can be fulfilled by adding $O(n)$ uniformly distributed pseudo vertices on M . The new mesh, denoted by M' , has exactly the same geometry and combinatorial complexity as M . Then the discrete geodesic graph G'_ε associated to M' can be used to compute the geodesic distances on M .

We firstly present the construction algorithm in Section 3.3.2. The graph is constructed on meshes with regular triangulation. In Section 3.3.3 we extend our scheme on anisotropic meshes.

Figure 3.2: (a) The tangent vector on a smooth manifold. Transportation along a geodesic maintain the parallel status. (b) The orange geodesic path is a direct one since there is no intermediate points. The long geodesic path $\omega(s, t)$ can be segmented into four direct short paths $\omega(s, v_1) \cup \omega(v_1, v_2) \cup \omega(v_2, v_3) \cup \omega(v_3, t)$. Each segment (v_i, v_{i+1}) simulates a part of $\omega(s, t)$.

3.3.2 DGG Construction Algorithm

A naïve method for constructing DGG is to first compute *all* direct geodesic paths using an exact geodesic algorithm (e.g., MMP or ICH). Then for each direct path $\omega(p, q)$, check whether

Algorithm 1 Constructing Discrete Geodesic Graph on Manifold Triangle Meshes

Input: A manifold triangle mesh $M = (V, E, F)$ and the desired accuracy $\varepsilon > 0$

Output: The discrete geodesic graph $G_\varepsilon = (V_\varepsilon, E_\varepsilon)$

```
1:  $E_\varepsilon \leftarrow \emptyset$ 
2: for all  $s \in V$  do
3:    $E_\varepsilon \leftarrow E_\varepsilon \cup \text{COMPUTECANDIDATEEDGES}(M, s, \varepsilon)$ 
4: end for
5: for all  $s \in V$  do
6:    $\text{PRUNEREDUNDANTEDGES}(G_\varepsilon, s, \varepsilon)$ 
7: end for
8: for all  $s \in V$  do
9:    $\text{ADDPSEUDOVERTICESANDEDGES}(G_\varepsilon, M, s, \varepsilon)$ 
10: end for
```

there exists a vertex $r \in V$ such that $|d_g(p, q) - d_g(p, r) - d_g(r, q)| \leq \varepsilon d_g(p, q)$. If so, delete edge $\omega(p, q)$ from the graph. Although this method is conceptually simple and easy to implement, it is not practical for large-scale meshes due to the high computational cost of finding *all* direct geodesic paths. To construct DGG efficiently, we present two techniques, namely, accuracy-aware window propagation and error-controlled edge pruning. Algorithm 1 describes the framework of constructing DGG with detailed pseudo-code.

3.3.2.1 Accuracy-Aware Window Propagation

From previous observations we see that short direct geodesic paths can be connected in a progressive way to replace a long one. For an early termination of window propagation to avoid very long and unnecessary paths, the accuracy-aware window propagation scheme is described as follows. Given a long direct geodesic path $\omega(s, t)$, the container face sequence $\hat{\mathcal{F}}(\omega(s, t))$ transforms correspondingly elongated. See Figure 3.3, a relay vertex $\hat{u} \in \hat{\mathcal{F}}(\omega(s, t))$ is used to approximate the unfolded image of the path $\hat{\omega}(s, t)$. A threshold d_{max} is associated to each window w . d_{max} set the maximum propagating distance. Before the propagation let initial $d_{max} = \infty$, based on the appended theorem d_{max} for the child windows of w is gradually updated.

Theorem 2. [62] "Let s be the source point and $w = (e_{ij}, b_0, b_1, d_0, d_1, 0)$, a direct window associated to an oriented half-edge $e_{ij} = (v_i, v_j)$. Denote by l the length of edge e_{ij} . For any geodesic path $\omega(s, t)$ through window w , if $d_g(s, t) \geq \frac{l^2}{8\varepsilon(\min\{d_0, d_1\} - l/2)} + \frac{l}{2} + \max\{d_0, d_1\} (\triangleq d_{max})$, then $\omega(s, t)$ can be approximated by some vertex $u \in \mathcal{F}(\omega(s, t))$ with the approximation ratio $1 + \varepsilon$, i.e., $d_g(s, u) + d_g(u, t) \leq (1 + \varepsilon)d_g(s, t)$."

Figure 3.3: (a) The exact geodesic algorithms propagate from source vertex s all the way until w_1 "touches" the target vertex t and compute the direct geodesic path $\omega(s, t)$. (b) Our DGG window w_2 on the contrary terminates its propagation until an end point or a vertex u is able to estimate the remaining path within the given error bound ε .

3.3.2.2 Error-Controlled Edge Pruning

There are remaining edges that is redundant after computing the candidate edges. We design a scheme to remove these edges. If a DGG edge $\omega(u, v) \in E_\varepsilon$ can be approximated by two other DGG edges $\omega(u, r), \omega(r, v) \in E_\varepsilon$ with an approximation ratio $1 + \varepsilon$, it is *redundant*.

Let $\mathcal{E}_\varepsilon(v, w)$ denote an ellipse with eccentricity $\frac{1}{1+\varepsilon}$ and focal points v and w . The area of $\mathcal{E}_\varepsilon(v, w)$ is $\frac{1}{4}\sqrt{\varepsilon(2+\varepsilon)}(1+\varepsilon)\pi\|vw\|^2$. The sum of distances for any point on the ellipse to the two focal points is $(1+\varepsilon)\|vw\|$.

Now consider a geodesic path $\omega(p, q)$. If the ellipse $\mathcal{E}_\varepsilon(\hat{p}, \hat{q})$ contains a vertex, then Lemma A2 (see Supplementary Material in [62]) guarantees that $\omega(p, q)$ can be approximated by some relay vertex with an approximation ratio $1 + \varepsilon$. Intuitively speaking, the longer the path $\omega(p, q)$, the larger the area of the ellipse $\mathcal{E}_\varepsilon(\hat{p}, \hat{q})$, hence, more likely $\mathcal{E}_\varepsilon(\hat{p}, \hat{q})$ containing some mesh vertex, and the higher the probability that $\omega(p, q)$ is redundant.

Theorem 3. [62] *"Assume that the vertices are uniformly distributed on the mesh M . For an accuracy parameter $\varepsilon > 0$, the following statements are true:*

- (i) *For a direct geodesic path ω with length l , the probability of ω being a DGG edge is $O(e^{-\frac{(n-2)l^2\sqrt{\varepsilon}}{A}})$, where A is the area of mesh M .*
- (ii) *The graph G_ε has $O(\frac{n}{\sqrt{\varepsilon}})$ edges.*
- (iii) *For any vertices $p, q \in V$, the shortest path distance $d_s(p, q)$ on G_ε approximates the geodesic distance $d_g(p, q)$ on M with an approximation ratio $1 + O(\varepsilon)$."*

3.3.3 Handling Anisotropic Meshes

All the previous analysis has a pre-conditioner, that is the meshes have uniform vertex distribution. For anisotropic meshes where a non-uniform distribution is common, this assumption fails to hold. A naive method for anisotropic case is inserting uniform sampled vertices to the existing vertex $V \in M$. This satisfies the δ -sampling criteria. For any two vertices the distance in between is no more than δ -distance. After remeshing M with new pseudo vertices into M' , the geometry of M' is identical with M . Then, distances on M' with approximation ratio $1 + O(\varepsilon)$ is computed on the discrete geodesic graph G'_ε . G'_ε can also approximate the geodesic distances from V with the same level of error, given that $V \subset V'$.

Due to the high computational cost in sampling and re-triangulation the above approach is not practical if the model is large. I propose a new scheme in this subsection for handling

```

1: function COMPUTECANDIDATEEDGES( $M, s, \varepsilon$ )
2:   for all edge  $e_{ij} = (v_i, v_j)$  opposite to  $s$  do
3:     Create a window  $w = (e_{ij}, v_i, v_j, \|sv_i\|, \|sv_j\|, 0)$ 
4:     Place window  $w$  into a priority queue  $\mathcal{Q}$  keyed by its distance to source  $s$ 
5:   end for
6:   repeat
7:     Extract the top window  $w$  from  $\mathcal{Q}$ 
8:     //  $w = (e_{ij}, v_i, v_j, d_0, d_1, 0), l = \|v_i v_j\|$ 
9:      $d_{max} \leftarrow \frac{l^2}{8\varepsilon(\min\{d_0, d_1\} - l/2)} + \frac{l}{2} + \max\{d_0, d_1\}$ 
10:    Propagate window  $w$  across its adjacent triangle
11:    Place its child windows into an array  $L$ 
12:    if window  $w$  lights a vertex  $v \in V$  then
13:      Update geodesic distance  $d_g(s, v)$ 
14:      Create a DGG edge  $(s, v)$  if it does not exist yet
15:    end if
16:    for all child window  $w' \in L$  do
17:      //  $w' = (e', b'_0, b'_1, d'_0, d'_1, d')$ 
18:      if  $d'_0 < d_{max}$  and  $d'_1 < d_{max}$  and  $d' = 0$  then
19:        // Only direct windows are added to  $\mathcal{Q}$ 
20:         $\mathcal{Q}.\text{push}(w')$ 
21:      end if
22:    end for
23:  until  $\mathcal{Q}$  is empty
24:  return the computed DGG edges incident to  $s$ 
25: end function

```

anisotropic triangulated meshes. For meshes with uniform sampled vertices, the graph edges are spread with evenly angle with their neighbors (see Figure 3.4(a)). From Figure 3.4(c) we see that in intense contrast the angle is large between adjacent edges on anisotropic meshes. It is more intuitive thinking to set foot on the gaps between adjacent edges. Our scheme intends to ensure that every polar angle of the gaps is less than $\sqrt{\varepsilon}$ by adding pseudo edges in between. Denote three vertices by v_i, v_j, v_k and the two neighbor DGG edges by $\omega(v_i, v_j), \omega(v_i, v_k)$. The gap angle is $\alpha > \sqrt{\varepsilon}$. Then we produce evenly spaced angle partitions by adding $\lfloor \frac{\alpha}{\sqrt{\varepsilon}} \rfloor$ pseudo edges. The length of each pseudo edge is l_ε . The pseudo edge is a geodesic path starting from source vertex v . Pseudo vertex is defined as the destination point of the pseudo edge geodesic path. In this case, denote the destination point by v' . We re-triangle the face where v' is located, since pseudo vertex $v' \notin V$. Finally we invoke COMPUTECANDIDATEEDGE to build the DGG edges incident to all the pseudo vertices. Figure 3.4(c)-(e) illustrate the framework and we show the pseudo code in ADDPSEUDOVERTICESANDEDGES().

Figure 3.4: An illustration for edge pruning and pseudo edges. (a) shows the all the edges from the output of computing candidate edges. The redundant edges are rendered in orange and the edges in green are preserved after edge pruning. (b) demonstrates a typical redundant case $\omega(s, t)$ (rendered in orange). $\omega(s, u)$ and $\omega(u, t)$ approximate the long edge instead (rendered in green). (c)-(d) Inserting the blue pseudo edges of length l_ε . The pseudo edges ensure the accuracy. The angle between any two edges is less than $\sqrt{\varepsilon}$. (e) u is a non-vertex point on the mesh surface. After creating the $\omega(s, u)$ as a pseudo edge, starting from u , we build the red direct paths shorter than l_ε and insert all the new edges into the existing graph.

```

1: function PRUNEREDUNDANTEDGES( $G_\varepsilon, s, \varepsilon$ )
2:    $V_s \leftarrow \emptyset$ 
3:   for all edge  $(s, v) \in E_\varepsilon$  do
4:      $V_s \leftarrow V_s \cup \{v\}$  //  $V_s$  contains  $s$ 's neighboring vertices
5:   end for
6:   // Run a local Dijkstra's algorithm at  $s$ 
7:    $d(s) \leftarrow 0$ 
8:    $\mathcal{Q}.\text{push}(s)$  //  $\mathcal{Q}$  is a priority queue keyed by  $d(v)$ 
9:   while  $\mathcal{Q}$  is not empty do
10:     $u \leftarrow \mathcal{Q}.\text{pop}()$ 
11:    for all edge  $(u, v) \in E_\varepsilon$  and  $v \in V_s$  do
12:      //  $w(u, v)$  is the edge length between  $u$  and  $v$ 
13:      if  $d(u) + w(u, v) \leq (1 + \varepsilon) \times w(s, v)$  then
14:         $d(v) \leftarrow d(u) + w(u, v)$ 
15:        Delete edge  $(u, v)$  from  $E_\varepsilon$ 
16:         $\mathcal{Q}.\text{push}(v)$ 
17:      end if
18:    end for
19:  end while
20: end function

```

3.3.4 Property and Time Complexity

Theorem 4. [62] "*The graph G_ε is a connected metric graph.*"

Proof. After computing candidate edges, all the mesh edges are included because each edge is a direct geodesic path. Although the pruning step remove part of the candidate edges, the mesh edges are kept because they cannot be replaced by others. As a result, as long as M is connected mesh, G_ε is a connected graph. It is not affected by removing $\omega(u, v)$.

Consider we have $s, t, u \in V$ and $\omega(s, t), \omega(t, u), \omega(s, u) \in E_\varepsilon$. If $d_s(s, u) + d_s(u, t) < d_s(s, t)$, then in the pruning step the edge (s, t) would be removed, moreover $d_s(s, u) + d_s(u, t) < d_s(s, t) < (1 + \varepsilon)d_s(s, t)$, such that the graph (V, E_ε) is a metric graph.

Now we analyze the *empirical* time complexity of Algorithm 1.

In function COMPUTECANDIDATEEDGES(M, s, ε), we compute direct geodesic paths incident

to vertex s using our accuracy-aware window propagation strategy, which terminates earlier than the MMP algorithm. Therefore, its time complexity is no worse than that of the MMP algorithm. As shown in Theorem 3, the average degree of the graph G_ε is $O(\frac{1}{\sqrt{\varepsilon}})$, hence, the accuracy-aware window propagation sweeps $O(\frac{1}{\sqrt{\varepsilon}})$ vertices, taking $O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ time. Therefore, it takes $O(\frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$ time to compute all candidate edges.

The function `PRUNEREDUNDANTEDGES`($G_\varepsilon, s, \varepsilon$) runs a Dijkstra-like sweep at vertex s . It is known that for a graph with $|E|$ edges, the priority-queue based Dijkstra's algorithm runs in $O(|E| \log |E|)$ time. Since the number of edges incident to s is $O(\frac{1}{\sqrt{\varepsilon}})$, function `PRUNEREDUNDANTEDGES`($G_\varepsilon, s, \varepsilon$) takes $O(\frac{1}{\sqrt{\varepsilon}} \log \frac{1}{\varepsilon})$ time, implying that the time complexity of edge pruning is $O(\frac{n}{\sqrt{\varepsilon}} \log \frac{1}{\varepsilon})$.

