Parallel computation of discrete geodesics and its applications

Le, Tien Hung

2017

Le, T. H. (2017). Parallel computation of discrete geodesics and its applications. Doctoral thesis, Nanyang Technological University, Singapore.

<http://hdl.handle.net/10356/72861>

<https://doi.org/10.32657/10356/72861>

NANYANG TECHNOLOGICAL UNIVERSITY

Parallel Computation of Discrete Geodesics and Its Applications

A thesis submitted
for a degree of Doctor of Philosophy
by

Le Tien Hung

School of Computer Science and Engineering
Nanyang Technological University

Jan, 2017

Abstract

Computing geodesics on meshes is a classical problem in computational and differential geometry. It measures the length of the local shortest path with minimum curvature along the surface between two points. Previous studies have documented many important properties of geodesics, such as being a distance metric, locally isotropic and invariant to isometric transformation. With those properties, geodesic distance could be considered as a more general definition of Euclidean distance. Thus, it plays an important role in constructing various algorithms, solving many real-world problems and developing countless applications in a wide range of fields, including computer graphic, digital geometric processing, computer vision and image processing, etc.

Although computing discrete geodesics has been widely studied, there are still many difficult barriers in finding of geodesic distances and paths, including high computational cost, dependence on quality of the input mesh and scalability for other input data types with less connectivity information. Effectively computing discrete geodesics overcoming those issues can be a key to open larger doors to solving a lot of related problems.

Traditionally, the proposed geodesic algorithms have tended to focus on geodesics between vertices rather than between arbitrary points on input mesh surfaces. Geodesics between two arbitrary points could be computed in a simple but inaccurate way using interpolation or in other way using re-triangulation which always is very expensive and can globally change the topology of the input meshes. The lack of this ability in computing geodesics can diminish its area of applications.

This thesis aims at developing efficient and flexible geodesic algorithms to address the challenges of computing geodesics at vertices/surface points on large real-world models with high time performance for interactive applications. Our contributions include:

The first contribution is the analysis and development of an algorithm for computing approximate geodesics on triangle meshes. We adapt the unfolding strategy and the label correcting algorithm to the well-known "Saddle Vertex Graph" algorithm to take their advantages for approximating geodesic distances at vertices/points with good time performance. The combination can work smoothly with very large-scale models and it is promising for developing interactive applications.

Based on the classical "Mitchell, Mount, and Papadimitriou" (**MMP**) algorithm, we introduced its improved version (**IMMP**) as the second contribution for discrete geodesic computation with high performance on triangle meshes. The first major improvement is about designing an efficient approach for windows clipping, which can be used extensively in other geodesic algorithms. In our approach, solving quadratic equations and computing Euclidean distance are required only in special cases, thus the number of calling expensive square-root function is significantly reduced. In addition, we use vertices instead of windows as the primitive in a priority queue with a simple window merging/updating to significantly cut down the overhead of the windows propagation management. We also proposed parallel improved MMP algorithm (**PIMMP**) as a parallelize version of IMMP by dividing the IMMP sequential algorithm into four phases, front node selection, window list propagation, window list merging and vertex update. We use two separating buffers for income windows on each edge so there is no data dependence or conflicts in each phase and the operations within each phase can be carried out in a CPU-base parallel fashion. The PIMMP algorithm is able to propagate a large number of windows simultaneously and independently, thus its performance is highly improved on real-world models.

As the third contribution, we apply our discrete geodesic algorithm into a multi-class shape distribution application. It allows users to control different properties of randomly distributed shapes, including their sampling rate, direction field and density. Furthermore, the anisotropic point-based and curve-base exponential mapping techniques are supported, which are used to draw shapes at points or follow curves. In our application, multiple complex 2D objects can be distributed on 3D surface without conflict. With the high performance of our geodesic algorithm, our application can work interactively, even when input meshes are very large.

Acknowledgments

I would like to express my gratitude to all those who gave me the possibility to complete this thesis.

First and foremost I wish to thank my main-advisor, Dr. He Ying, for his consistent support, patient guidance and encouragement. It is a great honor and luck for me to have a chance to working with such nice and creative advisor like him. His supervision and teaching truly rise my inside love with geometry and help me learn how the dry topics could be so much beautiful and fun for study. I would add special thanks to Dr. Jun Luo, my co-advisor for his valuable advices.

I also wish to thank all my friends of the Geometry Modeling Group at GameLab, in particular Mr. Wang Xiaoning, Mr. Ying Xiang, Mr. Zhang Minqi, Ms. Sun Qian, Mr. Wang Dayong, Mr. Fang Zheng, Mr. Hou Fei, and many others for years of company and friendship. This report would not be possible in this form without their support and collaboration.

Last, but certainly not least, I would like to thank my parents, my wife and my little baby. Despite the physical distance separating us, they have always been one of the biggest motivations which remind me to try hard and harder at work.

Publications

Published:

Xiaoning Wang*, **Tien Hung Le***, Qian Sun, Xiang Ying, Ying He. User controllable anisotropic shape distribution on 3D meshes. *Computational Visual Media*, Vol. 2, No. 4, pp. 305-319, 2016.

To be submitted

Tien Hung Le, Ying He. Parallel MMP (PMMP) Algorithm for Discrete Geodesics.

*Joint 1st authors

Contents

Abstract	i
Acknowledgments	iii
Publications	iv
List of Figures	vii
List of Tables	xii
1 Introduction	1
1.1 Motivation	1
1.1.1 Important role of computing discrete geodesics and its applications	1
1.1.2 Limitations of Geodesic Algorithms	5
1.2 Objectives	6
1.3 Thesis Organization	9
2 Related Work	10
2.1 Geodesic on smooth surfaces	10
2.2 Discrete geodesic computation	13
2.3 Exponential Map	25
2.4 Sampling and Shape Distribution	27
3 Efficient Computation of Discrete Geodesics	30
3.1 Combination of SVG and GTU	31
3.2 Computing geodesic paths	33
3.3 Improving the SVG Performance	34
3.4 Experimental results	38

4	Improved MMP algorithms	41
4.1	Preliminary	41
4.2	Window clipping schemes	43
4.3	Window pruning	52
4.4	Window propagation management	52
4.5	Parallelized MMP algorithm	56
4.6	Experimental Results	59
5	Application	67
5.1	Geodesic Polar Coordinates	68
5.2	Parallel Distribution of Anisotropic Shapes	71
5.2.1	User Input	72
5.2.2	Single-Class Shape Distribution	73
5.2.3	Multi-Class Shape Distribution	77
5.3	Experimental Results	79
6	Conclusions and Future Work	86
6.1	Conclusions	86
6.2	Limitations	88
6.3	Future work	89
	References	92

List of Figures

2.1	Geodesic curve on manifolds.	11
2.2	Propagation of a candidate interval.	14
2.3	The principle “one angle, one split”. The green windows on the edge (v_1, v_2) will propagate forward, and the red one must stop.	16
2.4	Geodesic information is propagated by the normal direction of the boundary.	17
2.5	The Fast Marching Approach.	18
2.6	Outline of the heat method.	20
2.7	The GTU algorithm’s pipeline. Figures are borrowed from [75].	21
2.8	Three different types of edges in the SVG. The saddle vertices are drawn in red, the non-saddle vertices are drawn in green. The exact geodesic paths from a saddle vertex to its saddle neighbors are showed in the Figure (a), from a non-saddle vertex to its saddle neighbors are showed in the Figure (b), from a non-saddle vertex to its non-saddle neighbors are showed in the Figure (c). Figures are borrowed from [78].	23
2.9	Exponential map creates a geodesic polar coordinate system on the surface. The Figure (a) shows the polar coordinate system on the tangent plane $\mathcal{T}_p$. The Figure (b) is the polar coordinate system on the surface. Figure (c) shows the exponential map naturally induces a texture mapping.	25
2.10	Some results of the Texture Brush interface. Figures are borrowed from [61].	27

3.1	Computing single-source all-destination geodesic distances on a low-resolution mesh. (a) Let p be the source point (a mesh vertex) and q be a point inside the triangle $\triangle v_1 v_2 v_3$. (b) Using the saddle vertex graph, we can accurately compute the geodesic distances from p to any mesh vertex. Once geodesic distances are defined on each vertex, we can easily estimate the geodesic distance $d(p, q)$ using linear interpolation. However, the interpolated result has very low accuracy compared with the exact result in (a), since distance is not a linear function. (c) The geodesic triangle unfolding method can significantly improve the accuracy. With known geodesic distances $d(p, v_i), i = 1, 2, 3$, we can unfold the geodesic triangles, $\triangle p v_1 v_2$, $\triangle p v_2 v_3$, and $\triangle p v_3 v_1$ onto R^2 . The geodesic distance $d(p, q)$ is approximately the minimum of the three Euclidean distances $d(p_i, q), i = 1, 2, 3$. (d) Texture mapping shows the high-quality result provided by the GTU method.	31
3.2	Computing the approximated geodesic path between two arbitrary points p and q by unfolding	34
3.3	Visual comparison of the accuracy between the exact geodesic distances in Figures (a), (c) and our approximate geodesic distances in the Figures (b), (d). The source is an arbitrary point. For the input model <i>Bunny</i> with 10k faces and $K = 10$, the relative error is 3.1% and the running time is 0.007s. With $K = 25$, the relative error is 0.55% and the running time is 0.008s.	39
3.4	Visual comparison of the accuracy. Our approximate geodesic path (red) is very close to the exact geodesic path (blue).	40
4.1	Three different types of vertices in the upper row and the unfolding of the vertex star, i.e. the set of faces meeting at a vertex, in the lower row.	42
4.2	The MMP window structure. The geodesic distance at point $p(x_p, 0)$ inside the window w equals to the sum of d_s and $d(s, p)$	43

4.3	Window propagation over the edge $\overline{v_0v_1}$: (a) only one potential window w_1 delivered onto the edge $\overline{v_0v_2}$; (b) two potential windows w_1, w_2 delivered onto the edges $\overline{v_0v_2}$ and $\overline{v_1v_2}$; (c) Special case of window propagation where v_1 is a saddle vertex. There are three potential windows w_1, w_2, w_3 delivered onto the edges $\overline{v_0v_2}$ and $\overline{v_1v_2}$	44
4.4	MMP window clipping: (a) w_1 is the old window on the edge $\overline{v_0v_1}$ and w_2 is the new potential window delivered onto $\overline{v_0v_1}$; (b) One of the splitting cases at point p	44
4.5	Window clipping of w_1 and w_2 with $d_{s_2} = d_{s_1}, x_{s_2} \neq x_{s_1}, y_{s_2} \neq y_{s_1}$	48
4.6	Window clipping of w_1 and w_2 with $d_{s_2} \neq d_{s_1}, x_{s_2} = x_{s_1}, y_{s_2} \neq y_{s_1}$	49
4.7	Window clipping of w_1 and w_2 with $d_{s_2} \neq d_{s_1}, x_{s_2} \neq x_{s_1}, y_{s_2} = y_{s_1}$	50
4.8	Window clipping of w_1 and w_2 with $d_{s_2} \neq d_{s_1}, x_{s_2} \neq x_{s_1}, y_{s_2} \neq y_{s_1}$	51
4.9	After intersecting of w_{income} with w_i , the red part of w_{income} is used as the new income window for intersecting with w_j	53
4.10	Separating points of the windows w_1 and w_2 on the edge (v_1, v_2)	54
4.11	The triangle-oriented window propagation in the VTP algorithm. Figures are borrowed from [55].	55
4.12	During the propagation around v_1 , window list wl_{12} on the edge (v_1, v_2) propagates to the edges (v_3, v_1) in (a), (b) and (v_2, v_3) in (c), (d). New income windows from wl_{12} on the edge (v_3, v_1) are stored in wl^{start} (a) or wl^{end} (b). Only the forward window lists on the edge (v_3, v_1) are pushed into the FIFO queue. New income windows from wl_{12} on the edge (v_2, v_3) are stored in wl^{end} (c) or wl^{start} (d) and will not be merged til the end of the list propagation around v_1	58
4.13	Testing the size factor f . Choosing a small f still fully utilizes the parallel propagating process because we always control the minimum number of running threads. On the other hand, choosing a large f would require many useless window list checking/merging and make the vertex queue remain longer, resulting in slow performance and high memory usage. We found the optimal value for f is around 0.5, with which the PIMMP archives its best execution time and memory usage.	61

4.14	Running out IMMP algorithm on anisotropic models.	63
4.15	Running out IMMP algorithm on high genus models.	64
5.1	Given a 3D triangle mesh M and a set of 2D anisotropic shapes, user first specifies a vector field for directional control (see the top-right inset) and a scalar field for density control (see the bottom-left inset). Then our method automatically distributes the given 2D shapes onto M , satisfying the user-specified directional and density constraints. It takes only 2.9 seconds to distribute 4 classes of objects on this 400K-face Fertility model. Timing was measured on a quad-core CPU at 2.66GHz.	68
5.2	Geodesic polar coordinates. (a) Our LC-enhanced SVG method is able to compute the geodesic distances for curved sources. Each offset curve is then parameterized by arc-length. (b) The offset distance δ together with the normalized arc-length $\hat{s}$ uniquely determine a point on the 3D surface. The geodesic polar coordinate system, formed by the 2-tuple $(\delta, \hat{s})$, defines a bijective map between $\mathbb{R}^2$ and the patch on 3D surface.	69
5.3	Polar coordinate on iso-curves.	70
5.4	Comparison with extended exponential map [61]. Our result (b) has comparable quality as Sun <i>et al.</i> 's method [61] (a) and much better time performance.	71
5.5	Algorithmic pipeline of single-class shape distribution. (a) User specifies a scalar field to control the shape density and the locations of singularities and their indices. (b) We adopt Crane <i>et al.</i> 's method [13] to compute the global vector field which controls the shape orientation. (c) Our method automatically determines the locations, sizes and orientations of the 2D objects. Each blue curve-segment represents the central line of one object. (d) All distributed objects are guaranteed to be collision-free.	72
5.6	For long shapes, we divide them to several parts and test the intersection separately.	74