On regular meshes with uniform vertex distribution, the adding pseudo edges is not performed. There is no influence on the complexities.

Overall empirical time for Algorithm 1 is $O(\frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$.

```

1: function ADDPSEUDOVERTICESANDEDGES( $G_\varepsilon, M, s, \varepsilon$ )
2:   Take an arbitrary DGG edge incident to  $s$  as the reference edge and sort the other
   incident edges according to their polar angles to the reference direction. Denote by  $\omega_i$  the
    $i$ -th DGG edge incident to  $s$  and  $\theta_i$  the polar angle,  $i = 1, \dots, k$ .
3:   for  $i = 1; i \leq k; i ++$  do
4:     if  $\theta_{i+1} - \theta_i \geq \sqrt{2\varepsilon}$  then
5:       for  $j = 1; j \leq \lfloor \frac{\theta_{i+1} - \theta_i}{\sqrt{\varepsilon}} \rfloor; j ++$  do
6:         Construct a geodesic path  $\omega(s, p_j)$  with polar angle  $\theta_i + j\sqrt{\varepsilon}$  and length  $l_\varepsilon$ 
7:         Add  $p_j$  to  $M$  as a pseudo vertex and triangulate the face containing  $p_j$ 
8:          $E_{p_j} \leftarrow \text{COMPUTECANDIDATEEDGES}(M \cup \{p_j\}, p_j, \varepsilon)$ 
9:         Add the pseudo vertex  $p_j$  and the pseudo edges  $E_{p_j}$  to  $G_\varepsilon$ 
10:      end for
11:    end if
12:  end for
13: end function

```

3.4 Computing Discrete Geodesics using DGG

The text and results in this section have been published in [62].

3.4.1 DGG-Tailored Label-Correcting Algorithm

Once the DGG G_ε is available, we compute geodesic distances by finding the shortest distances on G_ε . Dijkstra's algorithm is a popular method to solve the shortest path problem with nonnegative arc lengths. It assigns every vertex a tentative distance value: set it zero to the source vertex and infinity for all other vertices. It also marks the source vertex as *current* and all other vertices *unvisited*. Then it performs two steps repeatedly: vertex selection and distance update. The vertex selection step selects a vertex, say v_i , with the smallest distance label for examination. The distance update step scans each unvisited neighbor of v_i and updates its distance label if necessary. Placing vertices in a priority queue keyed by the distances to the source, Dijkstra's algorithm extracts the vertex with the smallest label in an iteration. The overhead for entering and leaving a priority queue is $O(\log |V|)$, hence Dijkstra's algorithm takes $O(|E_\varepsilon| \log |V|) = O(\frac{n}{\sqrt{\varepsilon}} \log n)$ time on G_ε . Since each vertex enters the queue exactly once and its label remains unchanged after leaving the queue, Dijkstra's algorithm is usually referred as a label *setting* method.

To improve the performance of Dijkstra's algorithm, we borrow a label correction scheme [4], which removes the $O(\log n)$ overhead per iteration. We also develop the *DGG-tailored* distance update to further reduce the time complexity to $O(n)$, making it insensitive to the user-specified accuracy parameter ε .

3.4.1.1 Label-Correcting based Vertex Selection

In contrast to the label setting method, the label correcting methods maintain the vertices in a first-in-first-out queue Q and adopt a faster strategy for vertex selection, at the expense of

multiple entrances of vertices in Q . Different label correcting methods are distinguished by the strategy for choosing the queue position to insert a vertex. A popular strategy is *Small Label to the Front* (SLF) [4]: whenever a vertex v_j enters Q , its label d_j is compared with the label d_i of the top vertex v_i of Q . If $d_j \leq d_i$, vertex v_j is inserted at the top of Q ; otherwise at the bottom of Q . Although the SLF scheme lacks a theoretical time complexity, extensive computational results show that it significantly outperforms the label-setting algorithm on real-world graphs [4], since a vertex enters the queue Q only a few times.

3.4.1.2 DGG-Tailored Distance Update

Recall that an approximate geodesic path $\omega(s, t)$ is a union of DGG edges $v_0(= s), v_1, \dots, v_k(= t)$, so that the angle between consecutive edges $v_{i-1}v_i$ and v_iv_{i+1} is no less than $\pi - \sin \sqrt{\varepsilon}$. Based on the above observation, we design the DGG data structure as follows: For every vertex v_i , we store its incident edges $\{e_{ij}\}$ and a set $\{f_{ij}\}$ of fan-shaped regions. Each incident edge $e_{ij} \in E_\varepsilon$ (i.e., an approximate geodesic path ending at v_i) corresponds to a region f_{ij} , where whose boundaries form an angle no less than $\pi - \arcsin \sqrt{\varepsilon}$ to e_{ij} . See Figure 3.5.

In the conventional Dijkstra's algorithm, when a vertex, say v_j , is visited, the algorithm considers *all* of its unvisited neighbors and calculates their distances. The DGG-tailored algorithm improves the performance by considering only *part* of v_j 's unvisited neighbors that provide the shortest distances. Assuming v_j is visited via a DGG edge $e_{ij} = (v_i, v_j)$, the algorithm searches only the neighboring vertices in the fan-shaped region f_{ij} . Intuitively speaking, the conventional Dijkstra's algorithm explores nodes in all directions uniformly, whereas the DGG-tailored label-correcting algorithm considers only the directions that lead to the shortest distances, i.e., the smaller the expected error ε , the narrower the searching range.

With the SLF strategy, it takes $O(1)$ time to enter a vertex to the queue Q and select a vertex from Q . Computational results show that on average, a vertex enters and leaves Q no more than

2 times. See Table 3.2. Assuming that the vertices are uniformly and randomly distributed, the number of DGG edges falling in the fan-shaped region is proportional to the included angle $2\sqrt{\varepsilon}$. Therefore, distance update takes $O(\sqrt{\varepsilon})$ time. Putting it all together, the DGG-tailored label correcting algorithm (Algorithm 3) runs in $O(|E_\varepsilon|\sqrt{\varepsilon}) = O(n)$ empirically.

```
struct DGGVertex{
    int id;
    vector<DGGEdge> edges;
};

struct FanShapedRegion{
    short starting_edge;
    short ending_edge;
};

struct DGGEdge{
    int v2; // the id of the terminal vertex
    double length; // the edge length
    FanShapedRegion fan;
};
```

3.5 Results & Discussions

The text and results in this section have been published in [62].

Since both the DGG and SVG methods are graph-theoretic, they share some common features:

Algorithm 2 DGG-Tailored Label-Correcting Algorithm

Input: DGG $G_\varepsilon = (V, E_\varepsilon)$, source vertex $s \in V$

Output: $d_g(s, v)$ for all $v \in V$

```
1: for all  $v \in V$  do
2:    $d_g(s, v) \leftarrow \infty$ 
3:   Mark  $v$  unvisited
4: end for
5:  $d_g(s, s) = 0$ 
6: Set  $s$  current and place it in a first-in-first-out queue  $\mathcal{Q}$ 
7:  $\bar{d} \leftarrow \frac{\sum_{v \in \mathcal{Q}} d_g(s, v)}{|\mathcal{Q}|}$ 
8: while  $\mathcal{Q}$  is not empty do
9:    $u \leftarrow \mathcal{Q}.\text{top}()$ 
10:  while  $d_g(s, u) > \bar{d}$  do
11:    Move  $u$  to the bottom of  $\mathcal{Q}$ 
12:     $u \leftarrow \mathcal{Q}.\text{top}()$ 
13:  end while
14:   $u \leftarrow \mathcal{Q}.\text{pop}()$ 
15:  Mark  $u$  visited
16:  for all  $v \in [\text{v.fan.starting\_edge}, \text{v.fan.ending\_edge}]$  do
17:    //  $w(u, v)$  is the edge length between  $u$  and  $v$ 
18:    if  $v$  is unvisited and  $d_g(s, v) + w(u, v) < d_g(s, v)$  then
19:       $d_g(s, v) \leftarrow d_g(s, u) + w(u, v)$ 
20:       $b(v) \leftarrow u$ 
21:      if  $v \in \mathcal{Q}$  then
22:        Update  $\sum_{v \in \mathcal{Q}} d_g(s, v)$ 
23:      else
24:        Insert  $v$  to the bottom of  $\mathcal{Q}$ 
25:      end if
26:      Update  $\bar{d}$ 
27:    end if
28:  end for
29: end while
```

Figure 3.5: The label correcting method only allow distance update to occur inside the fan-shape region associated with each DGG edge. (a) shows a vertex p at the center of all the incident DGG edges. In (b), the fan-shape region is displayed along with the two respective edges $\omega(p, q_1)$ and $\omega(p, q_2)$.

Figure 3.6: The DGG-tailored label-correcting has an empirical $O(n)$ time complexity on moderate-scale models. The timing increases flatly with decreasing user-specified accuracy ϵ .

- (i) Both methods solve the discrete geodesic problem by taking advantage of its local feature, therefore, they allow information reuse and can compute geodesic distances efficiently.

(a) T_p with fixed n

(b) T_p with fixed $\varepsilon = 0.1\%$

(c) T_p with fixed $\varepsilon = 0.01\%$

(d) Profiling

Figure 3.7: The DGG construction algorithm runs in $O(\frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$ empirically. (a) shows the construction time T_p with fixed n and varying ε . (b) and (c) show T_p with fixed ε but varying n . (d) Profiling of the DGG construction algorithm. The vertical axis is the ratio $\frac{T_p}{T_p + T_c}$, where T_p is running time of edge pruning and T_c is the time of computing candidate edges. Timing was obtained using single thread on an Intel Xeon E5-2609 v2 2.50GHz CPU.

- (ii) Both graphs are metric graphs, thus, the computed geodesic distances are guaranteed to be true metrics.
- (iii) Both methods can compute either the exact or approximate geodesic distances, depending on the user-specified parameter. For the SVG method, if the user specifies a sufficiently large K (which bounds the maximal vertex degree in the graph), the computed distances are exact. For the DGG method, the result is exact if the desired accuracy ε is sufficiently small.

However, the two methods differ fundamentally in graph construction, space complexity, error control, and runtime performance, which are thoroughly evaluated in Section 3.5.1 to 3.5.4. We also compare DGG with other discrete geodesic algorithms in Section 3.5.5.

The DGG algorithm works on geometric complex models well. The performance of DGG algorithm on typical examples including scanned terrain and architectures are shown in Figure 3.8. Additionally there are also a large amount of models with abundant geometric details tested in the Thingi10K dataset in Section 5.1.

3.5.1 Graph Construction

As analyzed in Section 3.3.4, the DGG construction algorithm runs in $O(\frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$ time empirically, which is also confirmed by experimental results (see Figure 3.6). The algorithm consists of three steps, i.e., computing candidate edges, pruning the redundant edges which can be well approximated by others, and adding pseudo edges if necessary. As Figure 3.7(d) shows, computing candidate edges is dominant. When $\varepsilon = 0.01\%$, edge pruning takes roughly half of the time as of computing candidate edges. For even higher accuracy $\varepsilon < 0.001\%$, most of the candidate edges become DGG edges, hence the ratio $\frac{T_p}{T_p + T_c}$ drops.

In contrast, SVGs can be constructed in a straightforward manner. For each vertex, one simply computes the direct geodesic paths within a geodesic disk of K vertices, which takes

Figure 3.8: Geodesics on models with sharp geometry. The accuracy control for DGG is $\varepsilon = 1\%$. The accuracy is not affected by geometric sharp features but the quality of mesh tessellation.

$O(K^2 \log K)$ time. Table 5.4 shows the runtime performance of the two construction algorithms. We observe that for the accuracy parameter $\varepsilon > 0.5\%$, the computational cost of constructing DGG is roughly the same as that of SVG, mainly because for relatively large parameter ε , the average degree is small, so is the number of redundant edges. So it takes little time to identify redundant edges. However, for very small ε (e.g., $< 0.01\%$), constructing DGG is more expensive than that of SVG, since DGG computes more candidate edges than SVG in order to ensure the user-specified accuracy. As Table 3.1 shows, more than 80% of the candidate edges are redundant. Identifying those edges also takes time.

Table 3.1: Statistics of edge pruning.

Model ($ V $)	$\varepsilon = 0.1\%$				$\varepsilon = 0.01\%$			
	Before pruning		After pruning		Before pruning		After pruning	
	$\bar{\varepsilon}$	D	$\bar{\varepsilon}$	D	$\bar{\varepsilon}$	D	$\bar{\varepsilon}$	D
Sphere (20K)	0.0067%	300.7	0.027%	83.2	0.00016%	2305.9	0.0028%	269.7
Fertility (30K)	0.0074%	298.1	0.029%	83.3	0.00013%	2066.6	0.0030%	266.7
Bunny (72K)	0.020%	229.8	0.036%	80.3	0.0045 %	1113.8	0.0040%	259.6
Armadillo (173K)	0.024%	180.8	0.042%	81.2	0.00096%	948.3	0.0033%	266.1
Lucy (263K)	0.054%	129.6	0.063%	67.9	0.0047 %	586.9	0.0054%	237.9
Buddha (500K)	0.021%	224.1	0.035%	79.8	0.0012 %	1662.6	0.0047%	261.4
Isidore Horse (1104K)	0.019%	234.3	0.031%	75.8	0.0011%	1564.2	0.0032%	261.2

3.5.2 Space Complexity

SVG partitions long geodesic paths using saddle vertices, thus, the size of an SVG is highly sensitive to the number of saddle vertices and their distributions. We observe that when the number of saddle vertices increases, the direct geodesic paths tend to be shorter and the vertex degree drops, hereby the runtime performance improves. Conversely, if a mesh has few saddle vertices, the vertex degree of SVG could be very high. In contrast, DGG approximates long geodesic paths using any kind of relay vertices, hence its size is not sensitive to the number of saddle vertices. See Figure 3.11.

Figure 3.9: The average degree of DGG as well as SVG converges to an upper limit when the error is small enough or K is large enough.

As shown in Section 3.3.4, the average degree of a DGG is $O(\frac{1}{\sqrt{\varepsilon}})$. It is worth noting that the upper bound is *not* tight. Given a sufficiently small ε , COMPUTECANDIDATEEDGES() returns *all* direct geodesic paths and none of them can be removed in the edge pruning stage.

As a result, the resulting graph is exact and the average degree is bounded by some model-dependent constant. See Figure 3.9.

Figure 3.10: Relation between the graph size or average degree of SVG and DGG and the actual relative mean error. DGG is a more compact graph than SVG under almost all error scenarios. With fixed average degrees, we observe a significant advantage of DGG that the accuracy is 2-5 times better than SVG.

Figure 3.12 compares SVG and DGG in terms of vertex degrees. The standard deviation of DGG's degree is 60% smaller than that of SVG, meaning that DGG is more balanced than SVG.

Figure 3.10 compares SVG and DGG in terms of graph size and quality. Take the 350K-vertex Gargoyle model as an example. Given a DGG and an SVG with similar sizes, the geodesic distances computed by DGG is nearly twice as accurate as those by SVG. Conversely, given the graphs producing geodesic distances of similar accuracy, the DGG is only $\frac{2}{3}$ the size of the SVG. Table 3.1 reports the statistics of edge pruning. We observe that a large amount of candidate edges are redundant, which also justifies the compactness of DGG.

Figure 3.11: The average degree of SVG is significantly affected by noise, since noise produces saddle vertices. In contrast, DGG is more robust to noise in terms of graph size.

		DGG		SVG	
		$\varepsilon = 0.1\%$	$\varepsilon = 0.01\%$	$K = 135$	$K = 500$
		$\bar{\varepsilon} = 0.027\%$	$\bar{\varepsilon} = 0.0028\%$	$\bar{\varepsilon} = 0.027\%$	$\bar{\varepsilon} = 0.0028\%$
$\bar{D}$	saddle	85.5	266.7	125.1	425.8
	non-saddle	85.3	265.8	127.4	438.3
	all	85.4	266.3	126.2	431.8
σ	saddle	4.29	14.17	4.37	34.93
	non-saddle	4.37	14.73	4.38	43.37
	all	4.33	14.46	4.52	39.73

Figure 3.12: Different type of vertex and the degrees statistics on the Bimba model with 74.5K vertices. DGG has smaller vertex degree variance than SVG.

Model	$ V $	τ	DGG ($\varepsilon = 10^{-2}$)						DGG ($\varepsilon = 10^{-3}$)						DGG ($\varepsilon = 10^{-4}$)						The Heat Method					
			w/o			w/			w/o			w/			w/o			w/			original mesh			Delaunay mesh		
			ε	T_g (s)	$\frac{ V_p }{ V }$	ε	T_g (s)	$\frac{ V_p }{ V }$	ε	T_g (s)	$\frac{ V_p }{ V }$	ε	T_g (s)	$\frac{ V_p }{ V }$	ε	T_g (s)	$\frac{ V_p }{ V }$	ε	T_g (s)	$\frac{ V_p }{ V }$	ε	T_g (s)	$\frac{ V_p }{ V }$	ε	T_g (s)	$\frac{ V_p }{ V }$

Figure 3.13: The adding pseudo edge scheme is effective on improving the overall accuracy. τ measures the anisotropy degree for mesh triangulation. “w/” means with pseudo primitives and “w/o” means without those primitives. T_g is for distance querying time. $\frac{|V_p|}{|V|}$ is pseudo vertex ratio.

3.5.3 Runtime Performance

Table 5.4 shows the statistics of mesh complexity and runtime performance. SVG adopts Dijkstra’s algorithm to compute geodesic distances in $O(Kn \log n)$ time, where K is the maximal degree. For small- and middle-scale meshes, we observe that the DGG-tailored label-correcting algorithm empirically runs in $O(n)$ time, which is insensitive to the user-specified accuracy parameter ε (see Figure 3.6). For large-scale meshes, the runtime performance of DGG slightly depends on ε . Take the 1M-vertex Blade model (Figure 1) as an example. It takes the DGG with $\varepsilon = 0.36\%$ 0.6s to compute a single-source geodesic distance, while the DGG with $\varepsilon = 0.0062\%$, which is two orders of magnitude more accurate, takes only 1.8s.

To make a fair comparison between SVG and DGG, we construct the graphs with similar accuracy. For common models with up to 3 million vertices, we observe that DGG runs 2 to 5 times faster than SVG. The improvement can be even higher for larger models, since DGG is more compact than SVG.

Table 3.2: The FIFO-queue statistics. Numbers are the account of elements entering the queue.

Model	$ V $	$\varepsilon = 0.1\%$	$\varepsilon = 0.01\%$	$\varepsilon = 0.001\%$
Bunny	72K	1.31	1.57	1.73
Armadillo	173K	1.27	1.51	1.72
Lucy	263K	1.27	1.46	1.63
Gargoyle	350K	1.29	1.47	1.79
Buddha	500K	1.34	1.54	1.79
Thai Statute	625K	1.39	1.58	1.77
Ramesses	825K	1.4	1.54	1.69
Isidore Horse	1.1M	1.26	1.49	1.73
Blade	2M	1.23	1.41	1.71
Dragon	3M	1.28	1.47	out of memory

3.5.4 Approximation Error

With SVG, users control the accuracy via specifying the maximal degree of the graph, which is non-intuitive. As mentioned above, direct error control is one of the unique features of DGG. Given a desired tolerance $\varepsilon > 0$, Theorem 3 guarantees that the DGG G_ε approximates geodesic distances with a relative mean error $O(\varepsilon)$. Through extensive evaluations on both isotropic and anisotropic meshes, we observe that the actual error $\bar{\varepsilon}$ never exceeds the user-specified tolerance ε . See Figure 3.15.

Figure 3.13 shows the results on anisotropic meshes. The Ellipsoid, Cyclide and Fertility models have very high degree of anisotropy with $\tau > 5$, implying that the DGG edges are distributed highly non-uniformly. After adding pseudo vertices and edges, we can ensure that for an arbitrary vertex $v \in V$, its incident edges can cover *all* radial directions, i.e., any two adjacent edges have an included angle no more than $\arcsin \sqrt{\varepsilon}$. As shown in the table, the accuracy of the graph is improved significantly, demonstrating the efficacy of procedure ADDPSEUDOVERTICESANDEDGES on highly anisotropic meshes. We also compare with the HM-DM method, which adopts Delaunay meshes [37] as preconditioner to improve robustness and accuracy. Delaunay meshes also add pseudo vertices to ensure that the local Delaunay condition holds everywhere. As Figure 3.13 shows, the number of pseudo vertices added by DM is com-

parable to ours. However, our method consistently outperforms both HM and HM-DM in terms of accuracy and performance.

It is also worth noting that for the Eight model with anisotropy measure $\tau = 1.38$, pseudo vertices are not necessary. Even without them, the graphs can produce geodesic distances satisfying the user-specified tolerances.

3.5.5 Comparison with Other Methods

Table 3.3: Typical discrete geodesic algorithms comparison. We assume that the comparison model has n vertices.

Method	Polyhedron	Approximation Ratio	Pre-Computing	Computing SSAD	Space complexity
Agarwal et al. [1]	convex	$1 + \varepsilon$	$O(n^2 \log n)$	$O(\frac{n}{\varepsilon} \log \frac{1}{\varepsilon} \log \frac{n}{\varepsilon})$	N.A.
Schreiber and Sharir [49]	convex	1	-	$O(n \log n)$	$O(n \log n)$
MMP [41]	non-convex	1	-	$O(n^2 \log n)$	$O(n^2)$
FWP-MMP [67]	non-convex	1	-	$O(n^2)$	$O(n^2)$
CH [8]	non-convex	1	-	$O(n^2)$	$O(n)$
ICH [65]	non-convex	1	-	$O(n^2 \log n)$	$O(n)$
Varadarajan and Agarwal [59]	non-convex	$7(1 + \varepsilon)$	-	$O(n^{\frac{5}{3}} \log^{\frac{5}{3}} n)$	N.A.
Varadarajan and Agarwal [59]	non-convex	$15(1 + \varepsilon)$	-	$O(n^{\frac{8}{5}} \log^{\frac{8}{5}} n)$	N.A.
FMM [50]	non-convex	N.A.	-	$O(n \log n)$	$O(n)$
The heat method [11]	non-convex	N.A.	factoring Laplacian matrix	empirical $O(n)$	empirical $O(n)$
GTU [64]	non-convex	N.A.	$O(mn^2 \log n)$	$O(n)$	$O(m^2 + n)$
AMMP [55]	non-convex	N.A.	-	empirical $O(n \log n)$	empirical $O(n)$
SVG [55]	non-convex	N.A.	$O(nK^2 \log K)$	$O(Kn \log n)$	$O(Kn)$
DGG	non-convex	theoretical $1 + O(\varepsilon)$ empirical $1 + \varepsilon$	empirical $O(\frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$	empirical $O(n)$	$O(\frac{n}{\sqrt{\varepsilon}})$

Table 3.3 summarizes the features of the existing discrete geodesic algorithms. Although DGG aims at computing approximate geodesic distances with bounded error, it is closely related to *exact* algorithms, by which we can compute the direct geodesic paths. Therefore, development of exact algorithms can improve the performance of DGG construction.

Surazhsky et al. [55] extended the MMP algorithm with a window merging operation: before propagating a window, it tries to merge it with adjacent windows on the same edge. They proved that the merging operation is has a *local* error bound when some conditions are met.

Table 3.4: Mesh complexity and runtime performance. We select K for SVG in order to compare with DGG under close relative mean error (represented by $\bar{\epsilon}$) for fair comparison. We name T_p for the preprocessing time and T_g for a single distance query time. Timings are measured in seconds. The experiments are conducted on an Intel Xeon 2609 v2 2.50 GHz CPU using a single thread with 32GB memory. τ is the anisotropy degree. $\bar{D}$ measures the average degree for SVG and DGG.

Model	V	τ	FWP-MMP	HM-DM			DGG ($\epsilon = 10^{-2}$)				SVG			
			T	T_p	T_g	$\bar{\epsilon}$	D	T_p	T_g	$\bar{\epsilon}$	K	D	T_p	T_g
Fertility	30K	1.27	1.1	1.09	0.034	0.90%	21.8	5.6	0.015	0.43%	34	32.7	6.3	0.021
Bunny	72K	1.26	2.58	4.78	0.10	0.57%	25.3	20.4	0.034	0.32%	32	30.6	15.6	0.046
Armadillo	173K	1.35	5.16	10.53	0.26	0.49%	26.0	24.5	0.091	0.36%	34	31.2	43.1	0.13
Lucy	263K	1.60	7.9	14.31	0.37	1.31%	22.2	45.5	0.12	0.48%	31	29.4	52.4	0.15
Gargoyle	350K	1.42	12.3	33.54	0.53	0.81%	22.5	65.6	0.17	0.43%	35	32.8	82.4	0.23
Buddha	500K	1.31	29.4	58.3	0.98	0.89%	21.2	130.5	0.23	0.45%	32	30.0	120.7	0.35
Thai Statute	625K	1.41	30.1	109.0	1.26	1.03%	23.0	137.6	0.53	0.38%	30	28.7	116.8	0.73
Ramesses	825K	1.16	23.5	325.8	1.85	0.69%	20.9	158.0	0.54	0.44%	29	27.4	152.8	0.89
Blade	1M	1.32	65.2	265.3	1.91	2.1%	19.5	204.6	0.66	0.36%	35	33.9	232.1	1.2
Isidore Horse	1.1M	1.19	96.3	272.6	1.90	1.14%	20.5	176.3	0.59	0.39%	31	29.5	223.5	1.56
Blade	2M	1.41	168.2	900.5	3.21	1.98%	19.5	347.2	1.15	0.59%	27	26.3	316.3	1.78
Dragon	3M	1.38	200.3	out of memory			16.5	498.5	1.11	0.74%	31	29.4	498.3	2.76

Model	V	τ	DGG ($\epsilon = 10^{-3}$)				SVG				DGG ($\epsilon = 10^{-4}$)				SVG			
			$\bar{D}$	T_p	T_g	$\bar{\epsilon}$	K	$\bar{D}$	T_p	T_g	$\bar{D}$	T_p	T_g	$\bar{\epsilon}$	K	$\bar{D}$	T_p	T_g
Fertility	30K	1.27	81.9	70.9	0.022	0.030%	133	127.5	34.1	0.07	267.1	2020.7	0.027	0.0029%	492	437.6	204.5	0.21
Bunny	72K	1.26	76.3	111.8	0.045	0.048%	117	111.2	68.4	0.14	255.6	1584.7	0.055	0.0054%	413	363.7	378.1	0.42
Armadillo	173K	1.35	75.1	199.7	0.14	0.049%	123	112.0	190.4	0.44	264.1	3315.2	0.20	0.0035%	518	409.9	1314.8	1.30
Lucy	263K	1.60	67.9	176.9	0.18	0.063%	102	93.8	211.4	0.48	228.9	2023.2	0.22	0.0067%	477	342.8	1608.7	1.52
Gargoyle	350K	1.42	83.0	864.5	0.28	0.031%	132	119.7	396.5	0.81	261.5	12028.1	0.40	0.0031%	482	373.4	2330.1	2.25
Buddha	500K	1.31	79.8	992.5	0.44	0.035%	130	110.5	516.2	0.83	261.4	37288.1	0.72	0.0047%	386	267.4	2338.5	2.35
Thai Statute	625K	1.41	83.5	1661.8	0.66	0.028%	132	123.2	699.0	1.03	266.7	43495.2	0.85	0.0029%	450	428.8	2865.2	2.97
Ramesses	825K	1.16	82.6	2254.2	1.02	0.026%	123	121.3	1063.1	1.67	263.9	48514.3	1.16	0.0028%	458	447.3	3745.3	3.73
Blade	1M	1.32	75.1	2156.3	1.2	0.049%	100	98.3	801.2	2.0	251.7	33622.3	1.8	0.0062%	392	390.5	9756.2	5.5
Isidore Horse	1.1M	1.19	75.8	3648.5	0.95	0.031%	131	128.3	1957.2	1.93	261.2	35422.5	1.22	0.0032%	423	415.1	4265.2	5.02
Blade	2M	1.41	72.7	10940.2	1.84	0.057%	84	81.8	3214.2	2.93	253.6	34056.2	2.56	0.0059%	358	353.5	8745.3	9.72
Dragon	3M	1.38	61.8	2466.9	2.39	0.071%	87	85.3	6754.6	5.78	228.9	28571.4	out of memory		350	346.2	13423.2	o.f.m.

However, their algorithm does not allow users to directly specify the *global* accuracy. Moreover, its runtime performance is not comparable to the pre-computation methods, such as the heat method, SVG and DGG, which do take advantage of information reuse.

The fast marching method and the heat method are popular approximate methods that work for a wide range of geometric domains, such as triangle meshes, point clouds, implicit functions and tetrahedral meshes. Both methods are conceptually simple and easy to implement, and they are much more efficient than the exact methods. However, FMM and HM do not allow error control and the results are sensitive to mesh triangulation and resolution. The computed

distances are not metrics, due to violation of triangle inequality and the symmetric condition.

I also observe that FMM and HM work fairly well on *smooth* models. However, they produce poor results on models with rich geometric details, even the models have highly regular triangulations. See Figure 3.16. In contrast, SVG and DGG do take advantage of local geometric features. The richer the geometric details, the more number of saddle vertices, the shorter the direct geodesic paths, hereby the stronger the local structure of the discrete geodesic problem and the better performance and accuracy SVG and DGG can provide.