5.7	Comparison with Poisson disk sampling. Row 1: Poisson disk sampling considers each sample as a disk and does not allow overlapping disks (a). There are 103 stars distributed in (b) Row 2: Our method allows a more dense distribution than Poisson disk sampling, as long as the stars are not overlapping. It randomly distributes 221 stars in (c) and greedy distributes 333 stars in (d).	75
5.8	The circumcircle of shape (a) and the minimum safe distance between two shapes (red line) (b).	76
5.9	Single-class shape distribution on the Kitten model with different density values and textures. We control the density values f by using a brush with the center at the left eye and the user-defined radius r . The default value of f is 1.0. For the region inside the brush we have $f = \min\{0.25, d/r\}$ where d is the geodesic distance to the center. . . .	80
5.10	Multi-class shape distribution on a bumpy sphere with 50K faces. . . .	80
5.11	Our method works well on the Bimba model with various resolutions: 30K faces in (a) and (b), 150K faces in (c) and (d).	81
5.12	Distortion of the local parameterization on model Bimba with low resolution 7k faces in (a). Our results become smoother when resolution increases to 30k faces in (b) and 150k faces in (c).	82
5.13	Comparison to [35]. The 2D Eagle shape is highly concave, so a simple bounding ellipse cannot capture its geometric features. As a result, requiring non-overlapping ellipses is very pessimistic. Li <i>et al.</i> 's method [35] can distribute only 115 eagles (left). Obviously, the distribution is not maximal. Our method produces a much more dense distribution, containing 224 eagles (right), since it allows overlapping ellipses as long as the adjacent eagles do not collide.	83
5.14	More results of multi-class sharp distribution on Gargoyle (a) and Sculpture (b) models.	84

List of Tables

3.1	Time and space complexities. n : # of vertices; m : # of sample points specified by user; K : the size of geodesic disk containing the direct geodesic paths; D : model-dependent-but-resolution-insensitive constant; SSAD: single-source-all-destination.	37
3.2	Performance of the classical Dijkstra, BFS, THRESH and SLF-LLL algorithms on SSAD task. Columns 3 to 6 report the running time (in seconds) of the mentioned algorithms on the same geodesic graphs which are constructed with $K = 50$. F is the number of faces on a model. . . .	39
3.3	Accuracy of the SVG-GTU combination with the label correcting algorithm. S is the size(the number of crossed faces) of a path and ε is the mean value of the relative error of the path's length. In the first case, the input meshes are triangulated using Delaunay triangulation. In the second case, these meshes are re-triangulated to get more obtuse triangles. F is the number of faces on a model.	40
4.1	Comparison of the old window intersecting scheme in MMP algorithm with our new window intersecting scheme. The columns 3 and 5 show the running time t required for intersecting process in MMP. Columns 4 and 6 show the calling time n of the function <i>sqr</i> t in intersecting process. . . .	51
4.2	Performance comparison of our IMMP and PIMMP algorithm with other exact algorithms on running time, maximum memory usage and total maximum number of active windows. F is the number of faces on a model.	62

4.3	Performance comparison of our IMMP and PIMMP algorithm with other exact algorithms on running time, maximum memory usage and total maximum number of active windows on anisotropic models. F is the number of faces on a model.	62
5.1	Mesh complexity and running time performance of sharp distribution process. T_1 and T_4 : running time (in seconds) on a quad-core CPU using a single core and all cores, respectively.	79

Chapter 1

Introduction

1.1 Motivation

1.1.1 Important role of computing discrete geodesics and its applications

Geodesic is a fundamental concept in differential geometry and it is well studied in the last years. Roughly speaking, the notion “geodesic” can be defined as a generalization version of a straight line from 2D/3D space to curved space. The correlation between them is quite intuitive: while a straight line minimizes the distance between two points on a plane, a geodesic minimizes the distance between two points on any surface with a Riemannian metric. In other words, a geodesic path is the local shortest path between two points and its length is called geodesic distance. There are also other ways to describe the notion “geodesic” in different areas.

Geodesic has some important properties. First, it is a distance metric which means that geodesic from a point x to other point y is always non-negative, symmetric and satisfies the triangle inequality. Second, geodesic from a source point s is locally isotropic and it gradually increases or decreases, depends on the radius of s ' neighborhood. Third,

geodesic is intrinsic, thus the distances and paths are isometry invariant, in other words, they do not change under isometric transformations of the surface.

The discrete geodesic problem divides itself into some special sub-problems such as "single source, single destination" [26, 30, 53, 38, 54, 72], "single source, multiple destinations" [60, 44, 9, 1, 31, 48, 62] and "all-pair geodesic" [64, 1, 24, 23, 2, 3]. These sub-problems are also closely related to each other.

Many geodesic algorithms have been proposed in the literature and could be classified into two groups as exact and approximate geodesic algorithms. The exact geodesic algorithms, included MMP [44], Chen and Han(CH) [9] and its improved version [73], strictly guarantee the resulting accuracy of geodesic distances and paths. Ying *et al.* [79] presented a paralleled variant of the CH algorithm using GPU, which allows us to compute geodesics on real-world models in a more efficient way. The approximate geodesic algorithms, such as FMM [58], heat method [14] and SVG [78], focus more on reducing the computing speed. They locally use various types of related information on input mesh, such as gradient vector field and local shortest paths, to approximate the geodesic distances and paths with better performance in comparison with the exact geodesic algorithms. Especially, the SVG algorithm is very promising since its result is highly accurate and its data structure has mature parallelable property.

There is another way to classify geodesic algorithms based on their information propagation mechanisms. The "windows propagation" mechanism is usually used in the exact geodesic algorithms. Geodesics are globally computed using propagation of windows which are created by partitioning edges [44, 9, 73]. The windows with encoded geodesic information are propagated in wavefront order from selected sources over vertices and edges on input surfaces. Due to the extremely large number of generated windows, these algorithms have very high computation cost and hard to be applied onto

real-world large models. The PDE mechanism is used most in the approximate geodesic algorithms [58, 14]. The propagation of the local geodesic information is encountered in solving Eikonal equation which is non-linear and can be approximated into other much simpler partial differential equations. By using modern numerical methods, these equations can be solved very fast, even in linear time, thus the time performance is a big advantage of this mechanism. Despite its efficiency, however, the PDE mechanism usually suffers from numerical instability if the input models have rich details because the approximation techniques applied in these details are inaccurate.

More details of these geodesic algorithms will be discussed in Chapter 2. See [43, 3, 45] for other surveys.

As mentioned above, geodesics are intrinsic on surfaces. That important property lets geodesic distance be considered as a general extension of Euclidean distance and used to perform basic tasks as computing geodesic offsets [74], texture mapping [83], surface parameterization [82], surface sampling [7], remeshing [51], computing Voronoi diagram [65], non-rigid registration [28], shape editing [81], shape segmentation [56], etc...

One of direct applications of geodesics is computing geodesic offsets on triangle meshes. Xin *et al.*[74] applied the ICH algorithm to obtain exact, efficient and robust results. The input meshes can be large scale models with complicated geometry and topology. Another example is proposed in [32], which uses the FMM algorithm to compute Voronoi diagram and offset curves on a manifold. Xu *et al.*[76] and Wang *et al.*[65] adopted the SVG algorithm with its advantages to compute centroidal Voronoi tessellation (CVT) on arbitrary triangle meshes.

In surface parameterization, Zhou *et al.*[82] applied spectral analysis on the matrix of

geodesics between pairs of surface points. This way provided two simultaneous solutions: partitioning the model into charts and parameterizing those charts.

In texture mapping, Zigelman *et al.*[83] applied the FMM algorithm to compute geodesic distances between every pairs of points. The approximated geodesics are used to design a surface flattening by classical Multi-dimensional Scaling to achieve accurate measurements that characterizes the geometry of the surface with a reasonable computational complexity. Thus, it leads to the texture mapping with minimal distortion, by preserving the local and global structure of the texture. By using the ICH algorithm to compute geodesic offsets, Sun *et al.*[61] developed a very interesting texturing interface called Texture Brush, which supports anisotropic texture mapping. Users can easily specify such local parameterization with a free-form stroke on the surface. A set of interactive 3D painting operations was also proposed.

In surface sampling, Liu *et al.*[35] presented the anisotropic blue noise sampling by extending dart throwing and relaxation for isotropic blue noise sampling. Their method can be applied to 3D surfaces to compute high-quality global parameterization. Although the performance can be improved as Wei's suggestions in [68], their implementation is difficult to be completed, since anisotropy poses a challenge in sampling and domain partitioning. Bowers *et al.*[7] presented a parallel algorithm for direct sampling on arbitrary surfaces by using grid cell structure in R^3 to manage the samples. The geodesic distances are approximated as the length of curves, which are interpolated by normals. Bower's approach is extrinsic and depends on the embedding space. Ying *et al.*[80] proposed an intrinsic and highly efficient algorithm for parallel Poisson disk sampling on arbitrary surfaces. The ICH algorithm is used to compute the geodesic distances for priority initiation and conflict detection. The generated Poisson disks are uniformly and randomly distributed on arbitrary surfaces in R^n without bias.

1.1.2 Limitations of Geodesic Algorithms

As mentioned above, various geodesic algorithms have been presented for different goals since 1980s. In spite of their popularity, computing geodesics still has several drawbacks which can seriously diminish its area of applications.

In designing a geodesic algorithm, time performance has to be taken into account as a major factor. Working with large amount of data is an inherent problem of computing geodesics, especially in exact geodesic algorithms. With the fast growth of modern 3D scanning techniques and devices in the last years, recent developments of geodesic algorithms have heightened in how to work with large real-world models. When input meshes come with high resolutions, the number of vertices, edges, faces and other related information need to be processed is increased rapidly. Therefore, geodesic algorithms must to adopt appropriate computation strategies to guarantee the numerical stability when handling encoded data with large size. Unfortunately, this requirement is not always satisfied. For such explosions of the input data, geodesic algorithms are required to be able to use a huge capacity of hardware resource to work well. Not all of the proposed geodesic algorithms can take advantages of parallel and multi-threads programming on new computing systems, which have become used more popularly. In addition, there will be very rich details in those input meshes, for which it is hard or expensive to obtain robust and accurate data approximations. In this case, the approximate geodesic algorithms provide unreliable results, or sometimes they do not work because their prerequisite is not satisfied. Therefore, performance improvement is always a main target of researchers when developing a new geodesic algorithm, no matter it is exact or approximate.

The lack of computing geodesics between arbitrary points on input surfaces is another problem for us to solve. Using interpolation is a simple and straightforward method for

computation of geodesic between two arbitrary points. Assume that the exact geodesics between vertices around two arbitrary points p and q are already computed. A linear or average interpolation on these exact geodesic distances can quickly result in the geodesic distance between p and q . Obviously this approximate result is inaccurate. Another solution is applying re-triangulation for p, q , the vertices around them and then computing geodesics between the new vertices at p, q . Although the result can be accurate in this case, re-triangulation is always expensive and should be avoided. Moreover, it can globally change the topology of the input meshes and lead to other unexpected problems, especially for the geodesic algorithms which use precomputed data. Thus, it is highly desirable to develop a general algorithm for computation of geodesics to extend their area of applications.

With technological advances over the last years, model hardware devices allow real-time acquisition of data and delivering dynamic virtual scenes to user. It emerged a novel trend within digital mesh processing, which concentrates on time-varying geometry and spans various applications from scientific problems, such as dynamic volume data processing, physical simulations up to pure entertainment, such as character animation. Most of the current geodesic algorithms compute geodesics in a global manner and/or they don't have the flexibility or require high processing cost for efficiently handling dynamic 3D models. This gives us a great need to design new algorithms for dynamic geodesic computation.

1.2 Objectives

From the problems mentioned above, the objective of this thesis is to develop general and efficient algorithm for computing geodesics on large input meshes. The resolution of input meshes can be high with millions vertices and faces. It is much larger than the

input meshes used in other proposed geodesic algorithms. Our algorithm can compute the geodesics between arbitrary points (not necessary to be vertices) with controllable accuracy and its domain can be extended on dynamic meshes. There are many rooms in our algorithm for paralleling, thus its performance is high and it is suitable to be used in interactive applications.

Efficient Computation of Discrete Geodesics: To overcome the challenge of fast computing geodesic between arbitrary points on real-world models, we adopt the graph-based strategy from Saddle Vertex Graph algorithm with unfolding approach instead of calling expensive mesh triangulation to reconstruct the precomputed geodesic graph or inaccurate linear interpolating. Using the label-correcting method to solve the finding shortest path problem on graph, we show that the time performance is significantly improved. It gives us opportunity to develop various interactive applications on large real-world models.

Improved MMP algorithm: The existing algorithms compute exact geodesic distances using windows propagation mechanism. The windows, which are created by partitioning edges, with encoded geodesic information are propagated in wavefront order from selected sources over vertices and edges on input surfaces. During the propagation, delivered windows can intersect with others to create new potential windows. The clipping operator in classic exact algorithms like MMP is expensive because it requires to solve quadratic equation and compute Euclidean distances. We propose a new window clipping, in which computing Euclidean distances is avoided and solving quadratic equation is required in some special cases only. Therefore, we can reduce the computing cost by calling the expensive square-root function in minimum times. Not only in MMP, our window clipping can be adopted extensively in other exact geodesic algorithms. With an efficient window list merging and using simple front node as primitives for propa-

gation management, our IMMP algorithm performs remarkably on large-scale models with good running time and low memory usage.