3.5.6 Applications

With a simple modification to DGG's data structure (Figure 3.17), the geodesic path between arbitrary points can be traced during each window propagation procedure. As DGG consists of segments of direct polyhedral geodesic path, recording the geometric information along the unfolded face sequence results in an exact representation in 2D coordinate system. Thus as long as an in-between angle from the mesh edge of the current window to the 2D unfolded geodesic path is recorded, it is then straightforward to trace back to the original source point. Given this feature as well as the fast querying, without loss of accuracy an approximate geodesic path is traced. One direct application is path planning for 3D scenarios, e.g. a terrain map. See Figure 3.18.

CyberTape and rope caging. Wang and Kwok et al. [60] [31] proposed an efficient method to simulate the behavior of wrapping tapes or ropes from any given point on 3D surface around nearby convex part. The methods seek approximate geodesic loops on the surface and can be achieved using the DGG scheme. With the precomputed DGG, an approximate method for finding the geodesic loops can adopt the same Dijkstra-like algorithm despite not for finding open paths. For example, a shortest path problem satisfy the following conditions. Firstly, the distance should not be zero. Secondly, for any path around middle vertex the 2D angle on an

unfolded plane should be approximately 180 degrees. Furthermore, bi-directional Dijkstra is efficient for this method. See Figure 3.19 for illustrations on a Fertility model.

Extracting cross-parameterization. Kwok et al. [32] described a robust method for cross-parameterization on polyhedral surfaces. Their method adopts a novel signature named Length-Preserved Base Domain (LPBD). The new signature maintains the distance measure similar on the paired models and balances runtime performance well. With the help of geodesic distance, either the iterations of the optimization scheme is lessened or a better distortion rate is achieved. The rapid path querying and tracing feature of DGG enables further enhancement in terms of the speed and anti-distortion for their method. An example with low distortion bijective patch is shown in Figure 3.20.

3.6 Summary

The text in this section have been published in [62].

I can show that the shortest path distances on DGG approximate geodesic distances with a *theoretical* approximation ratio $1 + O(\varepsilon)$. Extensive results show that the actual approximation ratio does not exceed $1 + \varepsilon$. Thus, it is of particular (theoretical) interest to refine the bound. On the practical side, there is a large room for improving the runtime performance of the DGG construction algorithm. As Table 3.1 shows, for ε (e.g., $< 0.1\%$), more than 80% of candidate edges are redundant, implying that our current strategy for accuracy control in window propagation is very pessimistic. Therefore, it is highly desired to develop a more effective method that could identify redundant edges at early stage, hereby reducing the construction time.

Figure 3.14: Runtime performance and error distribution. (a) shows the wavefronts resulted from FWP-MMP. From (b) to (f) is the error map of different methods rendered with colors. The precision of the heat method is improved by Delaunay meshes (DM). We adopt varied scales in the horizontal axes of error histograms for clearer visualization.

Figure 3.15: The experimental error $\bar{\varepsilon}$ and the user-specified tolerance ε . The empirical ratio $\bar{\varepsilon}/\varepsilon$ on the two representative models is between 0.3 and 0.6.

Figure 3.16: Differential equation methods show unstable performance in terms of accuracy with different level of geometric details.

```

struct DGGEEdge{
int v2; // the id of the terminal vertex
double length; // the edge length
FanShapedRegion fan;

IntersectionInfo intersect; // the window information
};

struct IntersectionInfo{
int eid; // window edge id
double corner_angle; // CW angle between path and mesh edge
};

```

Figure 3.17: The modified data structure for polyhedral path tracing.

Figure 3.18: Geodesics on a terrain model. Each path represented by gray tubes is calculated within 0.5 seconds. The model has 66,049 vertices.

Figure 3.19: Approximate geodesic loops simulate the tape and rope. A heuristic controls the smoothness of the ring when two shortest path meets at the same vertex.

Figure 3.20: The bijective boundaries on a hand-shaped model. The yellow dots represent the traced intersection point on mesh edges based on the modified data structure featured intersection information. The boundaries are then computed by connecting neighbor intersection points according to the unfolding sequence.

Chapter 4

Computing Discrete Green's Functions using DGG

4.1 Motivation

Although we can find the underlying connection between random-access Green's functions and geodesic distance metric on compact 2-Manifold from Green's third identity, there still remains challenge in terms of computation. On a closed triangle mesh $M = (V, E, F)$, Green's third identity has the following form

$$\begin{aligned}\phi_s(x) &= c + \sum_y g(x, y) \triangle \phi_s(y) A \\ &= c + \sum_y g(x, y) A^{-1} L_c \phi_s(y) A \\ &= c + \sum_y g(x, y) L_c \phi_s(y)\end{aligned}\tag{Eq. 4.1}$$

However, given $\phi_s(x)$ for all vertices, if we directly compute every $g(x, y)$ corresponding to all pairs of vertices, the cost of such computation is too big. As such, we need to find an approach to reduce the spatial complexity. In this chapter, we explain in detail the localization approach based on the local feature of harmonic B-spline. Figure 4.1 illustrates two harmonic basis functions. The left to right are figures showing one-ring neighbors of two knots with

Figure 4.1: The localization is based on harmonic B-spline. Harmonic B-spline relies on Green's functions. The figures showing the discrete Green's function on a triangle patch are illustrated by [15].

Laplacians, figures showing the discrete Green's function taking the black dot as center, and figures showing corresponding harmonic basis functions. The basis functions equal to the Laplacian values multiplied by the Green's functions.

4.2 Preliminaries

4.2.1 Discrete Green's Third Identity

Green's third identity in continuous manner is defined as

$$\begin{aligned} \phi_s(x) &= \iint_{\Omega} g(x, y) \Delta \phi_s(y) \, d\sigma \\ &+ \oint_{\partial\Omega} \left(\phi_s(y) \frac{\partial g(x, y)}{\partial n} - g(x, y) \frac{\partial \phi_s(y)}{\partial n} \right) \end{aligned} \quad (\text{Eq. 4.2})$$

On a compact smooth 2-manifold $\mathcal{M}$, we ignore the latter part

$$\phi_s(x) = \iint_{\Omega} g(x, y) \Delta \phi_s(y) \, d\sigma \quad (\text{Eq. 4.3})$$

$g(x, y), G$	Green's functions defined on (x,y)
$\phi_s(x), \Phi$	Geodesic distance from source s to point x
c, C	constant average function value
L_c, L	conformal laplacian matrix
$\mathcal{M}$	compact smooth 2-manifold w/o boundary
$M = (V, E, F)$	closed mesh with vertices V , edges E , faces F
V_i, i	vertex i
$\lambda_i^s, \lambda_i, \Lambda$	control points defined as geodesic distances from source s to vertex i
$\psi_i(x)$	basis function defined on vertex i
a_i, A_{VM}	constant average area
n_{ij}, N_L	element at row i column j in a laplacian matrix
N_V	total number of vertices(global)
$\mathcal{N}(x)$	user-defined neighbors of point x (local region) v_i
$\mathcal{N}_{tar}(x)$	$\mathcal{N}(x)$ plus one out-bound-ring
N_n	number of elements in $\mathcal{N}(x)$
N_t	number of elements in $\mathcal{N}_{tar}(x)$
$\mathcal{VT}(x)$	nearest 3 vertices of point x
$\omega_{x\epsilon}, \Omega$	barycentric weight of vertex ϵ

Table 4.1: Nomenclature

Discretizing it on a closed triangle mesh $M = (V, E, F)$

$$\begin{aligned}
\phi_s(x) &= c + \sum_y g(x, y) \triangle \phi_s(y) A \\
&= c + \sum_y g(x, y) A^{-1} L_c \phi_s(y) A \\
&= c + \sum_y g(x, y) L_c \phi_s(y)
\end{aligned} \tag{Eq. 4.4}$$

Or in the matrix form

$$\Phi^T L_c^T G = \phi_s(x) - c \tag{Eq. 4.5}$$

where x is an arbitrary point on $\mathcal{M}$, V_y are vertices of the closed mesh, and c is a constant number representing an average value of $\phi_s(x)$.

From a single source, $\Phi^T L_c^T$ is a dense row vector, and G is a column vector, and $\phi_s(x) - c$ is a single number.

Consider we directly solve G based on the above equations. We need as many sources s as y .

It will be the number of vertices to get all the pairs (x, y) .

Using multiple sources s , $\Phi^T L_c^T$ becomes a dense matrix, and it is infeasible for a large scale mesh.

4.2.2 Harmonic B-spline

Now we let the geodesic distance between vertex s to arbitrary point x be the harmonic B-spline function $\phi_s : \mathcal{M} \rightarrow \mathbb{R}^d$, which has the form

$$\phi_s(x) = \sum_{i=1}^m \lambda_i \psi_i(x), \quad (\text{Eq. 4.6})$$

where $\lambda_i \in \mathbb{R}^d$ is the control point, and m is the number of control points.

The basis function for Vertices $V_i \in V$ is defined as

$$\psi_i(x) = a_i + \sum_j n_{ij} g(x, j), \quad (\text{Eq. 4.7})$$

where $\{V_j\}$ are one-ring neighbors of V_i , and $n_{ij} = L_c(i, j)$.

If we let the left hand side of Green's third identity be the geodesics as well, from a global point of view, we have two equivalent form to calculate $\phi_s(x)$

$$\begin{aligned} \sum_{i=1}^m \lambda_i \sum_j n_{ij} g(x, j) &= \sum_y g(x, y) L_c \phi_s(y) \\ \sum_{i=1}^m \lambda_i a_i &= c \end{aligned} \quad (\text{Eq. 4.8})$$

where considering all the vertices, we have

$$j, y \in V, \quad x \in \mathcal{M}, \quad s \in \mathcal{S} \subset V$$

In matrix form, the first equation can be written as

$$\Lambda^T N_L G = G^T L_c \Phi \quad (\text{Eq. 4.9})$$

and as the definition of N_L

$$N_L = L_c \quad (\text{Eq. 4.10})$$

we can derive from the former two equations that

$$\Lambda = \Phi \quad (\text{Eq. 4.11})$$

4.3 Localized Solver for Discrete Green's Functions

The text and results of the presented work in sub-section 4.3.1 have been published in [62].

4.3.1 Construction of Discrete Geodesic Distance Fields Between Arbitrary Points

The discrete geodesic graphs presented in the previous chapter can register geodesic distances between any combination of vertices. It is exceedingly desired to calculate the geodesic distances and obtain the paths between non-vertex points for several applications such as sampling. Denote two non-vertex points by $p_1, p_2 \in M \setminus V$ located in faces $f_1, f_2 \in F$. As among vertices of f_1 and f_2 the geodesic distances are accessible, it is applicable to adopt bilinear interpolation to inexact $d_g(p_1, p_2)$. Nevertheless an import property of geodesic distances is non-linear. Also bilinear interpolation frequently creates poor outcomes with low triangulation quality. The GTU method [64] creates preferred outcomes over bilinear interpolation, be that as it may, it processes just a first-order estimate of $d_g(p_1, p_2)$.

I present a viable procedure based on the previous discussion and analysis for computing approximate geodesic distances between non-vertex points. Following the description in the previous chapter, we insert pseudo vertex p_i , split the containing surface and build the incident geodesic paths of p_i . The augmented graph remains $1 + O(\varepsilon)$ approximation to the geodesics

because newly added edges are direct geodesic paths. Figure 4.2 delineates the idea and Algorithm 3 provides with pseudo-code implementation.

To enhance the execution, we embrace the bidirectional Dijkstra's calculation, which two concurrent Dijkstra looks on the enlarged graph - one forward from p_1 and one in reverse from p_2 ceasing when the two compromise. Since locally refreshing the graph at p_1 and p_2 takes $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$ time, the method `BIDIRECTIONALDIJKSTRA()`, taking $O(n)$ time, is the overwhelming term regarding time multifaceted nature. In this way, registering the expanded edges does not build the general time multifaceted nature. See Figure 4.2 for an outline and Algorithm 3 for the pseudo-code.

Figure 4.2: DGG is capable of computing geodesic distance between arbitrary points. (a) p is located in face $f \in F$. $p \in M \setminus V$ is an arbitrary non-vertex point. (b) The blue curves represent the new DGG edges emanating from p after re-triangulate f . (c) We compute the with good accuracy source p from the augmented graph. The results is comparable to the exact distance in (d).

Using the augmented DGG, Figure 4.3 shows the resulting geodesic path between arbitrary non-vertex points.

4.3.2 Localized Harmonic B-splines

Since global computation of $g(x, y), x \in \mathcal{M}, y \in V$ is infeasible, we may further consider a local region where green's function has most influence over the whole mesh.

$$\phi_s(x) = \sum_{i=1}^m \lambda_i^s \psi_i(x) = \sum_{i=1}^m \lambda_i^s \sum_j n_{ij} g(x, j) + a_i \quad (\text{Eq. 4.12})$$

Algorithm 3 Computing Geodesic Distance between Arbitrary Points

Input: A triangle mesh $M = (V, E, F)$ and its DGG $G_\varepsilon = (V, E_\varepsilon)$, the source and destination points $p_1, p_2 \in M$, the desired accuracy ε

Output: The geodesic distance $d_g(p_1, p_2)$

```
1:  $\tilde{E}_\varepsilon \leftarrow E_\varepsilon$ 
2: for  $i=1$  to 2 do
3:   if  $p_i \notin V$  then
4:      $M' \leftarrow M \cup \{p_i\}$  // Add  $p_i$  as a temporary vertex
5:      $\tilde{E}_\varepsilon \leftarrow \tilde{E}_\varepsilon \cup \text{COMPUTECANDIDATEEDGES}(M', p_i, \varepsilon)$ 
6:   end if
7: end for
8:  $\tilde{G} \leftarrow (V \cup \{p_1, p_2\}, \tilde{E}_\varepsilon)$ 
9:  $d_g(p_1, p_2) \leftarrow \text{BIDIRECTIONALDIJKSTRA}(\tilde{G}, p_1, p_2)$ 
```

Figure 4.3: The DGG result between arbitrary non-vertex points.

Figure 4.4: Farthest point sampling (FPS) on a model with $|V| = 2.5K$. The FPS computes a new sample using the distance between existing ones. The feature of FMM and HM only allows them to compute distances between vertices. Given bad triangulation, the distance is not accurate and produces low quality sampling results. Red points indicate the final samples while the blue points are filtered out by FMM and HM. Our DGG with augmented graph produces good quality sampling compared to vertex-only methods FMM and HM.

We specify that

$$i \in \mathcal{N}(x), j \in \mathcal{N}_{tar}(x)$$

$\mathcal{N}(x)$ is a set of local neighbor vertices around x . $\mathcal{N}_{tar}(x)$ is the union of $\mathcal{N}(x)$ and the one-ring neighbors of $\mathcal{N}(x)$'s boundary vertices. Also we call $\mathcal{N}_{tar}(x)$ a 'target' set, since it is composed of all the vertex indices of local solution.

Note that in local manner, $|S| \geq |\mathcal{N}_{tar}(x)|$ ensures there are enough constraints, but S and $\mathcal{N}_{tar}(x)$ can be different sets.

Since $\phi_s(x)$ is the sum of , if we only consider part of the vertices, we need to guarantee the sum is not changed in order to make the equation(8) hold. Now we prove the omitted vertices $V - \mathcal{N}(x)$ have no influence to the calculation of $\phi_s(x)$.

First consider $x \in V$.

On a fixed vertex with index x , we have

$$\psi_i(x) = a_i + \sum_j n_{ij} g(x, j) \quad (\text{Eq. 4.13})$$

In Yau's and Lipman's theory, discrete green's function has the following property

$$L_i G = [0..1..0] - [\frac{1}{N}.. \frac{1}{N}] \quad (\text{Eq. 4.14})$$

where N is the size of the row. L_i is the i_{th} $N \times 1$ row of the Laplacian matrix L_c , G is a $N \times N$ matrix containing all-pair green's function for V . 1 is the i_{th} element of the first $N \times 1$ row vector, with 0 on the other positions, which means

$$\begin{aligned} \frac{1}{N} + \sum_j L_{ij} g(j, i) &= 1 \\ \frac{1}{N} + \sum_j L_{ij} g(j, k) &= 0, \quad k \neq i \end{aligned}$$

From Eq. 4.10, we know that $N = L_c$ or $n_{ij} = L_c(i, j)$ for all the vertices in V , and let $a_i = \frac{1}{N}$, as empirically if a_i is assigned to other average area value the result shows little deviation. And we have

$$\begin{aligned}\psi_i(x) &= a_i + \sum_j n_{ij}g(x, j) \\ &= a_i + \sum_j n_{ij}g(j, x) \\ &= \frac{1}{N} + \sum_j L_{ij}g(j, x)\end{aligned}$$

which indicates that

$$\psi_i(x) = \begin{cases} 1, & \text{if } i = x \\ 0, & \text{otherwise} \end{cases} \quad (\text{Eq. 4.15})$$

In order to get the same green's function value as in global case, we still need to keep the form Eq. 4.4 and Eq. 4.6 equivalent to represent $\phi_s(x)$, but the above procedure proves that we can simply remove any row of λ_i^s as long as $i \neq x$ in that the corresponding $\psi_i(x) = 0$. To keep the same form with Eq. 4.6 and Eq. 4.7, we let

$$\lambda_i = \phi_s(i) \text{ or } \Lambda = \Phi$$

but now the domain has changed to $i, y \in \mathcal{N}(x)$, $s \in \mathcal{S}$ and we can solve

$$g(x, j), j \in \mathcal{N}_{tar}(x), x \in V$$

Then we consider $x \in \mathcal{M} - V$.

Empirically for arbitrary point $x \in \mathcal{M}$, the basis function has such property

$$\begin{cases} \psi_i(x) > 0, & \text{if } i \in \mathcal{VT}(x) \\ \psi_i(x) = 0, & \text{otherwise} \end{cases} \quad (\text{Eq. 4.16})$$

where $\mathcal{VT}(x)$ is the vertex index set of the face in which the point x is located.

Since the basis function is monotone and smooth, if i is anywhere other than among $\mathcal{VT}(x)$, $\psi_i(x) = 0$.

As long as we remove rows $\notin \mathcal{VT}(x)$, we can solve

$$g(x, j), j \in \mathcal{N}_{tar}(x), x \in \mathcal{M} - V$$

However, it should be noted that the local area contains a set of boundary vertices which have no boundary values, making the rank of the matrix on the right-hand side lower than the number of variables. So we cannot solve the linear system directly.

4.3.3 Optimization

Since we cannot solve the dense linear system in the previous section, we adopt an optimization approach to achieve optimal result. Inspired by Lipman's discretization of Green's function associated with bi-laplacian [35], we give the initial value of Green's function for laplacian $g(x, k)$ from the solution $(g_d)_j$ of

$$L_c x = I - \frac{1}{N} \mathbf{1} \mathbf{1}^t \quad (\text{Eq. 4.17})$$

$$(g_d)_j = x - \frac{\mathbf{1}^t x}{\mathbf{1}^t \mathbf{1}} \mathbf{1} \quad (\text{Eq. 4.18})$$

where $(g_d)_j$ is the j_{th} column of g_d . x is an arbitrary point and j, k are vertices.

$$g_L(j, k) = (g_d)_j(k) \quad (\text{Eq. 4.19})$$

For the three vertices of $\mathcal{VT}(x)$, j_1, j_2, j_3 and the barycentric coordinate of x , $\omega_1, \omega_2, \omega_3$, using the above way we have $g_L(j_1, k)$, $g_L(j_2, k)$, $g_L(j_3, k)$, then we can determine the initial value $g(x, k)$ by simply linear interpolation

$$g(x, k) = \omega_1 * g_L(j_1, k) + \omega_2 * g_L(j_2, k) + \omega_3 * g_L(j_3, k) \quad (\text{Eq. 4.20})$$

To summarize the process, let $\omega_{x\epsilon}$ be the barycentric weight on each $\epsilon \in \mathcal{VT}(x)$, Ω be the 3×1 column vector containing the three weights. And we let G_L be the $N_t \times 3$ dense matrix

of $g_L(j, k)$, A_{VM} be the Voronoi area divided by manifold area for vertices in $\mathcal{N}(x)$, and it is a $N_n \times 1$ column vector. Then we minimize the l2-norm of the difference of final value and initial value under constraints composed of harmonic B-spline form equations.

$$\begin{aligned}
& \min_G \|G - G_L^T \Omega\|_F^2, \\
& \text{subject to } \Phi_{sx} = \Phi_{sn} L_c G + \Phi_{sn} A_{VM} \\
& \epsilon \in \mathcal{VT}(x), x \in \mathcal{M}, i \in \mathcal{N}(x), j \in \mathcal{N}_{tar}(x), s \in \mathcal{S}
\end{aligned} \tag{Eq. 4.21}$$

4.3.4 Evaluation

In this section, we evaluate the results of our method and simply linear interpolated result from [35]. The test is run on an Intel Xeon 2.50GHz CPU.

Visualization To evaluate the accuracy of our results, we calculate Green's functions on a unit sphere. The mesh contains 48.6K vertices. We compare the function values with the ground truth, the initial value of our optimization process, as well as the result using Lipman's method [35]. The comparison with the third algorithm need create a vertex at the arbitrary point we choose, and calculate the Green's function of the new vertex. We compute the ground truth using the form in [18]

$$\begin{aligned}
& \text{For } \epsilon, \eta \in \omega \text{ with } -1 \leq \epsilon \cdot \eta < 1 \\
& G(\triangle^*; \epsilon, \eta) = \frac{1}{4\pi} \ln(1 - \epsilon \cdot \eta) + \frac{1}{4\pi} - \frac{1}{4\pi} \ln 2
\end{aligned} \tag{Eq. 4.22}$$

Figure 4.5 shows the normalized Green's function value inside 0.2-radius local area around the central point we choose. The central point $x = (0.472637, -0.163706, -0.865918)$ is an arbitrary point inside a triangle face of the unit sphere. In this figure, we render the function values on each vertex in the local area, which may cause sharp corners visually. But the value-radius relationship is still clear. Theoretically the isolines should form concentric circles in

continuous region. Despite our discretization to vertices, the colors should be identical at equal distances from equation Eq. 4.22. However, in the figure the other algorithms all cause distortion in terms of the isolines' shapes. Especially using Lipman's method, although the vertex position is exactly the same with our central point. Due to local geometric feature changes, the Green's functions distort. So their method can be greatly effected by local triangle tessellation. But our method is quite robust against different triangle shapes, since we choose arbitrary central points ignoring the vertex positions.

Figure 4.5: Visualization of normalized Green's function calculated by different approaches. (a) Ground truth (b) Lipman's method with tessellation (c) Lipman's method with tessellation removing maximum value (d) Linear interpolation of neighboring vertices' GF using Lipman's method (e) Our method

Figure 4.6: Visualization of normalized Green's function on sphere calculated by different approaches for arbitrary points. The arbitrary points are samples in 100×100 pixels around the central point (a vertex). ε represents the normalized function error. For our method, distance from each pixel to the central vertex is computed first. The DGF is then induced by the distance. (a) Ground truth. (b) Lipman's method with subdivision surface. (d) Linear interpolation of DGF on neighboring vertices using Lipman's method. (e) Our method.

Precision Analysis Figure 4.7 illustrates the converging rates of our method. The horizontal axis shows the descending average edge length of the unit sphere, and the vertical axis is the average error of Green's functions at 50 arbitrary points, compared with the ground truth.

Relevant to the linear precision of Laplace–Beltrami operator, our discrete Green’s functions also reflect the property of linear precision.

Figure 4.7: converging rates with random sampled source point within 0.2 radius (average edge-length and precision)

4.4 Applications

We can apply our method on different applications that take the discrete Green’s functions as modeling basis. In the following paragraphs we discuss one typical applications: poisson vector graphics (PVG) refinement and leave another application biharmonic distance interpolation to future work.

4.4.1 PVG Refinement

PVG Mesh Solver The PVG mesh solver solves a poisson equation with piecewise linear constraints f on triangular meshes. It is an extension of 2D PVG and introduces Poisson curve and Poisson region on meshes. PVG mesh solver firstly builds Voronoi tessellation $\mathcal{V}$ based on sampled vertices on a mesh and constructs harmonic B-spline using the cell centers as control

points. To compute the color and height information for vertices, it has the following closed-form solution

$$u(\mathbf{x}) = \sum_{\mathcal{V}_j \in N(\mathbf{x})} \lambda_j \psi_j(\mathbf{x}),$$

where $\mathcal{V}_j$ is the one-ring neighbor Voronoi cell of $\mathbf{x}$, λ_j and $\psi_j(\mathbf{x})$ are the control coefficients and basis functions respectively. The harmonic B-spline form solution allows local computation and editing since the basis function decays to zero quickly around the query point. λ_j is the solution of equation

$$\mathbf{L}_{\mathcal{I}} \lambda_{\mathcal{I}} = \mathbf{b} - \mathbf{L}_{\mathcal{B}} \lambda_{\mathcal{B}},$$

where $\mathcal{I}$ means that the elements are inside a region and $\mathcal{B}$ represents the elements on the boundary. $\mathbf{b}$ stores the piecewise linear function f for all Voronoi cells. The basis function $\psi_j(\mathbf{x})$ is calculated as follows

$$\psi_j(\mathbf{x}) = \frac{A_{\mathcal{V}_j}}{A_{\mathcal{M}}} + \sum_{\mathcal{V}_i \in N_1(\mathcal{V}_j)} a_{ij} (\phi_{\mathcal{V}_i}(\mathbf{x}) - \phi_{\mathcal{V}_j}(\mathbf{x})),$$

$$\phi_{\mathcal{V}_j}(\mathbf{x}) = \frac{1}{A_{\mathcal{V}_j}} \sum_{\mathbf{y} \in \mathcal{V}_j} A_{\mathbf{y}} g_{\mathbf{y}}(\mathbf{x}),$$

where a_{ij} comprises where a_{ij} comprises the Laplacian matrix, A represents the area and g is the discrete Green's function defined on vertices.

Refinement Scheme Since our algorithm approximates the discrete Green's function for arbitrary points, we can further replace g in Equation Eq. 4.23 with our product. We firstly select a local region on a mesh for refinement. We calculate the color or displacement information among these fragments, which are distributed with more density and uniformity than vertices, particularly from an anisotropic or rough-tessellated mesh. This refinement procedure also enhance the accuracy in that with uniform sampling fragments inside the selected region, our scheme is robust to triangulation quality as long as we obtain a locally smooth geodesic distance field.

Algorithm 4 Refining PVG mesh

Input: PVG primitives on triangle mesh $M = (V, E, F)$ with constraints, refinement region R , position of fragments X
Output: Refined PVG mesh fragment I

$V_r = V \cup R$
 $\mathbf{d}_{V_r} = \phi$
for each vertex $v \in V_r$ **do**
 $d = \text{ComputeGeodesicDistance}(v, V_r)$
 $\mathbf{d}_{V_r} = \mathbf{d}_{V_r} \cup d$
end for
Compute the control coefficients λ
for each fragment $x \in X \cup R$ **do**
 $\mathbf{d} = \text{ComputeGeodesicDistance}(x, V_r)$
 $\mathbf{g}_{V_r}(x) = \text{ComputeDiscreteGreensFunctions}(x, \mathbf{d}, \mathbf{d}_{V_r})$
 Construct the harmonic B-spline basis function $\{\psi_j\}$ based on $\mathbf{g}_{V_r}(x)$
 $I(x) = \sum_j \lambda_j \psi_j(x)$
end for
 $I = \text{Upsampling}(I)$

4.5 Summary

Our algorithm is capable of computing accurate and robust discrete Green's functions for arbitrary points. On closed-triangle meshes, first we find the connection between geodesic distances and Green's functions by assigning geodesics to the left-hand side of Green's third identity. Then we consider geodesic distances for all the vertices of the mesh, in the form of harmonic B-spline, and we build equivalent formation with the Green's third identity. Utilizing the local properties of harmonic B-splines, we shrink the computation to a local area. Any point's function value (geodesic distances in our algorithms) outside the area, where the basis functions is equal to zero, has no influence on the geodesics of the inner points. Given any arbitrary position on the mesh, we build a dense yet small-scale linear system, the left-hand side is the vector of geodesic distances and the right-hand side is the matrix of control points multiplying the coefficients of the basis functions. However, as a consequence of cutting the local area, we produce a set of boundary vertices which have no boundary values. So we introduce

an optimization process, taking the previous linear system as constraints. As such, we compute the discrete Green's functions between the user-specified position and any vertex inside the local area. Although our arbitrary point based discrete Green's functions achieve better accuracy than vertex based discrete Green's functions algorithms, the computation cost is still high. In the optimization process, as we use a linear interpolation approach to calculate the estimated value of $G(x, j)$, $x \in \mathcal{M}$, $j \in V$, we need three vertices' value first. Since we use vertex based algorithm, we solve a linear system across all the vertices of the mesh. Despite we only need to consider a local area of vertices as optimization constraints, it is slower than vertex based methods, which only compute once. For applications requiring a real time performance, we may pre-process all the possible vertex pairs. However, it still demands strong conditions for the applications.

Chapter 5

Improvements and Extensions

5.1 Fast Construction of Discrete Geodesic Graphs

The text and results of the presented work in this section have been submitted to and conditionally accepted by ACM Transactions on Graphics (TOG).