The original MMP algorithm lacks a parallel solution because the windows are maintained in a priority order, which means that only one window is processed at a time. In this thesis we propose PIMMP algorithm as a parallelized version of IMMP algorithm to reduce the high computational cost and improve time performance. We divide MMP algorithm into

four phases, front node selection, window list propagation, window list merging, vertex updating and use two separate buffers for income windows on each window list to guarantee that there is no data dependence or conflict in each phase. It means in each wavefront, there is a large number of windows that can be propagated and all these phases can be parallelized on multithreading CPU.

Applications: To demonstrate the flexibility, high performance and accuracy of our discrete geodesic algorithm, we apply it to create a shape distribution application which support anisotropic point-based and curve-based exponential mappings. Users are allowed to control several properties of the distributed shapes, such as direction field, density, etc. Unlike the existing approaches, which usually assume the 2D objects are isotropic and of simple geometry, our algorithm applies for complex 2D objects and can guarantee the distribution is conflict-free, which is a critical constraint in many applications. Instead of using global surface parameterization, our method only computes local geodesic polar coordinates, thus it has a natural parallel structure and it depends only on the mesh metric instead of the embedding space. We evaluate our algorithm on real-world models with non-trivial topology and observe promising results.

1.3 Thesis Organization

The report is organized as follows. Chapter 2 gives some brief surveys about related works in discrete geodesics and its applications in sampling, shape distribution, dynamic mesh processing. Chapter 3 presents our implementation of efficient computation of discrete geodesics. Chapter 4 describes the major improvements for our algorithms, including IMMP algorithm and its parallelized version PIMMP algorithm. Our anisotropic shape distribution application and its experimental results are presented in Chapter 5. Finally, Chapter 6 concludes the thesis and points out some interesting directions for future work.

Chapter 2

Related Work

This chapter surveys some related work in geodesic problem and its applications.

2.1 Geodesic on smooth surfaces

In Riemannian geometry, a geodesic curve are defined not the same as the shortest path between two points on a surface, although the two concepts are closely related to each other. The difference between them is that geodesic curve has only locally minimum length and it is parameterized with "constant velocity" (and less geodesic curvature K_g). In general relativity, geodesic curves describe the paths of point particles, which move under the influence of gravity alone.

In this section, we consider a smooth surface S embedded into the three-dimensional Euclidean space, $S \subset \mathbb{R}^3$. Surface S can be locally described though a parametric function:

$$\varphi : x(x_1, x_2) \subset \Omega \rightarrow \tilde{x} = \varphi(x) \subset S$$

where Ω is an open domain of $\mathbb{R}^2$.

The tangent plane $T_{\tilde{x}}$ at a surface point $\tilde{x} = \varphi(x)$ is spanned as:

$$D\varphi(x) = \left(\frac{\partial \varphi}{\partial x_1}(x), \frac{\partial \varphi}{\partial x_2}(x) \right) \in \mathbb{R}^{3 \times 2}$$

A curve $\tilde{\gamma}: [0, 1] \rightarrow S$ traced on the surface has a length computed as:

$$L(\tilde{\gamma}) = \int_0^1 \|\tilde{\gamma}'(t)\|_{T_{\tilde{\gamma}(t)}} dt,$$

where $\tilde{\gamma}'(t) \in T_{\tilde{\gamma}(t)} \subset \mathbb{R}^3$ is the derivative vector, which lies in the embedding space $\mathbb{R}^3$, and is in fact a vector belonging to the 2D tangent plane $\mathcal{T}_{\tilde{\gamma}(t)}$ to the surface at point $\tilde{x}$ (see Figure 2.1).

Figure 2.1: Geodesic curve on manifolds.

The tensor T_x is defined as:

$$\forall x \in \Omega, T_x = \sqrt{W(x)} I_\varphi(x)$$

where $I_\varphi(x) \in \mathbb{R}^{2 \times 2}$ is the first fundamental form of S :

$$I_\varphi(x) = (D\varphi(x))^T D\varphi(x) = \left(\left\langle \frac{\partial \varphi}{\partial x_i}(x), \frac{\partial \varphi}{\partial x_j}(x) \right\rangle \right)_{1 \leq i, j \leq 2}$$

and where, given a positive symmetric matrix $A = (A_{i,j})_{1 \leq i,j \leq 2} \in \mathbb{R}^{2 \times 2}$, we define its associated norm: $\|u\|_A^2 = \langle u, u \rangle_A$ with:

$$\langle u, v \rangle_A = \langle Au, v \rangle = \sum_{1 \leq i,j \leq 2} A_{i,j} u_i v_j$$

The geodesic curve $\tilde{\gamma}^*$ is defined as the local shortest curve joining two points $\tilde{x}_s, \tilde{x}_e \in S$, which means that $\tilde{\gamma}^*$ is everywhere locally a distance minimizer.

$$\tilde{\gamma}^* = \arg \min_{\tilde{\gamma} \in S(\tilde{x}_s, \tilde{x}_e)} L(\tilde{\gamma})$$

And the geodesic distance between $\tilde{x}_s$ and $\tilde{x}_e$ is defined as the length of $\tilde{\gamma}^*$.

The geodesic distance from a point x to a set of points S in Ω is defined as the distance between x and its closest point in Ω :

$$U_S(x) = \min_{y \in S} d(x, y)$$

where $U_S(x)$ is called as a distance map. A geodesic curve γ^* from $x \in \Omega$ and S is a curve y^* in $\mathcal{P}(x, y)$ with $y \in S$ and $L(\gamma^*) = U_S(x)$.

The geodesic distance map U_S is differentiable and it satisfies the Eikonal equation:

$$\|\nabla U_S(x)\|_{T_x^{-1}} = 1$$

with boundary conditions $U_S(x) = 0$ on S

A function u is a solution of the Eikonal equation if for any $\phi \in C^1(\Omega)$ and for all $x_0 \in \Omega \setminus S$ we have:

$$\|\nabla \phi(x_0)\|_{T_{x_0}^{-1}} = 1$$

Crandall *et al.* [12] proved that if S is a compact set and $x \rightarrow T_x$ is a continuous mapping, then the geodesic distance map U_S is the unique solution of the equation:

$$\forall x \in \Omega, \|\nabla U_S(x)\|_{T_x^{-1}} = 1$$

where $\forall x \in S, U_S(x) = 0$.

In the Euclidean case, whole solution for $S = \{x_s\}$ is $U_{x_s}(x) = \|x - x_s\|$

More detail about the geodesic on smooth surfaces could be found in [52], [54].

2.2 Discrete geodesic computation

The discrete surface is a polyhedral model of a smooth surface, reconstructed from the given discrete data set, decomposed commonly in triangles or in quadrilaterals and other polygons. The computation of geodesic paths and distances on triangle meshes is a common operation in many computer graphics applications. There are several practical algorithms presented for computing such geodesics from a source point to one or all other points efficiently.

The MMP algorithm [44] is one of the first practical algorithms for exact geodesic computation on general polyhedral surfaces. Geodesic information is encoded into a simple data structure called “window” or “candidate interval”, which is a subsegment of an edge with associated information and could be propagated in the wavefront order. The MMP operates in the precise spirit of Dijkstra’s shortest path algorithm: once a source is selected, the “windows” are propagated from it to the rest of the vertices. When a new geodesic information is received at a vertex, it is compared with the old one. Afterwards, any change is updated and propagated further. Each edge may have $O(n)$ windows, therefore the total number of windows can be $O(n^2)$, where n is the number of edges of the surface. A $\log n$ priority queue is maintained to point to propagated windows for tracking. The MMP algorithm has space complexity $O(n^2)$ and time complexity $O(n^2 \log n)$. The shortest path from a source to a destination can be found in time $O(k + \log n)$, where k is the number of faces crossed by the shortest path.

Figure 2.2: Propagation of a candidate interval.

Surazhsky *et al.* [62] observed that the worst-case running time is overly pessimistic in the MMP algorithm. For typical meshes, the average number of windows on edges is $O(\sqrt{n})$, hence in practice the MMP algorithm runs much faster than suggested by worst-case analysis in [44]. To reduce the window complexity, they proposed an effective window intersection approach: before the moment when a candidate window is propagated, it could be merged with the existing adjacent windows on the same edge. The merging is only performed if some conditions can be satisfied to guarantee the visibility region of the new window. After the merging, the old windows are replaced by the new one in the event queue, therefore the number of windows on edges decreases significantly. As a result, the geodesic distances are computed efficiently with bounded error.

The degenerate cases that appear frequently in the MMP algorithm with real-world data were also analyzed in [37]. In interval propagation of windows, degeneracies commonly occur in two circumstances: geometric intersection and geodesic discontinuities

(because of the numerical errors in floating point computations). The authors then presented effective techniques to handle all the degenerate cases consistently and correctly, which make the implementation of the MMP algorithm in [44, 62] is very robust. Liu *et al.* [36] investigated the analytic structure of the iso-contours and bisectors of the discrete geodesic distance field on triangular meshes and proposed an efficient and practical MMP-based algorithm to explicitly compute the geodesic Voronoi diagrams. The experimental results are insensitive to triangle shape and density on triangulated surfaces.

The CH algorithm [9] adopts a different strategy of window organization to compute exact geodesics. Instead of the priority queue as in MMP, it uses a hierarchical structure called binary tree to control the order of window propagation. Two types of windows are used to encode the geodesic information: pseudo-source windows at vertices and interval windows on edges. A window, which is propagating, can have different number of children, depends on its type. According to the relationships between parent and children windows, a sequence tree with at most n levels can be built to contain the propagated windows, where n is the total number of faces on polyhedral surface. The sequence tree stores only the leaf nodes and the branch nodes while throwing away all the one-child interval windows, leading to linear space complexity $O(n)$. To avoid exponential explosion of the sequence tree, the children windows are selected/abandoned by principle "one angle, one split". In Figure 2.3, there are two windows w_1, w_2 entering into triangle $\triangle v_0 v_1 v_2$ from the edge (v_1, v_2) . Both of w_1, w_2 cover the vertex v_0 , hence four interval windows are created on the edge (v_1, v_2) as children of w_1 and w_2 . Principle of "one angle, one split" implies that only three of the children windows can provide shortest paths. Subsequently, only three selected interval windows will be propagated forward through triangle $\triangle v_0 v_1 v_2$. The CH algorithm determines the shortest paths in "single source, all vertices" problem with time complexity $O(n^2)$ and it works on polyhedral surfaces which not need to be convex.

Figure 2.3: The principle “one angle, one split”. The green windows on the edge (v_1, v_2) will propagate forward, and the red one must stop.

Surazhsky *et al.* [62] compared the performance of the CH and the MMP algorithms. Based on the provided experimental evidence, very interesting conclusion was made: the CH algorithm with time complexity $O(n^2)$ runs many times slower, in practice, than the MMP algorithm with time complexity $O(n^2 \log n)$. Through extensive experiments, Xin and Wang [73] showed that in many examples more than 99% of windows created by the CH algorithm do not contribute to define a shortest path at all. Therefore, they eliminated such useless windows through an effective window filter at little cost and avoid computation of further propagation for them. Thanks for the smaller total number of propagated windows, both time performance and space complexity of the original CH algorithm were significantly improved. In addition, they maintained a priority queue for Dijkstra’s algorithm, which does further improve the CH algorithm greatly in both time and space. As a result, their improved CH algorithm (ICH), in spite of an asymptotic time complexity $O(n^2 \log n)$, generally outperforms the original CH and MMP algorithms.

Recently, Ying *et al.* [79] presented PCH, a parallel version of CH algorithm, which decomposes the sequential CH algorithm into four stages, namely window selection,

window propagation, data organization and event processing so that there is no data dependency or conflicts within each stage. Thus there is a large number of windows to be propagated simultaneously and independently in each iteration. The GPU-based implementation of the PCH algorithm is one order of magnitude faster in comparison with the ICH algorithm.

The fast marching method (FMM) [58] is a grid-based numerical computational approach for tracking evolution of monotonically advancing interfaces by solving the non-linear Eikonal equation:

$$|\nabla u(x)| = F(x), x \in \Omega \quad (2.1)$$

The equation 2.1 subjects to $u = g(x)$ on surface Γ in Ω , where $F(x) > 0$ and Ω is a domain in R^2 or R^3 . Function $u(x)$ is supplied as the travel time or distance from the source, and the right-hand side $F(x)$ is specified as the speed function defined on Ω . In the special cases, $F(x)$ is given as a constant, usually equals to 1. Typically, the Eikonal equation describes the evolution of a closed curve or area in the normal direction at points on the boundary (see Figure 2.4).

Figure 2.4: Geodesic information is propagated by the normal direction of the boundary.

One of the main difficulties in solving the Eikonal equation is the non-differentiability of the appropriate weak solution. Based on adaptive updating scheme and ordered data structures on regular grid, the FMM is one of the most efficient strategies to solve Eikonal equation and it is optimal in the worst case. Function $F(x)$ can be set with a unchanged sign, so the front always moves forward or backward and crosses each point on grid only one time. This gives capacity to simplify the problem and decrease solving time.

Figure 2.5: The Fast Marching Approach.

The Fast Marching Method is closely related to Dijkstra's algorithm, which can compute the shortest path on the grid: the front propagates outward on the grid from an initial source and only one grid point is marched at a time during the propagating process. The most proper grid point for the fast marching event is systematically located by a fast heapsort technique. Each grid point is marched and updated only once, so when a grid point is marked as evaluated, the front never backtracks over it again. Suppose the distance is computed on an $n \times n$ grid, then n^2 time value computations will be made with total $\log n$ operations on heap to process all the grid cells. Therefore, the complexity of the FMM is $O(n^2 \log n)$.

Although the FMM only approximates the geodesic distances, it is still widely used in various areas thank for its simplicity and optimal time complexity. The working domain of the FMM can be extended on triangle meshes [31], implicit surfaces [40], point clouds [41] and broken meshes [8]. Weber *et al.* [67] presented a parallel version of FMM on geometry images and observed improvements in execution up to four orders of magnitude better than the conventional version. The FMM can be used to handle the special cases of motion with sign changing velocities [70] or anisotropic and more general Hamilton-Jacobi equations [59]. However, a good triangulation without obtuse triangles is required to preserve the monotonicity of the wavefront; otherwise, a complicated unfolding procedure must be applied.

The heat method [14] is a gradient-based approach with a completely different strategy for geodesic approximation, since in the heat method the distance field d and its gradient ∇d are decoupled. The heat method uses Varadhan's formula, which reveals the relationship between the heat kernel and the geodesic distance field. The heat method consists of three basic steps. It first integrates the heat flow u as

$$\dot{u} = \Delta u$$

for a fixed time t on the whole input mesh, where Δu is the negative-semidefinite Laplace-Beltrami operator on u . In the second step, the normalized gradient field X is evaluated by solving equation

$$X = -\nabla u / |\nabla u|$$

since the heat method asks only for the direction of the gradient ∇u and ignores the magnitude. In the last step, the geodesic distance field is recovered as the closest scalar potential Φ by minimizing $\int_M |\nabla d - X|^2$, which is equivalent to solve a Poisson equation

Figure 2.6: Outline of the heat method.

$\Delta\Phi = \nabla \cdot X$ over the entire mesh M . The overall procedure of the heat method is outlined in Figure 2.6.

The heat method is easy to be implemented, and can approximate geodesic distance field on a wide range of geometric domains including points cloud. Furthermore, Crane *et al.* [14] showed that if the Laplacian matrix is pre-factored, the heat method can run in near-linear time, which significantly outperforms the FMM in terms of time performance. The smooth nature of the heat kernel means that the geodesic distances should be approximated smoothly; however, the geodesic distance field in heat method is not smooth due to the existence of the ridge point. The heat method requires numerical solver, and is unstable for models with rich details: the resulting distance fields show distortions and there is not optimized heat integration time step for tuning.

The geodesic triangle unfolding (GTU) method, proposed by [75], flatten the curved geodesic triangle onto R^2 and then uses Euclidean distance to approximate geodesic distance. The algorithm's pipeline contains two steps, the preprocessing step and the query step, as show in the Figure 2.7.

In the preprocessing step, the farthest point sampling method [51] and the ICH algorithm [73] are adopted for finding isotropic sampling and geodesic triangulation. After that, the exact geodesic distances between any pair of the samples and the relative position of the vertices to the samples are saved into a file for later use. In the next step,

Figure 2.7: The GTU algorithm's pipeline. Figures are borrowed from [75].

some simple unfolding operations are presented to compute the geodesic distance between two points, which not need to be vertices. The resulting geodesic distances are approximated with an upper bound of error, which is smaller than 0.5% for the input models in the implemented experiments and can be controlled through the total number of samples. It is a trade-off between the accuracy and running time for preprocessing step. It is required to find enough sampling points to get a smooth geodesic distance field. How many sampling points is enough - the answer depends on the requirements of applications.

The most advantage of the GTU method is that it can compute the approximate geodesic distance between any two arbitrary point on the input mesh's surface. However, the preprocessing step takes $O(mn^2 \log n)$ time, where n is the number of vertices of the input mesh and m is number of samples on the input surface, which is expensive and can diminish the range of the GTU's applications.

The Saddle Vertex Graph (SVG) is one of the most efficient approximate geodesic algorithms, which was presented by Ying and his colleagues [78]. Consider a triangle mesh $M = (V, E, F)$, where V , E and F are the set of vertices, edges and faces, respectively. A vertex v is called *saddle* if the sum of its interior angles exceeds the value 2π . Mitchell *et al.* [44] proved that a discrete geodesic path goes through an alternating sequence of vertices and edges such that the unfolded image of the path along any edge sequence is a straight line segment and the angle of the path passing through a vertex is greater than or equal to π .

Let $\gamma(p, q)$ denote a geodesic path between two vertices $p \in M$ and $q \in M$. The geodesic path $\gamma(p, q)$ is called direct, if it does not pass through any saddle vertices. Otherwise, it is indirect and can be partitioned into several segments, each of which is direct. The SVG method takes advantage of the local nature of mesh structure to break down the

global geodesic problem into significantly smaller sub-problems. These sub-problems can be solved using exact geodesic methods like ICH and their results can be reused later. The local geodesic information is encoded as nodes in a sparse undirected graph, called SVG. The SVG algorithm's pipeline contains two steps, which are summarized below.

In the first step, the SVG with direct geodesic paths will be constructed. Since it is very expensive to build the exact SVG with all direct geodesic paths, an approximate SVG with controlling parameters is more preferred. Given a vertex $v \in V$ and a geodesic disk $\odot(v, R)$ with center v and radius R , which contains no more than K vertices inside. Then the direct geodesic paths within $\odot(v, R)$ are computed and set as edges of the SVG (see Figure 2.8). The parameter K can be used to control the accuracy, time and space complexity of SVG construction.

Figure 2.8: Three different types of edges in the SVG. The saddle vertices are drawn in red, the non-saddle vertices are drawn in green. The exact geodesic paths from a saddle vertex to its saddle neighbors are showed in the Figure (a), from a non-saddle vertex to its saddle neighbors are showed in the Figure (b), from a non-saddle vertex to its non-saddle neighbors are showed in the Figure (c). Figures are borrowed from [78].

There are two important propositions about SVG must be noted. First, the SVG of a connected triangle mesh is connected. It means there is always at least one continuous path between any two notes in SVG. Although computation of exact geodesic paths is

expensive, the required time and space for the SVG construction can be reduced by setting a big enough value to K . Moreover, the whole process can be parallelized on CPU/GPU. Second, a shortest geodesic path between two vertices on the input mesh M is a shortest path between the two appropriate nodes in the SVG associated with M . Therefore computing geodesic paths on the input mesh M is equivalent to finding the corresponding shortest paths in SVG, which can be done much more efficiently using shortest path algorithms, such as Dijkstra's algorithm [15].

The SVG method is very promising. Clearly, it out-performs over the methods listed above: the local geodesic sub-problems are insensitive to the mesh resolution and tessellation, the shortest path algorithm's speed on SVG is much faster in comparison with windows propagation's speed. The SVG method does not use any numerical solver, and it is always numerically stable. By using the controlling parameter K , the SVG method has big advantages in term of flexibility and scalability in comparison to the FMM or the MMP algorithms. However, computing the direct geodesic paths in the preprocessing step still is the bottleneck of the SVG method. In other side, it computes geodesic distances between only vertices of the input meshes. The nodes in SVG do not contain direction information and the triangle unfolding process for computing geodesic paths is very expensive. All these cons make SVG not flexible and efficient enough for interactive applications with large models.

In contrast to the CH and MMP algorithms, Qin *et al.* [55] proposed the **VTP** which adopts triangles instead of windows as the primitive for window propagation. In each iteration, it propagates a collection of windows from one triangle edge to its two opposite edges. Using an exhaustive list of rules, VTP can effectively identify redundant windows, hereby significantly reducing computational cost and memory usage at later stages. After each propagating iteration, old and new windows on each triangle edge

are merged by two order-free merging rules. The time complexity of VTP algorithm is $O(n^2)$ and it out-performs all of the exact geodesic algorithms listed above when running in CPU environment.

2.3 Exponential Map

The discussed above methods are improved in trend of decrease running time and increase resulting accuracy in computing of geodesic distances and paths. It could be a fundamental key for solving exponential map problems in digital geometry processing and gives opportunity to develop countless interesting applications.

Exponential mapping is another problem, which is strongly related to geodesic distances. The map is defined in a geodesic polar coordinate system on curved surfaces. Let $p \in S$ be an arbitrary point on a smooth surface S . The exponential map at p is a map from the tangent plane $\mathcal{T}_p$ at p to p 's local neighborhood.

Figure 2.9: Exponential map creates a geodesic polar coordinate system on the surface. The Figure (a) shows the polar coordinate system on the tangent plane $\mathcal{T}_p$. The Figure (b) is the polar coordinate system on the surface. Figure (c) shows the exponential map naturally induces a texture mapping.

The exponential map at point p takes geodesic paths starting at p to straight rays emanating from the origin point of the tangent plane $\mathcal{T}_p$. Conversely, for any unit vector $v \in \mathcal{T}_p$,

there exists a unique geodesic g_v parameterized by arc length such that $g_v(0) = p$ and $g_v'(0) = v$. In Figure 2.9, exponential map builds a geodesic polar coordinate system around a point on the curved surface.

The exponential map has been used in various graphics applications, such as decal compositing [57], texture mapping [61], Poisson disk sampling [80], etc. Schmidt *et al.* [57] proposed an $O(n \log n)$ algorithm to approximate the exponential map, which requires only a single additional step in Dijkstra's shortest path algorithm. The point-based exponential map approximation is interactively computed on an arbitrary point on surfaces. It allows the texture to be stable even under mesh deformation and surface animation. However, the input meshes or surfaces must not be large for good performance. By using ICH algorithm to compute geodesic offset of curves for local parameterization, Sun *et al.* [61] developed very interesting texturing interface called Texture Brush, which supports anisotropic texture mapping. Users can easily specify such local parameterization with a free-form stroke on the surface (see Figure 2.10). A set of interactive operations, which are mainly based on 3D painting metaphor, were also proposed thanks to the good performance of the ICH algorithm. However, each user-specified stroke has to be simple enough, i.e. without self-intersection to avoid high distortion in mapping area. The low efficacy with large curves or input models is another limitation of the Texture Brush framework.

Jin *et al.* [29] presented a conformal surface parameterization technique using Ricci flow for both global and local areas. In their paper, a Riemannian metric of a surface (edge lengths for a mesh) is computed such that the surface can be flattened onto the plane using the metric. First the discrete Euclidean Ricci flow is applied to determine a flat metric that induces zero Gaussian curvature for all interior vertices. The all the vertices are embedded onto the plane with flat metric to obtain a conformal parameterization of

the input mesh. By solving the Yamabe equation with the Ricci flow method, Wang *et al.* [66] proposed an intrinsic and stable technique to conformally parameterize a brain surface via a mapping to a multi-hole disk without singularities.

Figure 2.10: Some results of the Texture Brush interface. Figures are borrowed from [61].

2.4 Sampling and Shape Distribution

The widely used sampling techniques include blue noise [11], point repulsion [63], centroid Voronoi tessellation (CVT) [16], stratified sampling [47], spectral sampling [50], and polyominoes [49], etc.

Among them, Poisson disk sampling is popular due to its nice spatial and spectral properties. Dart throwing [11] is a popular technique to generate isotropic blue noise samples due to its simplicity. However, it is time consuming since a large number of trials are discarded due to conflicts. Many techniques have been proposed to improve the performance of dart throwing, including effective spatial data structures [42, 17, 21, 18], kernel density model [19], and tile-based approaches [10, 34, 33, 49]. Wei [68] presented the phase group algorithm which subdivides the sample domain into grid cells and drawing samples concurrently from multiple cells that are sufficiently far apart to avoid conflicts.

Thus the sampling process can run in multiple-threading mode on GPU with constant cost per sample and constant number of computation passes for a target number of samples. Wei’s algorithm also works in arbitrary dimension, and allows adaptive sampling from a user-specified importance field. However, the performance drops quickly in 4D and higher dimensional cases or when the importance field is un-uniform. The number of the samples generated by this algorithm is not guaranteed to be maximal and it is difficult to be controlled. Li *et al.* [35] generalized the isotropic blue noise sampling to the anisotropic setting. Using approaches based on warping and sphere sampling to extend Fourier spectrum analysis for adaptive and/or anisotropic samples, their technique can produce uniform placement for both isotropic and anisotropic texture patterns on surfaces with efficient computation. Although the output quality is high, the controlling number of the generated samples in this method is still a unresolved issue. Bowers *et al.* [7] parallelized dart throwing algorithm that generates high-quality surface samples followed a user-specified radius field. Using the spectral mesh basis derived from the discrete mesh Laplacian operator, the analyzing of the spectral quality of surface samples works directly on arbitrary manifold surfaces without requiring mesh parameterization. The algorithm can be extended for adaptive surface sampling, but it doesn’t run at interactive rates and is not suitable for spatially non-uniform sampling patterns. Wei [69] introduced multi-class blue noise sampling where each individual class and their union exhibit blue noise properties. In this algorithm, the number of generated samples can be controlled manually. However, the class priority can not be distributed explicitly and the time performance is slow, especially when the number of input classes increases.

Besides point based sampling, distributing 2D objects has a wide range of graphic applications. For example, line segment distributions are used in rendering applications, such as motion blur, defocus blur and scattering media [61], and rectangle distributions

are used to simulate decorative mosaics [25, 4]. Feng [20] generated non-overlapping ellipses with blue noise characteristic and the samples' size and density match a given anisotropic metric. Sun *et al.* [61] proposed a frequency analysis of line segment sampling, which can be generalized to arbitrary non-point shapes with requiring of the local retriangulating process around the input lines.