5.1.1 Overview

This algorithm aims at developing fast method for constructing DGG. In contrast to original DGG, this algorithm abandons the procedure of pruning redundant edges. Our method compute all the edges of the graph in one step. Assuming the triangular faces have $O(1)$ side length and $\Theta(1)$ area, I prove that our method produces DGG with $O(\frac{n}{\sqrt{\epsilon}})$ edges in $O(\frac{n}{\epsilon^{0.75}} \log \frac{1}{\epsilon})$ time, greatly improving the time complexity $O(\frac{n}{\epsilon} \log \frac{1}{\epsilon})$ of our previous approach as described in Chapter 3. The reduced space complexities allow us to compute DGG on very large models. With a low fixed accuracy parameter (e.g. 0.001%) our method has a runtime speed up to 100 times than the original DGG method. Moreover, thanks to its direct nature, our algorithm can handle meshes of moderate degree of anisotropy without adding any pseudo vertex.

There are two main drawbacks in previous DGG, leading to its high computational cost. First, as observed in [62], the majority of the candidate edges are redundant and will be pruned

later. Wang et al. showed that computing the candidate edges takes $O(\frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$ time, implying that for each vertex, there are $O(\frac{1}{\varepsilon})$ windows propagated. However, it is known that there are empirically $O(n^{1.5})$ windows for a single source vertex of a triangle mesh with fairly regular triangulations [55]. Since the average degree of a DGG is $O(\frac{1}{\varepsilon^{0.5}})$, the DGG edges incident to a vertex v_i roughly cover a region with $O(\frac{1}{\varepsilon^{0.5}})$ vertices. As a result, an ideal DGG construction algorithm should only generate $O(\frac{1}{\varepsilon^{0.75}})$ windows for each vertex. Second, the theoretical analysis of Wang et al.’s algorithm is based on the assumption that vertices are distributed uniformly. Such an assumption, however, does not hold for anisotropic meshes. To handle anisotropic meshes, their algorithm must add pseudo vertices and edges to poorly sampled regions, which increases not only the construction time, but also the graph complexity.

Examining closely, I find that the high time complexity of DGG algorithm is due to its indirect nature and the *loose* terminating condition in computing candidate edges. As a result, the algorithm sweeps a region much larger than necessary to compute the candidate edges. During window propagation, a direct window $w = (e_{ij}, b_0, b_1, d_0, d_1, 0)$ stops propagating when an endpoint of the edge e_{ij} can be used as a relay to approximate any geodesic path going through w . Intuitively speaking, their algorithm always takes the vertices *on* the wavefront as relay and does not consider those which are *inside* the swept region. However, interior vertices often provide better approximation than those on the wavefront, implying that a window has a distance longer than necessary and the terminating condition d_{max} is too loose. As a result, their algorithm yields redundant edges that will be deleted later.

To improve performance, we propose an accuracy-aware window propagation scheme, which stops propagating windows as soon as the desired accuracy is reached. Moreover, we only assume that triangles have $\Theta(1)$ area and $O(1)$ edge length, i.e., they are bounded by some resolution-independent constants. Such an assumption is weaker than the uniform distribution assumption in previous DGG. As a result, our method is able to compute DGG edges directly for those meshes without adding pseudo vertices and edges.

5.1.2 Runtime Performance

Our method is $\Omega(\frac{1}{\varepsilon^{0.25}})$ faster than Wang et al.’s algorithm. Computational results show that for $\varepsilon \in [0.01\%, 0.1\%]$, it runs 3-40 times faster than the indirect algorithm and the speedup can reach 100 for $\varepsilon = 0.001\%$. Figure 5.1 shows the profiling of our method and Wang et al.’s method on three representative models.

Figure 5.1: Runtime profiling on three representative models, Bunny $|V| = 72K$, Armadillo $|V| = 140K$, Gargoyle $|V| = 350K$ (from left to right). Since the mesh quality is good, Wang et al.’s method skips the step of adding pseudo vertices and edges. W_d and W_i are the number of windows for our direct method and Wang et al.’s indirect method, and T_d and T_i are the run time. I observe that the performance improvement T_d/T_i is roughly proportional to the window ratio W_d/W_i .

5.1.3 Robustness to Triangulation

In [62], Wang et al. assumed that vertices are uniformly distributed on meshes. Such an assumption plays a critical role in analyzing in error and complexity analysis (see Theorem 3.1). This *strong* assumption holds for meshes with fairly good triangulations, such as the models reported in Table 1, but it fails for anisotropic meshes. As a result, Wang et al.’s algorithm has to add pseudo vertices and edges to poorly sampled region in order to ensure the vertex distribution assumption holds. This additional step increases not only the construction time, but also

the graph complexity. In contrast, I assume that triangles have $O(1)$ edge length and $\Theta(1)$ area, which is weaker than the uniform distribution assumption. Since such an assumption holds for many real-world and synthetic models, our method can work without adding pseudo vertices.

Figure 5.2: Comparing DGG with HM and its variants. For each row on the left is the results of HM and on the right is our results. From the error map we can see that, our method control the accuracy well.

To quantitatively evaluate both methods on anisotropic meshes, I define $\tau(f) = \frac{H}{2\sqrt{3}S}$ the quality measure of a triangular face f , where S and H are the area and perimeter of f . Then I define $\tau = \frac{\sum_{f \in F} \tau(f)}{|F|}$ to measure the anisotropy degree of mesh M . Obviously, $\tau \geq 1$ for all triangle meshes. An isotropic mesh has $\tau = 1$. In general, the larger the value of τ , the higher degree of anisotropy. I evaluate the accuracy of both methods on the Thingi10K dataset with various τ .

5.2 Discrete Geodesic Graphs on Point Clouds

5.2.1 Overview

Given a set of point $\chi = \{x_1, \dots, x_q, x_{q+1}, \dots, x_n\} \in \mathbb{R}^m$, we present a new approach for the computation of geodesic map on arbitrary point x_q . The key idea is to define which points are adjacent to x_q inside an h -offset band that the errors of Euclidean distance between x_q and its neighbors are bounded. Once these local approximate geodesics are available, we can then determine the shortest path from the source point x_q to every point in graph.

Recall that from Mémoli's work in [39], given x_p and x_q be two points in a submanifolds of $\mathbb{R}^d$ represented in point clouds S , they proved the error of their approximated geodesic distance within a h -offset shell is bounded by

$$|d_S(p, q) - d_{\Omega^h}(p, q)| \leq C_S \sqrt{h}, \quad (\text{Eq. 5.1})$$

where d_S and d_{Ω^h} represent intrinsic and extrinsic distances respectively. The constant C_S depicts the geometry of S and does not depend on h . Note that the computation of $d_{\Omega^h}(p, q)$ is undertaken in Cartesian grids using fast matching algorithm.

Motivated by the convergence and error bounded in an offset band, our method approximates a long geodesic path γ by a series of short Euclidean paths that are falling within a h -offset

Figure 5.3: Results on the subset of the Thingi10K shape repository, consisting of 4,439 models, each of which is connected, manifold triangle mesh. Among them, I further selected 590 models with low anisotropy measure $\tau \leq 3$ so that Wang et al.'s method can be applied. The gray planes denote the user specified tolerance ε for Wang et al.'s method and the accuracy parameter ε for our method. Each point corresponds to a 3D model. I color the models passing the test ($\bar{\varepsilon} \leq \varepsilon$) in green, and the ones failing the test ($\bar{\varepsilon} > \varepsilon$) in red. See the supplementary material for the detailed results.

band. The central idea is similar to Wu's method, named discrete geodesic graph (DGG), which replaces γ by a number of short geodesic paths that are close enough to γ . However, due to the lack of manifold details in point clouds, it is difficult to find an exact geodesic path in between points, even those points are close. Instead, we consider using multiple short Euclidean paths δ to constitute a long geodesic path, where the error of each δ is bounded according to the following theorem.

Definition 1 . Consider $\delta(p, q)$ a Euclidean path from point p to q . δ is called directly connected if the path is linked in a compact region, and vice versa if any part of the path lying outside the region.

A direct Euclidean path is the shortest path between two points in the domain.

Theorem 1 . Let p, q be two points from $\chi \subseteq \Omega^h$ that can be directly connected inside Ω^h , where Ω^h is the band constructed from χ . Then there exists a $h_\delta \leq h$ such that $|d_S(p, q) - d_\delta(p, q)| \leq C_\delta \sqrt{h_\delta}$, where $C_\delta \leq C_S$ is a constant depending on partial geometry in between p and q .

Proof. By simple considerations, a direct connected path can exist in a tighter band region. If the orthogonal distance (maximum depth of curvature) between $\gamma(p, q)$ and $\delta(p, q)$ is h_δ , we can easily find an h_δ -offset band that completely cover up the path $\delta(p, q)$.

Next, we give the definition of an "accurate" geodesic path on point cloud.

Definition 2 . Let $\gamma(p, q)$ be a path from arbitrary point p and q . Given a set of points $g \in G$,

Figure 5.4: Geodesic path on point cloud. The path (dashed brown) is more accurate than the path in blue, because some distances from $f(g_i)$ to G is large (in red).

which are sampled from $\gamma(p, q)$ and independent of χ . Each of g has a closest point $f(g_i) \in \chi$. A path is said to be accurate, not merely the shortest distance of $\gamma(p, q)$, but also with the smallest distance for each $f(g_i)$ to G . (see fig.5.4)

A geodesic path on mesh doesn't necessary to pass through all the vertices as long as it is constrained on surface with shorter distance. Similarly, for point cloud data, we assume that a path is the shortest only if it is constrained in the thinnest shell region with minimum distance. In this paper, we use Euclidean distance to measure the distance of $\gamma(p, q)$ as well as the distance between G and χ .

We generalize the idea and come up with the following theorem.

Theorem 2 . Let p, q be two points from $\chi \subseteq \Omega^h$ that can be directly connected inside Ω^h . A Euclidean path $\delta(p, q)$ consist of $g = \{g_1, \dots, g_m\} \in G$, where each of them obtains the closest point $f(g_i) \in \chi$. This path $\delta(p, q)$ is **sufficiently accurate** if shortest distance for all closest point to G is smaller than a tolerance ϵ , where $\epsilon \leq h$.

The parameter ϵ bounds the error of Euclidean path δ with $C_\delta \sqrt{h_\epsilon}$ according to theorem 1. Therefore when we look for the neighbors from each point $\in \chi$, it is guaranteed that the path

Figure 5.5: An example of searching neighbors for point χ_p . The points in green and yellow are directly connected to χ_p inside Ω^h . But only yellow points will be discarded if the distance is larger than K .

to their neighbors are accurate within a controllable error conditions. In fact, the parameter ϵ is not a constant because points are connected with uneven or sparsely distributed neighbors.

According to our observation, among all the directly connected neighbors, any points with the longest distance along the same direction imply they could have the largest value of ϵ . It is because they have a higher possibility to include some closest points $\in \chi$ with large ϵ . A simply way to guarantee small value of ϵ is to discard those neighbors with long distance. However, if we only concern the local error, excessive Euclidean paths will occur and thus maximize the global (accumulated) error. Therefore, we want to find an optimal distance K that can minimize both the ϵ and the global error. In later section, we will investigate how the global error react to the change of K and discuss the methods that can determine an appropriate K .

Intuitively, we use a geodesic disk, where the radius is denoted by K to control the distance of neighbors.

We define K by

$$K = \frac{\sum_{i=1}^{m(h)} \|\delta_i\|}{\alpha \cdot m(h)}$$

where α is a user-defined parameter for any points with m number of directly connected neighbors, where m is a function of offset distance h . Because the number $m(h)$ is able to determine the “flatness” of a point p , we can observe the relative curvature from this quantity. Clearly, a higher curvature point can only be tracked in small range of neighbors within the shell, and vice versa.

The main advantage of this method is that the radius K varied based on curvature so that the corresponding Euclidean distance traverses approximately along the surface. However, this scheme (A) has a limitation. If the curvature of input points vary largely (e.g. an object contains sharp features), changing the proportion of neighbors is not the optimal way to define K . For example, a large value of α could lead to zero neighbors at highly curved region, while a small value of α may induce inaccuracy in flatten region. We propose another scheme (B) to determine K .

$$K = wid \times (h + \alpha)$$

The K is defined as a constant, and all the variables can be assigned by user. This method usually applies on uniformly distributed points with sharp features, and is a great fit for parallelism. Taking advantage of the local nature and no connectivity information involved, our algorithm is insensitive to noise and can be applied on many type of geometric representation, including regular grids, polygonal meshes, and unstructured point clouds.

After the neighbors of point clouds are determined, we construct a neighborhood graph and weight it by Euclidean distance instead of geodesic distance. And then, we estimate the geodesic distance between a source point with all other points by computing their shortest path distance in the graph.

In this section, I present experimental results which demonstrate the robustness and effectiveness of our method to compute discrete geodesic graphs on 3D Point Clouds. All the experiments were conducted on a PC equipped with an Intel Xeon E5-2609 v2 CPU running at 2.50GHz, and the framework was implemented in C++.

5.2.2 Accuracy

In this section the tested models are triangulated meshes removing all the connecting edges and faces. I compare our result to the exact polyhedral distance [41] [65] on triangulated surface. I also examine errors with other approximate methods such as heat method [11] and fast marching method [39] using the mesh vertices as input. Figure 5.6 visualizes the relative difference $|d(x, y) - \hat{d}(x, y)|/d(x, y)$ between the exact distance $d(x, y)$ and the approximate distance $\hat{d}(x, y)$ computed from different methods. Results show that our method outperforms the fast marching and heat method by up to an order of magnitude (see Table 5.1).

I study the impact of changing the value of K with respect to the accuracy using our method. The result confirms that the geodesic error can reach to a minimum when the radius K is small but not smaller than h . When considering the value of K becomes smaller than the offset distance, the error may become critical because the number of neighbor points that can fulfill the distance requirement is very small, or zero. As shown in figure 5.10, for $\alpha = 2.6$ in our method, more than 20% of points obtain the distance in less than 0.001% of error; while for $\alpha = 2.5$, more than 70% of points obtain the distance in less than 0.003% of error. Therefore, the e_{avg} of dragon model for $\alpha = 2.5$ is smaller than when $\alpha = 2.6$. Similar to the our method, I also make the radius K as small as possible in scheme A by enlarging the α .

One of the challenges of this work is to choose an appropriate values for α , h and res , since the radius K is influenced by these parameters. Intuitively, h and res mean all points are assumed to be able to find at least one neighbor within an Euclidean distance $d_W = wid \times h \times 2$. A large

d_W is assigned if the data points are sparsely distributed while a low value of d_W is appropriate for dense points. The α makes a balance between the d_W and K . For scheme A, according to our experiments, the accuracy is the optimal when setting $\alpha \approx h$. But for our method, we can only estimate that the α has to be larger than h for accurate distance approximation. The difference between two schemes are shown in Table 5.1.