In contrast to the 2D counterpart, shape distribution on 3D surfaces remains largely unexplored. Some of the above-mentioned algorithms can be extended to work on 3D surfaces, but they do not support multiple classes of anisotropic shapes [7] or require the expensive global parameterization [35].

Chapter 3

Efficient Computation of Discrete Geodesics

Both the GTU method and the SVG method are graph-theoretic algorithms: the former forms a dense graph with $O(m^2 + n)$ edges, where m is the number of samples (specified by user), and the latter forms a sparse graph with $O(Dn)$ edges, where D is a model-dependent-but-resolution-independent constant ($D \ll n$). Each approach has its own merits and limitations. The SVG method can compute highly accurate geodesic distances between any pair of mesh vertices. However, many applications require the distances between arbitrary points of the input mesh. As Figure 3.1 shows, linearly interpolating vertex distances produces poor results on meshes with large and/or irregular triangles, since distance is a non-linear function. On the other hand, the GTU method can efficiently compute the geodesic

In this chapter, we first show that the SVG and GTU methods can be naturally combined so that we can take advantage of the merits of both and avoid their limitations. We then adopt a label correcting method to improve the performance of shortest path computation in SVG.

3.1 Combination of SVG and GTU

(3.1.a)

(3.1.b)

(3.1.c)

(3.1.d)

Figure 3.1: Computing single-source all-destination geodesic distances on a low-resolution mesh. (a) Let p be the source point (a mesh vertex) and q be a point inside the triangle $\triangle v_1 v_2 v_3$. (b) Using the saddle vertex graph, we can accurately compute the geodesic distances from p to any mesh vertex. Once geodesic distances are defined on each vertex, we can easily estimate the geodesic distance $d(p, q)$ using linear interpolation. However, the interpolated result has very low accuracy compared with the exact result in (a), since distance is not a linear function. (c) The geodesic triangle unfolding method can significantly improve the accuracy. With known geodesic distances $d(p, v_i), i = 1, 2, 3$, we can unfold the geodesic triangles, $\triangle p v_1 v_2$, $\triangle p v_2 v_3$, and $\triangle p v_3 v_1$ onto R^2 . The geodesic distance $d(p, q)$ is approximately the minimum of the three Euclidean distances $d(p_i, q), i = 1, 2, 3$. (d) Texture mapping shows the high-quality result provided by the GTU method.

Consider a triangle mesh $\mathcal{M} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$, where $\mathcal{V}$, $\mathcal{E}$ and $\mathcal{F}$ are the set of vertices, edges and faces, respectively. A vertex v is called *saddle* if the sum of its interior angles exceeds 2π . Mitchell *et al.* [44] proved that a discrete geodesic path cannot pass through a spherical vertex unless it is an endpoint or a boundary point and the unfolded image of the path along any edge sequence is a straight line segment.

The geodesic path $\gamma(v_i, v_j)$ between two vertices $v_i, v_j \in \mathcal{V}$ is called direct, if it does not pass through any saddle vertices. Otherwise, it can be partitioned into several segments so that each segment is direct. Let $\Gamma = \{\gamma(v_i, v_j) | \gamma(v_i, v_j) \text{ is direct}, \forall v_i, v_j \in \mathcal{V}\}$ denote the set of all direct geodesic paths on $\mathcal{M}$. Then the saddle vertex graph associated to mesh $\mathcal{M}$ is an undirected graph $\mathcal{G} = (\mathcal{V}, \Gamma)$. All the existing exact geodesic algorithms, such as the MMP algorithm [44], the ICH algorithm [73], and most recently, the FWP-enhanced algorithm [77], can be used to construct SVG. Ying *et al.* [78] observed that it is *not* necessary to compute *all* direct geodesic paths, in fact, a small subset of Γ can lead to a pretty good result. Therefore, they suggested a simple-yet-effective heuristic to control the SVG size using a parameter K : for a vertex v , only the direct geodesic paths within a geodesic disk containing K or less vertices are considered. They observed that a typical $K \in [50, 200]$ produces geodesic distances of accuracy $10^{-3} \sim 10^{-4}$, which are good for most applications.

The SVG $(\mathcal{V}, \Gamma)$ allows us to compute the geodesic distance between any pair of mesh vertices. To combine SVG and GTU, we take all mesh vertices as the samples, i.e., $m = n$. This strategy has three advantages: First, we totally avoid the construction of geodesic triangulation on $\mathcal{M}$, since each $f \in \mathcal{F}$ is a geodesic triangle. Second, we do not need to explicitly compute and store the *entire* dense weighted graph for the GTU method, since the SVG method allows us to compute the geodesic distance between any pair of samples (i.e., vertices) on the fly. Third, as pointed out by Xin *et al.* [75], the

larger the value of m , the higher the accuracy of the computed geodesic distance. The best accuracy of the GTU method is obtained by setting $m = n$.

Now we are ready to compute the geodesic distance between arbitrary surface points $p, q \in \mathcal{M}$. To ease presentation, we first address the simple case as in Figure 3.1 when one point, say $p \in \mathcal{V}$, is a vertex, and the other $q \notin \mathcal{V}$ is not. Let $\triangle v_1 v_2 v_3$ be the triangle containing q . To compute the geodesic distance $d(p, q)$, we first solve the "single source, multiple destinations" problem on the SVG graph to compute geodesic distances $d(p, v_i), i = 1, 2, 3$. Note that these geodesic distances together with three mesh edges $v_1 v_2, v_2 v_3, v_3 v_1$ can form three geodesic triangles $\triangle p v_1 v_2, \triangle p v_2 v_3$ and $\triangle p v_3 v_1$. After unfolding these triangles onto $\mathbb{R}^2$, we can obtain three images of p , namely, p_1, p_2 and p_3 . Finally, the geodesic distance $d(p, q)$ is approximated by the minimal value of three Euclidean distances $d(p_i, q), i = 1, 2, 3$.

The general case $p, q \notin \mathcal{V}$ can be solved by unfolding both of the triangles containing p and q , respectively. Readers can refer to [75] for the details.

3.2 Computing geodesic paths

Other approximate algorithms, e.g. the FMM, the heat method, computes the geodesic path by tracing the gradient direction of the distance function. The accuracy of the discrete gradient direction field strongly depends on the quality of the input meshes because the distance function is non-linear. For input mesh with rich details or non-optimal triangulation, the approximate result could be very inaccurate. By using the triangle unfolding approach to compute geodesic paths, our method is insensitive with obtuse triangles.

When point q is successfully projected onto plan (v_1^p, v_2^p, v_3^p) , we can find the unfolded intersecting point I' , and then, the intersecting point I . In the next step, the new trian-

Figure 3.2: Computing the approximated geodesic path between two arbitrary points p and q by unfolding

gle, which shares the intersecting edge with the triangle (v_1^q, v_2^q, v_3^q) , will be unfolded. The unfolding process continues till the moment the new intersecting point falls inside the triangle (v_1^p, v_2^p, v_3^p) [see Figure 3.2]. It is required to run SSMD version of SVG algorithm only three times for whole back-tracing process.

3.3 Improving the SVG Performance

SVG naturally links the discrete geodesic problem on polyhedral surfaces and the shortest path problem on graphs, since computing the geodesic distance between p and q is equivalent to finding the shortest path on the corresponding saddle vertex graph. Dijkstra's algorithm [15] is a widely used technique for computing the shortest path. In the following, we review some fundamental concepts of the Dijkstra's algorithm and its improvements.

To compute the shortest paths from a single node, say n_1 , to all of the other nodes in graph $\mathcal{G}$, the Dijkstra's algorithm maintains a label vector $(d_1, d_2, \dots, d_{|V|})$ and a set of

nodes $\mathcal{C}$, called the candidate list, starting with $d_1 = 0$, $d_i = \infty$ for $i \neq 1$, $\mathcal{C} = \{v_1\}$. The Dijkstra's algorithm iteratively processes the nodes from the candidate list $\mathcal{C}$ and terminates when $\mathcal{C}$ is empty. Upon termination, each label d_i is the shortest distance from the source v_1 to node v_i .

In each iteration, the Dijkstra's algorithm takes the node with the smallest label in the candidate list $\mathcal{C}$ to update distances. Different data structures like binary heap, Fibonacci heap could be used to compute the minimal label node from $\mathcal{C}$. Since each node enters and exits $\mathcal{C}$ exactly once, Dijkstra's algorithm takes totally $|V|$ iterations. Methods that do not follow this node selection policy are called *label correcting*, or **LC** for short. These methods maintain $\mathcal{C}$ as a queue with multiple entrances, from which nodes can be inserted or extracted in constant time $O(1)$.

We adopt the Small Label to the Front (SLF) scheme [5] for node insertion and the Large Label Last (LLL) scheme [6] for extraction: when a node v_j enters the queue $\mathcal{C}$, its label d_j is compared with the label d_i of the top node v_i of $\mathcal{C}$. If $d_j \leq d_i$, node v_j will be entered at the top of $\mathcal{C}$; otherwise v_j will be entered at the bottom of $\mathcal{C}$. Node v_i keeps staying at the top of $\mathcal{C}$ while its label is less than a threshold; otherwise it will be sent to the bottom of $\mathcal{C}$. The threshold is usually set as the mean label value of all the nodes in $\mathcal{C}$.

SLF queue insertion strategy:

Let v_i be the top node of C and let v_j be a node that enters C
 if $d_{v_i} \leq d_{v_j}$ then enter v_j at the top of C
 else enter v_j at the bottom of C

LLL node selection strategy:

Let v_i be the top node of C and let $s = \frac{\sum_{v_j \in C} d_{v_j}}{\|C\|}$
 if $d_{v_i} > s$ then move v_i at the bottom of C .
 Repeat until a node v_i such that $d_{v_i} \leq s$ is found and is removed from C

The Dijkstra's algorithm performs at most one iteration per node and requires some extra overhead (e.g, extracting the node with minimum label) per iteration. The label correcting methods, in contrast, takes more iterations than the Dijkstra algorithm, but the overhead per iteration is smaller.

Algorithm 1 Combination of GTU and SVG

Input: A triangle mesh $M = (V, E, F)$, the source vertex p , the parameter K for SVG and the surface point q inside triangle (v_1, v_2, v_3) on M .

Output: The geodesic distance between p and q .

Geodesic graph $G \leftarrow$ Generate SVG graph with the input parameter K

Geodesic distances $d(p, v_1), d(p, v_2), d(p, v_3) \leftarrow$ Run SLF-LLL algorithm on G with the source vertex p

Point $p_{12} \leftarrow$ intersecting point of two circles $(v_1, d(p, v_1))$ and $(v_2, d(p, v_2))$ on the plan (v_1, v_2, v_3)

Point $p_{23} \leftarrow$ intersecting point of two circles $(v_2, d(p, v_2))$ and $(v_3, d(p, v_3))$ on the plan (v_1, v_2, v_3)

Point $p_{31} \leftarrow$ intersecting point of two circles $(v_3, d(p, v_3))$ and $(v_1, d(p, v_1))$ on the plan (v_1, v_2, v_3)

Point $p' = \{p_{min} \in \{p_{12}, p_{23}, p_{31}\} | d(p_{min}, q) = \min\{d(q, p_{12}), d(q, p_{23}), d(q, p_{31})\}\}$

return $d(p', q)$

It is worth noting that the LC-enhanced SVG has an empirical $O(Dn)$ time complexity to compute the single-source-all-destination geodesic distances, since the graph has $O(Dn)$ edges and the overhead per iteration is $O(1)$.

Breadth-First-Search (BFS) [46] is another algorithm for traversing over the geodesic graph G . It uses a FIFO queue instead of stack or priority queue for front vertices and its check the visiting status of a vertex before enqueueing it into the front queue. The time complexity can be defined as $O(|V| + |E|)$ where $O(|E|)$ can vary from $O(1)$ to $O(|V|^2)$ depending on how sparse the geodesic graph G is. Glover *et al.* [22] presented a new polynomially bounded assignment method, which is called as the threshold method (THRESH). During the traversing over G , the queue C is partitioned in two disjoint

queues C_1 and C_2 . At each iteration, the top vertex from C_1 is popped out for checking/updating the distance at its neighboring vertices. A new vertex is added to bottom of C_1 or C_2 depending on whether its distance exceeds or does not exceed a certain threshold parameter, respectively. We showed the time performance of the listed algorithms on some models with different resolutions in Table 3.2 for an overall comparison.

Algorithm 2 Serial Breadth First Search (BFS) algorithm

Input: A triangle mesh $M = (V, E, F)$, the source vertex p on M and the geodesic graph G generated to M
Output: The geodesic distance field d at vertices on M .

```

 $d_p \leftarrow 0$ 
For each vertex  $v_i \in M - p$ 
     $d_i \leftarrow \inf$ 
End for

FIFO queue  $C \leftarrow p$ 
While  $C \neq \emptyset$ 
     $v \leftarrow \text{pop the front value of } C$ 
    For each  $v' \in M$  such that  $(v, v') \in G$ 
        If  $d_{v'} == \inf$ 
             $d_{v'} == d_v + \text{length}(v, v')$ 
             $C \leftarrow v'$ 
        End if
    End for
End while

return  $d$ 
    
```

Table 3.1: Time and space complexities. n : # of vertices; m : # of sample points specified by user; K : the size of geodesic disk containing the direct geodesic paths; D : model-dependent-but-resolution-insensitive constant; SSAD: single-source-all-destination.

Method	Pre-computing time	Space	SSAD	Query points
GTU [75]	$O(mn^2 \log n)$	$O(m^2)$	$O(n)$	arbitrary
SVG [78]	$O(nK^2 \log K)$	$O(Dn)$	$O(Dn \log n)$	mesh vertices only
LC-SVG+GUT	$O(nK^2 \log K)$	$O(Dn)$	empirical $O(Dn)$	arbitrary

We compute our approximate geodesic distances $d(p, q)$ with some arbitrary points

picked as sources. The exact geodesic distances $d'(p, q)$ also are computed by ICH method with the same sources after an additional triangulation. In the Figure 3.3 we measure the relative error $|d(p, q) - d'(p, q)|/d'(p, q)$. Clearly, the relative error becomes smaller with the models with higher resolution. The accuracy can be controlled by changing the value of K and the running time increases not too much.

The accuracy of the combination depends on how long the geodesic paths from p to q are or how many faces the paths cross over and on the quality of the triangulation. The relative error is smaller for input models with higher resolution. In Table 3.3 the exact geodesic paths and our approximate geodesic paths with different sizes are compared. They are very close to each other with $K = 50$. These models are also stretched to make the triangles on them more obtuse. The experimental results show that the relative error could increase in the areas which have obtuse angles.

3.4 Experimental results

We evaluated the performance of the Dijkstra's algorithm and the SLF-LLL based label correcting method on several real-world models for comparison. As Table 3.2 shows, the label correcting method can almost improve the performance by reducing the run time to half of the Dijkstra's algorithm with an Intel Xeon quad-core CPU at 2.66GHz. More results from Breadth-First Search algorithm and threshold method are also compared to demonstrate the speed-up factor on the same geodesic graph.

Table 3.2: Performance of the classical Dijkstra, BFS, THRESH and SLF-LLL algorithms on SSAD task. Columns 3 to 6 report the running time (in seconds) of the mentioned algorithms on the same geodesic graphs which are constructed with $K = 50$. F is the number of faces on a model.

Model	$ F $	$t_{Dijkstra}(s)$	$t_{BFS}(s)$	$t_{THRESH}(s)$	$t_{SLF-LLL}(s)$
Bunny	160K	0.05	0.039	0.034	0.025
Sphere	328K	0.086	0.051	0.050	0.041
Fertility	800K	0.286	0.216	0.196	0.145
Kitten	1.1M	0.359	0.270	0.255	0.196
Bimba	1.792M	0.609	0.450	0.451	0.351

(3.3.a)

(3.3.b)

(3.3.c)

(3.3.d)

Figure 3.3: Visual comparison of the accuracy between the exact geodesic distances in Figures (a), (c) and our approximate geodesic distances in the Figures (b), (d). The source is an arbitrary point. For the input model *Bunny* with 10k faces and $K = 10$, the relative error is 3.1% and the running time is 0.007s. With $K = 25$, the relative error is 0.55% and the running time is 0.008s.

Figure 3.4: Visual comparison of the accuracy. Our approximate geodesic path (red) is very close to the exact geodesic path (blue).

Table 3.3: Accuracy of the SVG-GTU combination with the label correcting algorithm. S is the size(the number of crossed faces) of a path and ε is the mean value of the relative error of the path's length. In the first case, the input meshes are triangulated using Delaunay triangulation. In the second case, these meshes are re-triangulated to get more obtuse triangles. F is the number of faces on a model.

Model	$ F $	K	S_1 (faces)	ε_1	S_2 (faces)	ε_2
Armadillo	346K	50	45	0.02%	53	0.02%
Sculpture	400K	50	110	0.09%	148	0.12%
Bunny	640K	50	792	0.02%	871	0.03%
Pegaso	667K	50	139	0.06%	171	0.08%
Fertility	800K	50	126	0.04%	150	0.05%
Kitten	1.1M	50	441	0.02%	510	0.03%
Dragon	1.5M	50	708	0.02%	920	0.04%

Chapter 4

Improved MMP algorithms

In this chapter we propose an Improved MMP (**IMMP**) algorithm in which our efficient window intersecting schemes are combined with the rule "one angle one split" to eliminate a big number of redundant windows. We also adopt the "face-oriented propagation" strategy with a parallelized window propagating process to archive a better time performance.

4.1 Preliminary

Let $M = (V; E; F)$ be a triangle mesh representing an orientable 2-manifold where V , E and F are the vertex, edge and face sets, respectively. Figure 4.1 shows three different types of vertices, which are defined by the difference of 2π and the sum of the angles between adjacent faces adjoining each vertex. For a vertex $v \in V$, the total angle is the sum of interior angles formed between each pair of neighboring edges incident at v . A vertex v is called *spherical* if its total angle is less than 2π , *Euclidean* if the total angle equals 2π , and *saddle* if the total angle is greater than 2π . Mitchell *et al.* [44] proved that there exists a geodesic path from the source s , typically a mesh vertex, to any other point on the surface. They also showed that a geodesic path cannot pass through a spherical

vertex except that it is an end point. If a geodesic path from the source s to destination t passes through one or more saddle vertices, we call the vertex, which is nearest to t , a pseudo source. Intuitively speaking, a pseudo source acts as a relay, which carry the geodesic distance toward the destination.

Figure 4.1: Three different types of vertices in the upper row and the unfolding of the vertex star, i.e. the set of faces meeting at a vertex, in the lower row.

Surazhsky *et al.* [62] used a structure "window" to encode the points whose shortest paths share the same vertex-edge sequence. Windows are generated by partitioning each mesh edge into a set of intervals. All the shortest paths within a window can be encoded locally using a 6-tuple $(x_{start}, x_{end}, x_s, y_s, d_s, \tau)$. The windows are then propagated across faces of the mesh in Dijkstra-like sweep.

In Figure 4.2, the pseudo-source s is crucial to define a window w since the geodesic distance d_p at point p inside w equals to the sum of geodesic distance d_s at s and the straight-line distance between the s and p . Assume that w is on the edge $\overline{v_0 v_1}$, the pseudo-source s and the begin/end points of w are transformed into a 2D coordinate system with the origin at V_0 . Thus the pseudo-source s is defined by its two coordinates x_s and y_s , while $(x_{start}, 0)$ and $(x_{end}, 0)$ are the begin and end points, respectively. The parameter τ shows which direction w follows to enter into the triangle (v_0, v_1, v_2) .

Figure 4.2: The MMP window structure. The geodesic distance at point $p(x_p, 0)$ inside the window w equals to the sum of d_s and $d(s, p)$.

$$d_p = d_s + \sqrt{(x_s - x_p)^2 + (y_s)^2} \quad (4.1)$$

4.2 Window clipping schemes

When a window w propagates through edge $\overline{v_0 v_1}$, it can define new potential windows on the two opposing edges $\overline{v_1 v_2}$, $\overline{v_2 v_0}$. To define these potential windows, we simply unfold the pseudo-source point s onto plan (v_0, v_1, v_2) and then find where two side rays of "unfolded" w intersect with the edges $\overline{v_0 v_2}$ and $\overline{v_1 v_2}$ (see Figure 4.3).

The edges $\overline{v_1 v_2}$ and $\overline{v_2 v_0}$ may already contain propagated windows from different window propagations because there could be more than one wavefront cross over these edges. To guarantee that all stored windows on these edges have minimum geodesic distance and they don't overlap each other, we must intersect these previous windows

Figure 4.3: Window propagation over the edge $\overline{v_0v_1}$: (a) only one potential window w_1 delivered onto the edge $\overline{v_0v_2}$; (b) two potential windows w_1, w_2 delivered onto the edges $\overline{v_0v_2}$ and $\overline{v_1v_2}$; (c) Special case of window propagation where v_1 is a saddle vertex. There are three potential windows w_1, w_2, w_3 delivered onto the edges $\overline{v_0v_2}$ and $\overline{v_1v_2}$.

with the new potential ones, to capture their combined minimum distance field. In other words, we only keep the part of a newly computed potential window that has a smaller geodesic distance than that already associated with points on $\overline{v_0v_2}$ or $\overline{v_1v_2}$.

Figure 4.4: MMP window clipping: (a) w_1 is the old window on the edge $\overline{v_0v_1}$ and w_2 is the new potential window delivered onto $\overline{v_0v_1}$; (b) One of the splitting cases at point p .

Assume that w_1 is the existing window on edge $\overline{v_0v_1}$ and w_2 is the newly created window which intersects with w_1 at interval $\omega = w_1 \cap w_2$. To decide which of the windows w_1 and w_2 defines the minimal distance function for each point in ω , we need to find all the splitting points $p \in \omega$ where the distance functions defined by w_1 and w_2 are equal. In

the 2D coordinate system as shown in Figure 4.4, the shortest distance pass through w_1 to point p can be described as: $\Phi_1(p) = d_{s_1} + \sqrt{(x_{s_1} - x_p)^2 + y_{s_1}^2}$

Similarly, the shortest distance at point p through w_2 is: $\Phi_2(p) = d_{s_2} + \sqrt{(x_{s_2} - x_p)^2 + y_{s_2}^2}$

Let's call $\Phi(x) = \Phi_2(x) - \Phi_1(x) = d_{s_2} - d_{s_1} + \sqrt{(x_{s_2} - x)^2 + y_{s_2}^2} - \sqrt{(x_{s_1} - x)^2 + y_{s_1}^2}$ as the splitting function of the windows w_1, w_2 . The x_p coordinate of a splitting point p must be the solution of the equation $\Phi(x) = 0$ or:

$$d_{s_2} - d_{s_1} + \sqrt{(x_{s_2} - x)^2 + y_{s_2}^2} - \sqrt{(x_{s_1} - x)^2 + y_{s_1}^2} = 0 \quad (4.2)$$

Surazhsky *et al.* [62] showed that the equation above can be reduced to a quadratic and its solution must be in the required range, i.e. $x_p \in \omega$. After that, the interval ω is split into sub-intervals and the Euclidean distances from s_1, s_2 to an arbitrary point inside these sub-intervals are compared to decide w_1 or w_2 will cover these sub-intervals. Although solving a quadratic and computing Euclidean distances in the original MMP algorithm are simple, there are some factors could be considered non-optimal. Firstly, solving quadratic equation is always required to find the splitting points or to check if they exist. And a solution x^* of the quadratic not always satisfies the condition $x^* \in \omega$, thus unnecessary computing operation could happen. The third disadvantage is that the expensive square-root function is called frequently after solving quadratic to compute Euclidean distances to define which windows from w_1 and w_2 is better for covering a sub-interval of ω .

To avoid these disadvantages, we check whether the solutions of Equation 4.2 exist inside ω before solving its correspondence quadratic. With a simple transformation:

$$\Phi(x) = d_{s_2} - d_{s_1} + \frac{2 * (x_{s_1} - x_{s_2}) * x + x_{s_2}^2 + y_{s_2}^2 - x_{s_1}^2 - y_{s_1}^2}{\sqrt{(x_{s_2} - x)^2 + y_{s_2}^2} + \sqrt{(x_{s_1} - x)^2 + y_{s_1}^2}} \quad (4.3)$$

we can compute the limits of $\Phi(x)$:

$$\lim_{x \rightarrow +\infty} \Phi(x) = d_{s_2} - d_{s_1} - (x_{s_2} - x_{s_1}) \quad (4.4)$$

$$\lim_{x \rightarrow -\infty} \Phi(x) = d_{s_2} - d_{s_1} + (x_{s_2} - x_{s_1}) \quad (4.5)$$

and its first order derivative:

$$\Phi'(x) = \frac{A(x) * B(x)}{\Gamma_1(x) * \Gamma_2(x) * [(x - x_{s_2}) * \Gamma_1(x) + (x - x_{s_1}) * \Gamma_2(x)]} \quad (4.6)$$

where:

$$A(x) = (y_{s_1} + y_{s_2}) * x - (x_{s_1} * y_{s_2} + x_{s_2} * y_{s_1}) \quad (4.7)$$

$$B(x) = (y_{s_1} - y_{s_2}) * x + (x_{s_1} * y_{s_2} - x_{s_2} * y_{s_1}) \quad (4.8)$$

$$\Gamma_1(x) = \sqrt{(x_{s_1} - x)^2 + y_{s_1}^2} \quad (4.9)$$

$$\Gamma_2(x) = \sqrt{(x_{s_2} - x)^2 + y_{s_2}^2} \quad (4.10)$$

Because $A(x), B(x)$ are monotonic, it is easily to compute the sign of function $\Phi'(x)$ and local extrema of $\Phi(x)$. From these informations, we can instantly verify the existing of solutions in Equation 4.2 and then solve it only for the cases when at least one of the solutions exist. In old MMP algorithm, after the clipping of w_1 and w_2 , the best window from w_1 and w_2 could be selected to cover a newly created sub-interval by computing Euclidean distances from the pseudo sources of w_1 and w_2 to a random point within the sub-interval. In our intersecting scheme, we always know the sign of $\Phi'(x)$, thus we can monitor the monotonic property of $\Phi(x)$ and then compare $\Phi_1(x)$ and $\Phi_2(x)$ without computing any Euclidean distances.

Let's have a quick analysis for the splitting function $\Phi(x)$.

Case 1: the simplest case when $x_{s_2} = x_{s_1}, y_{s_2} = y_{s_1}$, so $\Phi(x) = d_{s_2} - d_{s_1}$. If $d_{s_2} \geq d_{s_1}$ then $\Phi(x)$ is non-negative and it means w_1 is better than w_2 for covering ω . Otherwise we chose w_2 for the job.