5.2.3 Robustness

Our method is robust because it depends on the position of points and no topology information is used. Also, using offset band to look for neighbors is tolerant of noise in the algorithm. Figure 5.8 and Figure 5.11 verify that the DGG method works well on noisy and non-uniformly distributed point clouds. For such cases we use a large value of h to create a band that able to cover up all the points and an appropriate value of α to adjust the accuracy. Given the mesh surface, I can compute the geodesic distance from its original vertices or a new set of points by resampling the surface.

5.2.4 Efficiency

The geodesic distance computation is divided into two phases: the first is the precomputation phase, and the second is the real-time phase. In the first phase, the neighborhood graph for each points can be precomputed independent of the querying points. Unfortunately the time to compute the graph is expensive, takes $O(n \times m(h) \times res)$ in the worst case , which arises when the number of points or d_W increased. But thanks to the local nature of the DGG method, it is practical to implement the algorithm on a graphics processing unit (GPU) to gain significant speedup over the serial version. The second phase is dynamic calculation at arbitrary source point, therefore is implemented in CPU.

Figure 5.6: Geodesic map on models. Top row: Exact geodesic computation. Middle row: approximated geodesic by our method. Bottom row: approximated geodesic by [39]. The mean relative error are reported in table 5.1.

Table 5.1: Comparing the mean relative error of our methods with [39] on the same offset band resolution 512^3 .

Model	size	Scheme A			Our method			FMM [39]	
		h -offset	α	e_{avg}	h -offset	α	e_{avg}	h -offset	e_{avg}
Sphere	655362	2	2.9	$7.21E-5$	2	10	$7.14E-5$	1.75	0.037
Torus	57200	4	2.5	0.0012	4	11	0.0017	3.4	0.018
Terrain	66049	2	4.0	0.0031	2	2.8	0.0038	1.75	0.027
Bunny	30002	4	3.2	0.0023	4	8	0.0017	3.4	0.024
Dragon	138272	2	4.2	0.0066	2	2.5	0.0028	3.4	0.024

Table 5.2: Experiments on uniformly and non-uniformly distributed point clouds

Model	size	h -offset	α	res	pre-process time
Dinosaur (dense)	57210	4	3.0	512	24.1
Dinosaur (sparse)	22947	4	3.0	512	10.2
GLady	44230	4	3.0	512	17.3
Angelo	13903	4	3.0	512	7.9

Table 5.3: Comparisons between different models in offset, β , ε , graph size and time factors. T_p represents pre-computation time in minutes. T_d represents Dijkstra time in seconds.

		res	h -offset	α	Graph Size	T_p (min)	T_d (s)
Kitten	600k	1024	1	0.0	2.19GB	41.4	8.73
Kitten	600k	1024	1	0.0	2.09GB	41.3	8.44
Kitten	600k	1024	1	0.0	842MB	39.4	3.27
Kitten	600k	1024	1	2^{-03}	390MB	43.9	1.83
David Head	396k	512	1	0.01	245MB	19.1	1.05
David Head	396k	512	1	0.005	336MB	19.9	1.38
David Head	396k	512	1	0.1	53.8MB	17.3	0.35
Human Scan	84k	512	2	0.0	1.3GB	20.4	3.58
Human Scan	84k	512	2	0.0	491MB	19.5	1.66
Human Scan	84k	512	2	0.0	234MB	18.5	0.72
Inukshuk	206k	512	2	1^{-04}	487MB	80.3	1.63
Inukshuk	206k	512	2	2^{-03}	374MB	64.1	1.26
Inukshuk	206k	512	2	2^{-02}	144MB	52.4	0.61

Figure 5.7: Example of point clouds with holes on potential surface.

Figure 5.8: Example of noisy point clouds.

Figure 5.9: Centroidal Voronoi Tessellation constructed by (a) RVD-based method and (b) our method with 50 generators.

Figure 5.10: Relative error changes with different value of α . I plot the distribution of the distance error for the variable of points. The geodesic distance is computed from a single source point.

Figure 5.11: Yoga model with different value of h and α .

5.3 Tracing Iso-distance Curve

The text and results of the presented work in this section have been accepted by the Computational Visual Media Conference 2019 and to be published (accepted by) IEEE Transactions on Visualization and Computer Graphics (TVCG).

5.3.1 Overview

Geodesic isolines are very helpful for visualizing how the intrinsic metrics varies with different geometric shapes and exhibiting how accurate a given geodesic approach is. A common solution for computing geodesic isolines on triangle meshes is to compute the distances for mesh vertices and then use linear interpolation to approximate the distances for points inside a triangle face. Although it is easy to implement, this method is not accurate since geodesic distance field is highly non-linear. As a result, the traditional way to visualize geodesic isolines may cause conspicuous artifacts, i.e., it is difficult to reconstruct sharp corners or topological changes of the real isolines even if the input mesh is subdivided into a very high resolution.

In this paper, I show that the geometric and topological features of geodesic isolines can be encoded by Apollonius diagrams, just like that a medial-axis surface can be reported by the Voronoi diagrams w.r.t. a dense set of boundary samples. In implementation, I use a cubic function to approximate the change of the distance function on each mesh edge, generate the Apollonius diagrams w.r.t. a set of weighted sample points distributed on mesh edges, and then partition each triangle face into a set of sub-regions such that the distance field in each sub-region can be deemed to change linearly. Extensive evaluations show that our approach is able to produce much more accurate geodesic isolines than the commonly used linear interpolation with only a little extra computational cost.

Figure 5.12: The cell of P_1 is S_1 . The Apollonius vertices are shown in yellow, and the Apollonius edge are shown in green.

Figure 5.13: Algorithmic pipeline. (a) shows the input model. (b) shows a mesh triangle and a geodesic path (in red) and uses cubic function interpolation. (c) shows a set of sites (in red) in triangle. (d) builds Apollonius diagram for those sites. (e) builds the Apollonius vertices and Steiner point triangulation. (f) shows the isolines of the geodesic distance field.

Figure 5.14: Our method is able to compute high-quality isolines of geodesic distance fields on triangle meshes, whereas the simple linear interpolation fails to capture their geometric and topological features. To visualize the quality, I overlay the results with the exact solution. The red, black and blue-gray colors indicate the results of our method, the piecewise linear method and the exact solution.

5.3.2 Performance

Our algorithm has $O(kn)$ space complexity and $O(kn \log k)$ time complexity and works on common models used in geometry research field. The running time of our method is composed of three parts: interpolation, Apollonius diagram calculation and extraction of isolines. The Apollonius calculation usually costs around 90% of the whole running time. Take the 49,990-vertices Buddha for example, the timing of these three parts is 1.65s, 38.97s, 1.44s. On a smaller 4,990-vertices Buddha, the timing is 0.16s, 3.29s, 0.11s. The number of Steiner points on each edge in these tests is 2. We can see that the larger Buddha results in an approximately 10 times slower running time. Table 5.4 presents our runtime performance and the comparison with the piecewise linear method in terms of accuracy. Isolines are stored as discrete line segments based on triangles.

Figure 5.15: The memory and time of computing the isolines on the Buddha and Gargoyle model with 2,000 to 20,000 vertices and 0, 2, 10, 20, 50 Steiner points.

5.3.3 Approximation Error

I calculate the approximation error as follows. Compare the isoline distance with the exact distance on the middle point of each isoline segment and compute the relative error. The approximation error is then the mean value of all the relative errors. I pick the middle points to avoid the case that the two end points of an isoline segment approximate the exact distance well while the segment is way off the exact isoline. The resulting mean error is able to reflect the deviation of isoline around the ridges. From Table 5.4, our method traces the isolines with significant lower mean error than the piecewise linear method. Note that we only show the results with maximum 2 Steiner points on the edge ($k = 2$) because the mean error reaches a relatively low point when $k = 2$ (see Figure 5.16). From the visualization in Figure 5.17, we zoom out over the ridges being captured via the Apollonius diagram. Sharp features of isolines are well preserved compared with the piecewise linear method. And our method is

Table 5.4: Runtime performance on 5 low resolution models and 5 higher resolution models. On each model there are $|V|$ vertices and k Steiner points on each edge. t and RAM show the running time and the peak memory usage of our algorithm. Timing unit is one second. The experiments are done on a workstation with an Intel Xeon 2.50 GHz CPU and 8 GB RAM. $\bar{\varepsilon}$ is the mean error of our algorithm and $\bar{\varepsilon} - Linear$ is the mean error of the piecewise linear method.

Model	$ V $	k	t (s)	RAM (MB)	$\bar{\varepsilon}$	$\bar{\varepsilon} - Linear$
Beetle	0.6K	0	0.211	10	0.170%	0.516%
		2	0.456	11	0.120%	
	5K	0	1.198	58	0.082%	0.223%
		2	2.637	64	0.019%	
	10K	0	2.970	111	0.021%	0.096%
		2	6.332	122	0.009%	
Bottle	1K	0	0.339	17	0.132%	0.425%
		2	0.638	23	0.096%	
	10K	0	2.970	111	0.021%	0.096%
		2	6.332	122	0.009%	
	12K	0	4.535	135	0.038%	0.140%
		2	9.608	148	0.037%	
Fertility	1.2K	0	0.367	19	0.091%	0.227%
		2	0.745	26	0.074%	
	12K	0	4.535	135	0.038%	0.140%
		2	9.608	148	0.037%	
	5K	0	0.452	21	0.039%	0.199%
		2	0.807	29	0.030%	
Gargoyle	5K	0	0.452	21	0.039%	0.199%
		2	0.807	29	0.030%	
	50K	0	19.869	534	0.005%	0.019%
		2	49.042	588	0.004%	
	5K	0	1.275	58	0.055%	0.100%
		2	2.868	64	0.028%	
Buddha	5K	0	1.275	58	0.055%	0.100%
		2	2.868	64	0.028%	
	50K	0	19.167	534	0.012%	0.035%
		2	48.988	586	0.004%	
	5K	0	1.275	58	0.055%	0.100%
		2	2.868	64	0.028%	

able to recover the ridges consistently with different mesh resolution, number of sources and triangulation quality.

5.3.4 Extension to PDE methods

Endowed with distance-on-vertex input, our method is able to enhance isoline tracing on any existing geodesic computing approaches. It is especially important to PDE methods as these methods is not capable of computing the distance in the interior of triangles directly. There is no efficient isoline tracing approach other than piecewise linear interpolation. We extend

Figure 5.16: The comparison of mean error between our method and the piecewise linear method. Our method (in lighter orange) significantly reduces the mean error compared with the piecewise linear method (in dark orange).

our method to two popular PDE methods, i.e., the Fast Marching Method (FMM) and the Heat Method (HM). After using these methods to compute the geodesic distances for mesh vertices, we apply our method to extract the isolines. Figure 5.18 illustrates that our method trace isolines in good quality with approximate distances. Note that the mean error of our method is slightly better than the piecewise linear method, with less improvement than using the exact distance. This is because PDE methods initially produce approximation error. As our calculation in accuracy takes the exact discrete geodesic distance as baseline, the error remains or accumulates. In spite of that fact, our method overcomes the flat-ridge artifact produced by

Figure 5.17: Visualization of our method with $k = 2$. The isolines of our method are rendered in black color and those of the piecewise linear method in red. Top Left: Beetle $|V| = 5K$, $\varepsilon = 0.019\%$, 2 sources. Top Right: Buddha $|V| = 5K$, $\varepsilon = 0.028\%$, 1 source. Bottom Left: Bottle $|V| = 1K$, $\varepsilon = 0.096\%$, 3 sources. Bottom Right: Fertility $|V| = 1.2K$, $\varepsilon = 0.074\%$, 2 sources.

Figure 5.18: Top: the Fast Marching Method. Bottom: the Heat Method. The isolines of our method are rendered in black and the piecewise linear method in red.

piecewise linear interpolation (Figure 5.18 top) and the partial isoline inaccuracy caused by PDE’s sensitivity to triangulation quality (Figure 5.18 bottom).

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis I propose a novel graph-based method for solving the discrete geodesic problems. Highlights and advantages of DGG include direct error control, extensive computation capabilities for arbitrary non-vertex points, theoretical soundness, compact size, linear-time solving complexity, and extension to other discrete domains, such as point clouds.

Based on the faster and more precise geodesic distances computed with DGG, especially distances between arbitrary points and vertices, I assign the distances as function values for Green's third identity and build the equivalent localized harmonic B-spline. I present a quadratic programming scheme to optimize discrete Green's functions for arbitrary points. I also prove that the function values are more accurate description of Green's functions on manifold triangle meshes than current algorithm only computed on vertices. I may further adapt our discrete Green's functions on various applications, especially for the continuation of discrete domains. In addition to discrete geodesic graphs, I further improve the computational efficiency and extend its variant methods to point clouds as well as iso-curve tracing. These methods outperform current algorithms in their corresponding field.

6.2 Future Work

6.2.1 Closed-form Solver for Poisson Equations and Non-homogeneous Bi-Laplace Equations

Solving Poisson equations and Bi-Laplace equations has been applied in many directions. However, robust closed-form solutions are great challenges on 3D mesh surfaces. Although on 2D plane, using the exact Green's functions we may get the desired closed-form solutions, such as diffusion textures [54], for arbitrary 3D mesh Green's functions at random points have no definition at all. The discrete Green's functions provide calculation for such Green's functions, and in turn I may apply them on finding closed-form solutions for various scenarios. For diffusion process on 3D surfaces, we may avoid the explicit color representation given by diffusion textures which requires 2D to 3D mapping. We directly compute colors on the corresponding 3D positions. It would add intuition to additional functions like editing.

6.2.2 Discrete Green's Functions for Bi-Laplace Operator

In this report, to calculate the Green's function for laplacian, I use analog of Lipman et al. [35] formation of Green's function associated with bi-laplacian. Originally the approach shows that the discrete Green's function g_d can be calculated by

$$L_c^T A^{-1} L_c x = I - \frac{1}{N} \mathbf{1} \mathbf{1}^t \quad (\text{Eq. 6.1})$$

$$(g_d)_j = x - \frac{\mathbf{1}^t x}{\mathbf{1}^t \mathbf{1}} \mathbf{1} \quad (\text{Eq. 6.2})$$

where $(g_d)_j$ is the j_{th} column of g_d . If replacing $A^{-1} L_c$ in our formation with $L_c^T A^{-1} L_c$, I can easily extend to biharmonic case. And in this case, I conclude the computation of bi-Laplace operator on the surface to both vertices and arbitrary points. Hou et al. [25] use biharmonic

B-splines on meshes and optimize the knot positions. Given discrete Green's functions for bi-Laplacian, biharmonic B-splines based on knots and control points located at arbitrary position on surface can better fit the original data.

6.2.3 Unified Framework for Computing Discrete Distances

There have been constant effort in improving distance metric on manifolds. Lipman et al. [35] proposed biharmonic distances which is a typical spectral distance metric. Heat method [11] first formulate a hybrid of pure geodesics and spectral distances. Earth-mover distance (EMD) [52] generalizes these two scenarios and is shape-aware globally. EMD is symmetric and also satisfies triangle inequality. I may also unify geodesic distances and spectral distances with Green's functions, extending the domain from vertices to arbitrary points. Another potential contribution would be to extend earth-mover distances to continuous region on meshes.