Case 2: with $d_{s_2} = d_{s_1}, x_{s_2} = x_{s_1}, y_{s_2} \neq y_{s_1}$. From Formula 4.3 we have:

$$\Phi(x) = \frac{(y_{s_2} - y_{s_1}) * (y_{s_2} + y_{s_1})}{\Gamma_2(x) + \Gamma_1(x)} \quad (4.11)$$

If $y_{s_2} > y_{s_1}$ then $\Phi(x) < 0$ and we chose w_2 to cover ω . If $y_{s_2} < y_{s_1}$ then w_1 is the better window to pick up.

Case 3: with $d_{s_2} = d_{s_1}, x_{s_2} \neq x_{s_1}, y_{s_2} = y_{s_1}$. The identical solution of Equation 4.2 is $x^* = 0.5 * (x_{s_2} + x_{s_1})$, the first order derivative $\Phi'(x)$ is positive with $x_{s_2} > x_{s_1}$ or negative with $x_{s_2} < x_{s_1}$. From this monotonic property we can easily find out w_1 or w_2 is the better window for covering ω .

Case 4: with $d_{s_2} = d_{s_1}, x_{s_2} \neq x_{s_1}, y_{s_2} \neq y_{s_1}$. There is only one solution of Equation 4.2:

$$x^* = \frac{x_{s_2} + x_{s_1}}{2} + \frac{y_{s_2}^2 - y_{s_1}^2}{2 * (x_{s_2} - x_{s_1})} \quad (4.12)$$

In addition, $\Phi(x)$ has a local extremum at:

$$x_{ext} = \frac{x_{s_1} * y_{s_2} - x_{s_2} * y_{s_1}}{y_{s_2} - y_{s_1}} \quad (4.13)$$

If $y_{s_2} > y_{s_1}$ then $\Phi'(x) < 0$ with $x < x_{ext}$ and $\Phi'(x) > 0$ with $x > x_{ext}$. The sign of $\Phi'(s)$ changes oppositely if $y_{s_2} < y_{s_1}$.

As it is shown in Figure 4.5, w_1 or w_2 could be selected for covering ω by checking whether x^* belongs to ω .

Case 5: with $d_{s_2} \neq d_{s_1}, x_{s_2} = x_{s_1}, y_{s_2} \neq y_{s_1}$. We can simplify:

$$\Phi'(x) = \frac{(x - x_{s_1})(y_{s_1} - y_{s_2})(y_{s_1} + y_{s_2})}{\Gamma_2(x) * \Gamma_1(x) [\Gamma_2(x) + \Gamma_1(x)]} \quad (4.14)$$

to see that $\Phi(x)$ has a local extremum $\Phi(x) = d_{s_2} - d_{s_1} + y_{s_1} - y_{s_2}$.

To solve Equation 4.2, we need to compute the value:

$$\Delta = \frac{[y_{s_1}^2 - y_{s_2}^2 + (d_{s_2} - d_{s_1})^2]^2}{4 * (d_{s_2} - d_{s_1})^2} - y_{s_1}^2 \quad (4.15)$$

Figure 4.5: Window clipping of w_1 and w_2 with $d_{s_2} = d_{s_1}, x_{s_2} \neq x_{s_1}, y_{s_2} \neq y_{s_1}$.

If there are solutions of Equation 4.2 exist, they should be $x_1^* = x_{s_1} - \sqrt{\Delta}$, $x_2^* = x_{s_1} + \sqrt{\Delta}$. Assume ε is the smallest valid size of windows in MMP algorithm. If $y_{s_2} > y_{s_1}$

Figure 4.6: Window clipping of w_1 and w_2 with $d_{s_2} \neq d_{s_1}, x_{s_2} = x_{s_1}, y_{s_2} \neq y_{s_1}$.

(Fig 4.6.a), the first conditions to guarantee that the intersecting points exist and new windows could be validly generated from them are $d_{s_2} - d_{s_1} > 0$, $d_{s_2} - d_{s_1} + y_{s_1} - y_{s_2} < 0$ and $x_2^* - x_1^* > \varepsilon$ or $4 * \Delta > \varepsilon^2$. Furthermore, the distances from d_{s_1} to the intersecting points can be computed directly for new windows generation:

$$d_{s_1, x_{1,2}^*} = \frac{y_{s_1}^2 - y_{s_2}^2}{2 * (d_{s_2} - d_{s_1})} + \frac{d_{s_2} - d_{s_1}}{2} \quad (4.16)$$

$$d_{s_2, x_{1,2}^*} = \frac{y_{s_1}^2 - y_{s_2}^2}{2 * (d_{s_2} - d_{s_1})} - \frac{d_{s_2} - d_{s_1}}{2} \quad (4.17)$$

if $y_{s_2} < y_{s_1}$, the conditions are checked similarly as in Fig 4.6.b.

Case 6: with $d_{s_2} \neq d_{s_1}, x_{s_2} \neq x_{s_1}, y_{s_2} = y_{s_1}$.

Function $\Phi(x)$ has a local extremum $\Phi_{ext} = d_{s_2} - d_{s_1}$ at $x_{ext} = \frac{x_{s_2} + x_{s_1}}{2}$. By computing the value:

$$\Delta = \frac{(d_{s_2} - d_{s_1})^2}{4} + \frac{y_{s_1}^2 * (d_{s_2} - d_{s_1})^2}{(x_{s_2} - x_{s_1})^2 - (d_{s_2} - d_{s_1})^2} \quad (4.18)$$

we have the potential solutions $x_1^* = \frac{x_{s_2} + x_{s_1}}{2} - \sqrt{\Delta}$, $x_2^* = \frac{x_{s_2} + x_{s_1}}{2} + \sqrt{\Delta}$.

Figure 4.7: Window clipping of w_1 and w_2 with $d_{s_2} \neq d_{s_1}, x_{s_2} \neq x_{s_1}, y_{s_2} = y_{s_1}$.

In Figure 4.7.a when $x_{s_2} > x_{s_1}$, Equation 4.2 has maximum one solution x^* if $d_{s_2} - d_{s_1} + x_{s_2} - x_{s_1} > 0 > d_{s_2} - d_{s_1} - (x_{s_2} - x_{s_1})$. Based on the monotonic property of $\Phi(x)$, we can select x^* from x_1^*, x_2^* after checking the sign of Φ_{ext} . Concretely, if $\Phi_{ext} > 0$ then x^* must be greater than $\frac{x_{s_2} + x_{s_1}}{2}$, so it means $x^* = x_2^*$. If $\Phi_{ext} < 0$ then $x^* = x_1^*$.

The conditions for $x_{s_2} < x_{s_1}$ (see Figure 4.7.b) could be analyzed similarly.

Case 7: with $d_{s_2} \neq d_{s_1}, x_{s_2} \neq x_{s_1}, y_{s_2} \neq y_{s_1}$. For this general case (see Figure 4.8), function $\Phi(x)$ has local extrema:

$$\Phi_{ext}(x_{ext}) = d_{s_2} - d_{s_1} + \text{sign}(y_{s_2} - y_{s_1}) * \sqrt{(x_{s_2} - x_{s_1})^2 + (y_{s_2} - y_{s_1})^2} \quad (4.19)$$

at the extremal point $x_{ext} = \frac{x_{s_1} * y_{s_2} - x_{s_2} * y_{s_1}}{y_{s_2} - y_{s_1}}$

In Figure 4.8.a, we need to solve quadratic equation only if $d_{s_2} - d_{s_1} + x_{s_2} - x_{s_1} > 0 > d_{s_2} + d_{s_1} - (x_{s_2} - x_{s_1})$ or $d_{s_2} + d_{s_1} - (x_{s_2} - x_{s_1}) > 0 > d_{s_2} + d_{s_1} - \sqrt{(x_{s_2} - x_{s_1})^2 + (y_{s_2} - y_{s_1})^2}$. Note that the distance between two potential solutions for the last condition must be greater than ε . We can use the inequality $\Delta \geq A^2 * \varepsilon^2$, where $A = (x_{s_2} - x_{s_1})^2 - (d_{s_2} - d_{s_1})^2$, to prevent unnecessary calling square-root function when two solutions are too close to each other. All another cases in Figure 4.8 can be analyzed in the same manner.

 Figure 4.8: Window clipping of w_1 and w_2 with $d_{s_2} \neq d_{s_1}, x_{s_2} \neq x_{s_1}, y_{s_2} \neq y_{s_1}$.

 Table 4.1: Comparison of the old window intersecting scheme in MMP algorithm with our new window intersecting scheme. The columns 3 and 5 show the running time t required for intersecting process in MMP. Columns 4 and 6 show the calling time n of the function sqr t in intersecting process.

Model	$ V $	t_{old} (s)	n_{old} (million)	t_{new} (s)	n_{new} (million)
Bunny	80K	0.43	12.7	0.19	2.1
Pegaso	100K	0.35	10.1	0.19	1.7
Aotu	111k	1.05	30.1	0.53	4.1
Kids	200K	0.80	23.2	0.39	3.8
Fertility	400K	4.44	103.5	1.99	15.3

4.3 Window pruning

During the window propagating process, there are many windows generated but do not contribute to the geodesic distance at vertices. We combine our window intersecting scheme with the rule "one angle one split" [9] as one window merging rule to eliminate redundant windows or redundant parts of windows from propagation. In our algorithm, windows on the same edge are organized in two lists wl_f and wl_b where wl_f stores the windows which direct on the forward halfedge and wl_b stores the windows which direct on the backward halfedge. All the window in the same list are stored in an order-preserving way without overlapping. In a list wl on the edge e , if w_j is the window stored right after a window w_i then the stop point of w_i is always smaller than the start point of w_j . Assume there is a new window w_{income} propagates on wl and it intersects with w_i . Using the intersecting scheme, we can compute the subintervals in th intervals covered by w_i , w_{income} and define which window from w_i , w_{income} is better to cover these subintervals. If the last subinterval is covered by w_{income} then we keep that part of w_{income} for the intersecting step with w_j (see Figure 4.9). For this situation, the flag $w_{income_survive}$ is set as *TRUE* in Algorithm 3. After merging, all the new windows created from w_{income} in wl are checked again by the rule "one angle one split". The new windows created from w_i need to be checked only when the geodesic distances of two start/end vertices of e are updated. The experimental results show that our algorithm prunes more redundant windows and requires less computing of Euclidean distance in comparison to the VTP schemes.

4.4 Window propagation management

In the MMP and ICH algorithms, all windows on the wavefront are stored in a priority queue to guarantee the quality of the propagating wavefront. Each window enter into

Figure 4.9: After intersecting of w_{income} with w_i , the red part of w_{income} is used as the new income window for intersecting with w_j .

the queue only one time and at the beginning time of the propagation process, the window with smallest distance in the queue is popped out for propagation. Because the total number of the windows which enter into the queue is very big, the propagation management using priority queue of windows has large overhead and it is the most time consuming step in MMP/ICH algorithms. The second reason which makes the using window priority queue becomes a bottleneck is the small number of propagated windows in each iteration of the wavefront propagation. Qin *et al.* [55] proposed the VTP algorithm with a new triangle-oriented window propagation to process more windows in each iteration of wavefront.

In the VTP algorithm, the separating point sp_i of each window w_i in a window list wl is defined as a point inside w_i where the distance from s_i to v_3 through sp_i is minimal (see Figure 4.10). The separating point sp of wl is defined as the best from all the separating points sp_i of the window $w_i \in wl$. If w_i is the window which provides the separating point sp then the geodesic distance from s_i to v_3 through sp is always smaller than all the geodesic distances from other windows in wl to v_3 .

For the propagation management in VTP, all the vertices on the wavefront are stored in a set S using geodesic distance as the factor to define the priority. If v is the front vertex at the top of S . Before the propagation from v , all the window lists on its adjacent edges are added into a FIFO queue wl_{queue} . Assume window list wl is the first list in

Figure 4.10: Separating points of the windows w_1 and w_2 on the edge (v_1, v_2) .

wl_{queue} and it has the correspondent edge (v_1, v_2) of the triangle $\triangle(v_1, v_2, v_3)$. During the propagation of wl over $\triangle(v_1, v_2, v_3)$, the geodesic distance of v_3 is approximated as the length of the geodesic path over the separating point sp of wl . After the propagation of wl , v_3 is always inserted into the S and the non-empty lists on the edges (v_2, v_3) , (v_3, v_1) are pushed into wl_{queue} . The propagating process of v finishes when wl_{queue} is empty. This strategy of propagation management is more effective than the MMP algorithm because the size of vertex queue is much smaller than the size of window queue and each propagating iteration processes more than one window. Unlike the window queue in MMP, vertex v in VTP can be inserted into S more than one time and its geodesic distance could be updated multiple times. When v is currently in the queue S we need to delete v from S , update its geodesic distance and re-enter it into S again to guarantee the valid order in S . Hence there is still a big amount of overhead incurred for inserting of new vertices into S and updating geodesic distance of vertices in S .

To avoid these expensive operators which have logarithmic complexity, we use a simple class *QueueNode* as the primitive of window propagation management and all the front

(4.11.a) Window lists on the adjacent edges (4.11.b) The triangles inside ΔR are covered around A are propagated follow a triangle-oriented order after propagating all the window lists on the adjacent edges around A .

Figure 4.11: The triangle-oriented window propagation in the VTP algorithm. Figures are borrowed from [55].

node are organized in a priority queue Q .

```
struct QueueNode
{
unsigned id_vertex;      //index of the related front vertex
double temp_distance;    //temporary distance at the inserting moment
};
```

When the geodesic distance d of a front vertex v changes, we simply insert a node $(v.id, d)$ into Q . Thus there could be more than one node in Q with the same id_vertex value and different $temp_distance$ values. At the beginning of a propagating iteration, we pop out the top node N from Q . If $N.temp_distance$ is larger than the current geodesic distance of the vertex with index $N.id_vertex$, then N is eliminated from the propagation. In Figure 4.10, to avoid of doing unnecessary "one angle one split" checking of the window lists around v_3 , we insert v_3 into Q only when it gets an exact geodesic distance, or in other words, there is at least one window from the edge (v_1, v_2) covered v_3 .