Figure 6.1: Biharmonic distance field visualization from [35]. (a) Biharmonic distance from a source point (darkest blue) (b) Isolines follow the shape indifferently to the missing parts and holes (c) Earth-mover distances under different parameter conditions (d) Comparison of geodesic distances

References

- [1] Pankaj K Agarwal, Sarel Har-Peled, Micha Sharir, and Kasturi R Varadarajan. Approximating shortest paths on a convex polytope in three dimensions. *Journal of the ACM*, 44(4):567–584, 1997.
- [2] Lyudmil Aleksandrov, Mark Lanthier, Anil Maheshwari, and Jörg-Rüdiger Sack. An ε -approximation for weighted shortest paths on polyhedral surfaces. In *Proceedings of the 6th Scandinavian Workshop on Algorithm Theory (SWAT '98)*, pages 11–22, 1998.
- [3] Alexander G. Belyaev and Pierre-Alain Fayolle. On variational and pde-based distance function approximations. *Computer Graphics Forum*, 34(8):104–118, 2015.
- [4] Dimitri P Bertsekas. *Network Optimization: Continuous and Discrete models*. Athena Scientific Belmont, 1998.
- [5] John C. Bowers, Jonathan Leahey, and Rui Wang. A ray tracing approach to diffusion curves. In *Proceedings of the Twenty-second Eurographics Conference on Rendering, EGSR '11*, pages 1345–1352, 2011.
- [6] Simon Boyé, Pascal Barla, and Gaël Guennebaud. A vectorial solver for free-form vector gradients. *ACM Trans. Graph.*, 31(6):173:1–173:9, 2012.
- [7] Marcel Campen and Leif Kobbelt. Walking on broken mesh: Defect-tolerant geodesic distances and parameterizations. *Comput. Graph. Forum*, 30(2):623–632, 2011.
- [8] Jindong Chen and Yijie Han. Shortest paths on a polyhedron. In *Proceedings of the Sixth Annual Symposium on Computational Geometry*, pages 360–369. ACM, 1990.
- [9] Fan Chung and S.-T. Yau. Discrete green’s functions. *Journal of Combinatorial Theory, Series A*, 91(1):191 – 214, 2000.

- [10] Ronald R. Coifman and Stéphane Lafon. Diffusion maps. *Applied and Computational Harmonic Analysis*, 21(1):5 – 30, 2006. Special Issue: Diffusion Maps and Wavelets.
- [11] Keenan Crane, Clarisse Weischedel, and Max Wardetzky. Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Transactions on Graphics*, 32(5):152, 2013.
- [12] Eugene D’Eon and Geoffrey Irving. A quantized-diffusion model for rendering translucent materials. *ACM Trans. Graph.*, 30(4):56:1–56:14, July 2011.
- [13] Miles Detrixhe, FréDÉRic Gibou, and Chohong Min. A parallel fast sweeping method for the eikonal equation. *J. Comput. Phys.*, 237:46–55, 2013.
- [14] Zeev Farbman, Raanan Fattal, and Dani Lischinski. Convolution pyramids. *ACM Trans. Graph.*, 30(6):175:1–175:8, December 2011.
- [15] Powei Feng and Joe Warren. Discrete bi-laplacians and biharmonic b-splines. *ACM Trans. Graph.*, 31(4):115:1–115:11, July 2012.
- [16] Mark Finch, John Snyder, and Hugues Hoppe. Freeform vector graphics with controlled thin-plate splines. *ACM Trans. Graph.*, 30(6):166:1–166:10, 2011.
- [17] F. Fouss, A. Pirotte, J. Renders, and M. Saerens. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 19(3):355–369, March 2007.
- [18] W. Freeden and M. Schreiner. *Spherical Functions of Mathematical Geosciences: A Scalar, Vectorial, and Tensorial Setup*. Advances in Geophysical and Environmental Mechanics and Mathematics. Springer Berlin Heidelberg, 2008.
- [19] Qian Fu, Fei Hou, Qian Sun, Shi-Qing Xin, Yong-Jin Liu, Wencheng Wang, Hong Qin, and Ying He. Decorating 3d models with poisson vector graphics. *Computer-Aided Design*, 102:1–11, 2018.
- [20] Zhisong Fu, Won-Ki Jeong, Yongsheng Pan, Robert M. Kirby, and Ross T. Whitaker. A fast iterative method for solving the Eikonal equation on triangulated surfaces. *SIAM J. Sci. Comput.*, 33(5):2468–2488, 2011.

- [21] George Green. An essay on the application of mathematical analysis to the theories of electricity and magnetism. *Journal für die reine und angewandte Mathematik*, 44:356–374, 1852.
- [22] Sarel Har-Peled. Constructing approximate shortest path maps in three dimensions. In *Proceedings of the Fourteenth Annual Symposium on Computational Geometry*, pages 383–391, 1998.
- [23] John Hershberger and Subhash Suri. Practical methods for approximating shortest paths on a convex polytope in $\mathbb{R}^3$. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 447–456, 1995.
- [24] Michael Hofer and Helmut Pottmann. Energy-minimizing splines in manifolds. *ACM Transactions on Graphics (TOG)*, 23(3):284–293, 2004.
- [25] F. Hou, Y. He, H. Qin, and A. Hao. Knot optimization for biharmonic b-splines on manifold triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, PP(99):1–1, 2016.
- [26] F. Hou, Q. Sun, Z. Fang, Y. Liu, S. Hu, A. Hao, H. Qin, and Y. He. Poisson vector graphics (pvg). *IEEE Transactions on Visualization and Computer Graphics*, pages 1–1, 2018.
- [27] Peter Ilbery, Luke Kendall, Cyril Concolato, and Michael McCosker. Biharmonic diffusion curve images from boundary elements. *ACM Trans. Graph.*, 32(6):219:1–219:12, 2013.
- [28] Stefan Jeschke. Generalized diffusion curves: An improved vector representation for smooth-shaded images. *Computer Graphics Forum*, 35(2), 2016.
- [29] Stefan Jeschke, David Cline, and Peter Wonka. A GPU Laplacian solver for diffusion curves and Poisson image editing. *ACM Trans. Graph.*, 28(5):116:1–116:8, 2009.
- [30] Stefan Jeschke, David Cline, and Peter Wonka. Rendering surface details with diffusion curves. *ACM Trans. Graph.*, 28(5):117:1–117:8, 2009.

- [31] T. Kwok, W. Wan, J. Pan, C. C. L. Wang, J. Yuan, K. Harada, and Y. Chen. Rope caging and grasping. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1980–1986, May 2016.
- [32] T. Kwok, Y. Zhang, and C. C. L. Wang. Efficient optimization of common base domains for cross parameterization. *IEEE Transactions on Visualization and Computer Graphics*, 18(10):1678–1692, Oct 2012.
- [33] Henrik Lieng, Flora P. Tasse, Jiří Kosinka, and Neil A. Dodgson. Shading curves: Vector-based drawing with explicit gradient control. *Computer Graphics Forum*, 2015.
- [34] Yaron Lipman, David Levin, and Daniel Cohen-Or. Green coordinates. *ACM Trans. Graph.*, 27(3):78:1–78:10, August 2008.
- [35] Yaron Lipman, Raif M. Rustamov, and Thomas A. Funkhouser. Biharmonic distance. *ACM Trans. Graph.*, 29(3):27:1–27:11, July 2010.
- [36] Yong-Jin Liu. Exact geodesic metric in 2-manifold triangle meshes using edge-based data structures. *Computer-Aided Design*, 45(3):695–704, 2013.
- [37] Yong-Jin Liu, Chun-Xu Xu, Dian Fan, and Ying He. Efficient construction and simplification of Delaunay meshes. *ACM Transactions on Graphics*, 34(6):174:1–13, 2015.
- [38] Yong-Jin Liu, Qian-Yi Zhou, and Shi-Min Hu. Handling degenerate cases in exact geodesic computation on triangle meshes. *The Visual Computer*, 23(9-11):661–668, 2007.
- [39] F. Méholi and G. Sapiro. Distance functions and geodesics on submanifolds of $\mathbb{R}^d$ and point clouds. *SIAM J. Appl. Math.*, 65(4):1227–1260, 2005.
- [40] Facundo Méholi and Guillermo Sapiro. Fast computation of weighted distance functions and geodesics on implicit hyper-surfaces. *Journal of Computational Physics*, 173(2):730–764, 2001.
- [41] Joseph S.B. Mitchell, David M. Mount, and Christos H. Papadimitriou. The discrete geodesic problem. *SIAM Journal on Computing*, 16(4):647–668, 1987.
- [42] Mattia Natali, Silvia Biasotti, Giuseppe Patanè, and Bianca Falcidieno. Graph-based representations of point clouds. *Graphical Models*, 73(5):151–164, 2011.

- [43] Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. Diffusion curves: A vector representation for smooth-shaded images. *ACM Trans. Graph.*, 27(3):92:1–92:8, 2008.
- [44] Wai-Man Pang, Jing Qin, Michael Cohen, Pheng-Ann Heng, and Kup-Sze Choi. Fast rendering of diffusion curves with triangles. *IEEE Computer Graphics and Applications*, 32(4):68–78, 2012.
- [45] Romain Prévost, Wojciech Jarosz, and Olga Sorkine-Hornung. A vectorial framework for ray traced diffusion curves. *Comput. Graph. Forum*, 34(1):253–264, 2015.
- [46] Yipeng Qin, Xiaoguang Han, Hongchuan Yu, Yizhou Yu, and Jianjun Zhang. Fast and exact discrete geodesic computation based on triangle-oriented wavefront propagation. *ACM Transactions on Graphics (TOG)*, 35(4):125, 2016.
- [47] Mauro R. Ruggeri, Tal Darom, Dietmar Saupe, and Nahum Kiryati. Approximating geodesics on point set surfaces. In *Symposium on Point-Based Graphics*, 2006.
- [48] Yevgeny Schreiber. An optimal-time algorithm for shortest paths on realistic polyhedra. *Discrete & Computational Geometry*, 43(1):21–53, 2010.
- [49] Yevgeny Schreiber and Micha Sharir. An optimal-time algorithm for shortest paths on a convex polytope in three dimensions. *Discrete & Computational Geometry*, 39(1-3):500–579, 2008.
- [50] James A. Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of National Academy of Sciences*, 93:1591–1595, 1996.
- [51] James A. Sethian and Alexander Vladimirsky. Fast methods for the Eikonal and related Hamilton-Jacobi equations on unstructured meshes. *Proceedings of National Academy of Sciences*, 97:5699–5703, 2000.
- [52] Justin Solomon, Raif Rustamov, Leonidas Guibas, and Adrian Butscher. Earth mover’s distances on discrete surfaces. *ACM Trans. Graph.*, 33(4):67:1–67:12, 2014.
- [53] Alon Spira and Ron Kimmel. An efficient solution to the Eikonal equation on parametric manifolds. *Interfaces and Free Boundaries*, 6(4):315–327, 2004.

- [54] Xin Sun, Guofu Xie, Yue Dong, Stephen Lin, Weiwei Xu, Wencheng Wang, Xin Tong, and Baining Guo. Diffusion curve textures for resolution independent texture mapping. *ACM Trans. Graph.*, 31(4):74:1–74:9, 2012.
- [55] Vitaly Surazhsky, Tatiana Surazhsky, Danil Kirsanov, Steven J. Gortler, and Hugues Hoppe. Fast exact and approximate geodesics on meshes. *ACM Trans. Graph.*, 24(3):553–560, 2005.
- [56] Kenshi Takayama, Olga Sorkine, Andrew Nealen, and Takeo Igarashi. Volumetric modeling with diffusion surfaces. *ACM Trans. Graph.*, 29(6):180, 2010.
- [57] John N. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Transactions on Automatic Control*, 40(9):1528–1538, 1995.
- [58] Marc Van Kreveld. Efficient methods for isoline extraction from a tin. *International Journal of Geographical Information Systems*, 10(5):523–540, 1996.
- [59] Kasturi R Varadarajan and Pankaj K Agarwal. Approximating shortest paths on a non-convex polyhedron. *SIAM Journal on Computing*, 30(4):1321–1340, 2000.
- [60] Charlie C.L. Wang. Cybertape: an interactive measurement tool on polyhedral surface. *Computers & Graphics*, 28(5):731 – 745, 2004.
- [61] Ruimin Wang, Zhouwang Yang, Ligang Liu, Jiansong Deng, and Falai Chen. Decoupling noise and features via weighted ℓ_1 -analysis compressed sensing. *ACM Trans. Graph.*, 33(2):18:1–18:12, April 2014.
- [62] Xiaoning Wang, Zheng Fang, Jiajun Wu, Shi-Qing Xin, and Ying He. Discrete geodesic graph (DGG) for computing geodesic distances on polyhedral surfaces. *Computer-Aided Geometric Design*, 52:262–284, 2017.
- [63] Ofir Weber, Yohai S. Devir, Alexander M. Bronstein, Michael M. Bronstein, and Ron Kimmel. Parallel algorithms for approximation of distance maps on parametric surfaces. *ACM Trans. Graph.*, 27(4):104:1–104:16, 2008.
- [64] Shi-Qing Xin, Dao TP Quynh, Xiang Ying, and Ying He. A global algorithm to compute defect-tolerant geodesic distance. In *SIGGRAPH Asia 2012 Technical Briefs*, pages 23:1–23:4, 2012.

- [65] Shi-Qing Xin and Guo-Jin Wang. Improving Chen and Han’s algorithm on the discrete geodesic problem. *ACM Transactions on Graphics*, 28(4):104, 2009.
- [66] Shi-Qing Xin, Xiang Ying, and Ying He. Constant-time all-pairs geodesic distance query on triangle meshes. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 31–38, 2012.
- [67] Chun-Xu Xu, Tuanfeng Y. Wang, Yong-Jin Liu, Ligang Liu, and Ying He. Fast wavefront propagation (FWP) for computing exact geodesic distances on meshes. *IEEE Transactions on Visualization and Computer Graphics*, 21(7):822–834, 2015.
- [68] Xiang Ying, Xiaoning Wang, and Ying He. Saddle vertex graph (SVG): A novel solution to the discrete geodesic problem. *ACM Transactions on Graphics*, 32(6):170:1–12, 2013.
- [69] Xiang Ying, Shi-Qing Xin, and Ying He. Parallel Chen-Han (PCH) algorithm for discrete geodesics. *ACM Transactions on Graphics*, 33(1):9:1–9:11, 2014.
- [70] Hongkai Zhao. Parallel implementations of the fast sweeping method. *Journal of Computational Mathematics*, 25(4):421–429, 2007.