4.5 Parallelized MMP algorithm

Most of the existing exact geodesic algorithms [9, 62] don't work in parallel mode due to the lack of parallel structure. They adopt the "window propagation" strategy to propagate geodesic information from sources to vertices. All generated windows queue in an associative container and they are processed sequentially and globally. Without a strict order of queuing, the wavefronts become non-optimal and the performance drops significantly.

Ying [79] extended the classic CH algorithm to propose a GPU-based method called PCH for parallel computing the exact geodesics on triangle meshes. The main idea is about dividing CH's sequential algorithm into four stages: parallel window selection, parallel window propagation, data organization and processing the update events. All the generated windows are organized in such a way that there is no data dependence in each phase. Hence these phases can be completed in parallel manner. In each iteration, more than one window can be selected for propagation to speed up the algorithm convergence.

Since all these parallel approaches are GPU-base, it raises an obvious question about the existing of a parallel version of IMMP algorithm on CPU and how much the performance could be improved from other algorithms. However, in MMP algorithm, generated windows are managed in a priority queue and they are pop out one by one following a very strict order. It means the MMP algorithm is hard to be parallelized and its performance strongly depends on the windows management.

To overcome these challenges, we propose a new parallel IMMP algorithm (**PIMMP**) which can process window lists around multiple vertices. The window propagating process is divided into four phases, including front node selection, window list propagation,

window list merging and vertex updating to guarantee that there is no data dependence or conflict in each phase.

Front Node Selection: In each propagating iteration, our algorithm propagates window lists around T vertices, which are closest to the source points. In each $i - th$ thread, we process only adjacent windows lists from selected vertex $i - th$ queue. This process runs in sequential manner to guarantee the selected vertices are popped out from the top of the vertex queue v_queue .

Window Propagation: Because there could be maximum two flows of windows from two separate thread comes into the same window list, for each window list wl we set two temporary buffers wl^{start} and wl^{end} to store its income windows (see Figure 4.12). Unlike in the IMMP algorithm, if v_i is selected at the front node selection phase then only adjacent lists of v_i follow the counterclockwise order are pushed into the FIFO queue wl_queue_i . In Figure 4.12.a, we merge the new income window in wl^{start} with the forward list wl_{31} on the edge (v_3, v_1) before pushing wl_{31} into the FIFO queue of v_1 . In Figure 4.12.b, we merge the new income window in wl^{end} with the forward list wl_{31} on the edge (v_3, v_1) before pushing wl_{31} into the FIFO queue of v_1 .

Window list merging: In this phase, for each selected vertex v we merge the forward adjacent lists around v with their non-empty buffers. For example, in Figure 4.12.a, we merge the new income window in wl^{end} with the forward list wl_{31} on the edge (v_3, v_1) . In Figure 4.12.b, we merge the new income window in wl^{start} with the forward list wl_{31} on the edge (v_3, v_1) before pushing wl_{31} into the FIFO queue of v_1 .

After that, we need to merge the opposite lists around v with their non-empty buffers. In Figures 4.12.c and 4.12.d, the new income windows stored in wl^{start} and wl^{end} will be merged into the opposite list on the edge (v_2, v_3) .

(4.12.a)

(4.12.b)

(4.12.c)

(4.12.d)

Figure 4.12: During the propagation around v_1 , window list wl_{12} on the edge (v_1, v_2) propagates to the edges (v_3, v_1) in (a), (b) and (v_2, v_3) in (c), (d). New income windows from wl_{12} on the edge (v_3, v_1) are stored in wl^{start} (a) or wl^{end} (b). Only the forward window lists on the edge (v_3, v_1) are pushed into the FIFO queue. New income windows from wl_{12} on the edge (v_2, v_3) are stored in wl^{end} (c) or wl^{start} (d) and will not be merged til the end of the list propagation around v_1 .

Vertex updating: we update geodesic distance at the start/end vertices of the lists which are merged in two phases above. If vertex v is selected out from w_queue in the "front node selection" phase, it will be pushed back into v_queue if at least one of its forward adjacent lists is non-empty.

4.6 Experimental Results

We implemented our geodesic algorithms on a 64-bit PC with an Intel Core i7 6700k 4.0GHz CPU and 32GB memory. The parallelized parts are implemented to run on CPU using multi-threading OpenMP interface.

As Chen and Han [9] suggested, in the worst case, the number of created windows are still $O(n^2)$. All window list propagation and pruning operations in IMMP have linear complexity with respect to the windows involved. Therefore, their total cost is $O(n^2)$. The front node management using priority queue has $O(n \log n)$ time complexity. In summary, the time complexity of IMMP is $O(n^2 + n \log n) = O(n^2)$. In our algorithms, windows are deleted right after their propagation, thus the space complexity of our algorithms is also $O(n^2)$. Although the space complexity in our algorithms is similar to all existing exact algorithms, it is shown in Table 4.2 that the required space in our algorithms are low, thanks to the efficiency of the window clipping schemes, which can effectively prune redundant windows or parts of windows.

To experimentally estimate the optimal value of f for the PIMMP algorithm, we run it on a variety of models with different resolution. Our results shown that the PIMMP algorithm archives its best time and space performances with f is around 0.5. For the values less than 0.5, our algorithm still performs well because we set the minimum number of running threads equal to the maximum number of available threads on the system to take the advantage of multi-threading CPUs. Our PIMMP algorithm runs slower and

requires much more memory when f increases closer to 1. It can be explained as if we select too many nodes in vertices queue to propagate their window lists around then the number of unnecessary "one angle, one split" checking/merging is called too often and the selected front vertices have very high change to be inserted back into the queue. Thus the size of the queue stays long in almost all propagating iterations, which leads to very high memory usage. In Figure 4.13 we show the results of our PIMMP algorithm running on two models Bunny (160k faces) and Kitten (1.1M faces) with different values of the input size factor f for illustration.

We also tested our algorithms on a wide range of models, from the small Gargoyle (40k faces) to the large-scale Bunny (1.6M faces), and compare them with the existing exact algorithms (ICH, MMP, VTP) to validate their performance. To evaluate overall performance, we measure running time, maximum number of active windows and maximum memory usage. From the results in Table 4.2 we can see that our IMMP out-performs all the existing exact algorithms in CPU-running mode. Its result is on more than 10% better than the VTP's result in both time and space aspects. In other hand, our PIMMP algorithm can significantly improve the time performance on large-scale models with a small additional memory usage.

The new algorithms IMMP and PIMMP can performs well on anisotropic models as Table 4.3 shows.

(4.13.a) Testing the size factor f on the model Bunny (160k faces).

(4.13.b) Testing the size factor f on the model Kitten (1.1M faces).

Figure 4.13: Testing the size factor f . Choosing a small f still fully utilizes the parallel propagating process because we always control the minimum number of running threads. On the other hand, choosing a large f would require many useless window list checking/merging and make the vertex queue remain longer, resulting in slow performance and high memory usage. We found the optimal value for f is around 0.5, with which the PIMMP archives its best execution time and memory usage.

Table 4.2: Performance comparison of our IMMP and PIMMP algorithm with other exact algorithms on running time, maximum memory usage and total maximum number of active windows. F is the number of faces on a model.

Model	Performance	ICH	MMP	VTP	IMMP	PIMMP ($f = 0.5$)
Gargoyle ($F = 40k$)	Running time (s)	0.95	0.66	0.16	0.13	0.09
	# active windows	874,215	668,091	6,061	5,558	6,530
	Max. memory usage (MB)	76.93	48.58	0.39	0.36	0.42
Armadillo ($F = 346k$)	Running time (s)	18.97	8.93	1.80	1.39	1.22
	# active windows	10,215,769	6,724,512	20,436	18,238	20,447
	Max. memory usage (MB)	898.99	488.32	1.31	1.17	1.31
Fertility ($F = 800k$)	Running time (s)	239.95	249.38	14.12	12.62	10.62
	# active windows	84,011,889	71,607,326	205,856	181,599	180,301
	Max. memory usage (MB)	7,393.05	5,165.33	13.17	11.62	11.54
Kitten ($F = 1.1M$)	Running time (s)	366.56	378.01	19.57	18.07	15.74
	# active windows	143,109,906	112,886,256	153,886	143,714	168,508
	Max. memory usage (MB)	12593.7	8,140.97	9.85	9.20	10.78
Bunny ($F = 1.6M$)	Running time (s)	598.73	496.06	25.77	23.84	18.83
	# active windows	211,204,023	154,547,216	164,644	149,694	206,641
	Max. memory usage (MB)	18,586	11,146.6	10.54	9.58	13.23

Table 4.3: Performance comparison of our IMMP and PIMMP algorithm with other exact algorithms on running time, maximum memory usage and total maximum number of active windows on anisotropic models. F is the number of faces on a model.

Model	Performance	ICH	MMP	VTP	IMMP	PIMMP ($f = 0.5$)
Block ($F = 800k$)	Running time (s)	106.9	90.7	8.0	7.9	6.9
	# active windows	47,845,555	37,940,667	97,032	80,726	92,643
	Max. memory usage (MB)	4210.41	2741.33	6.21	5.17	5.93
Rocker arm ($F = 960k$)	Running time (s)	252.0	225.9	13.0	12.7	12.4
	# active windows	10,215,769	6,724,512	20,436	18,238	20,447
	Max. memory usage (MB)	9592.91	5804.25	9.37	8.59	9.38
Impeller ($F = 1600k$)	Running time (s)	242.0	162.0	17.7	16.1	15.6
	# active windows	100,391,197	72,780,334	89,928	77,236	111,279
	Max. memory usage (MB)	8,834.43	5,259.38	5.76	4.94	7.12

(4.14.a) Model Block (800k faces).

(4.14.b) Model Impeller (1.6M faces).

(4.14.c) Model Rocker arm (960k faces).

Figure 4.14: Running out IMMP algorithm on anisotropic models.

(4.15.a) Model Bojito (24k faces).

(4.15.b) Model Knot (194k faces).

(4.15.c) Model Kids (400k faces).

Figure 4.15: Running out IMMP algorithm on high genus models.

Algorithm 3 Window merging Algorithm

Input: window list wl , income window w_{income} .
Output: window list wl .

If ($wl == \emptyset$)
 $wl \leftarrow w_{income}$
 return wl
End if

$w, w_{last} \leftarrow$ first window of wl
vector of sub-intervals $I \leftarrow \emptyset$
 $w_{income_survive} \leftarrow TRUE$
 $w_{income_cover_first} \leftarrow FALSE$ //it is *TRUE* if w_{income} covers the first sub-interval
While ($w \neq NULL$)
 $I, w_{income_cover_first} \leftarrow \text{IntersectingScheme}(w_{income}, w)$
 If ($S == \emptyset$)
 $w \leftarrow$ next window in wl
 continue
 End if

$n \leftarrow$ number of the sub-intervals in I
For i from 1 to $n - 1$ **do**
 If ($w_{income_cover_first} == TRUE$)
 create new window w_{new} as a part of w_{income} on $I[i]$
 $wl \leftarrow$ insert w_{new} right after w_{last}
 $w_{last} \leftarrow w_{new}$
 $w_{income_cover_first} \leftarrow FALSE$
 Else
 create new window w_{new} as a part of w on $I[i]$
 $wl \leftarrow$ insert w_{new} right after w_{last}
 $w_{last} \leftarrow w_{new}$
 $w_{income_cover_first} \leftarrow TRUE$
 End if
End for

If ($w_{income_cover_first} == TRUE$)
 $w_{income_survive} \leftarrow TRUE$
 modify w_{income} as the last sub-interval of I
 $w \leftarrow$ next window in wl
Else
 create new window w_{new} as a part of w on the last sub-interval of I
 $wl \leftarrow$ insert w_{new} right after w_{last}
 $w_{income_survive} \leftarrow FALSE$
 End if
End while

If ($w_{income_survive} == TRUE$)
 $wl \leftarrow$ insert w_{new} right after w_{last}
 $w_{income_survive} \leftarrow FALSE$
End if

return wl

Algorithm 4 Improved MMP Algorithm

Input: A triangle mesh M , the source vertex p ,

Output: The geodesic distance field D at vertices of M .

priority queue $v_queue \leftarrow \emptyset$

update geodesic distances of all adjacent vertices around p

$v_queue \leftarrow$ all adjacent vertices around p

update window lists on all opposite edges around p

While $v_queue \neq \emptyset$

QueueNode $n \leftarrow$ pop the top node v_queue

vertex $v \leftarrow M.vertex(n.id_vertex)$

If $n.temporal_distance > v.geodesic_distance()$

continue

End if

$E \leftarrow$ adjacent edges of v

If $v.type() == SADDLE$

 update geodesic distance of all adjacent vertices around v

 create and merge pseudo-window on all opposite edges of v

End if

FIFO queue $wl_queue \leftarrow$ non-empty lists on adjacent/opposite edges of v

While $wl_queue \neq \emptyset$

$wl \leftarrow wl_queue.front()$ pop the first list of wl_queue

check wl using "one angle one split" rule

If $wl \neq \emptyset$

$wl_{out_left}, wl_{out_right} \leftarrow$ propagate wl on edges e_{left}, e_{right}

 merge wl_{out_left} into wl_{left} on e_{left} ,

 merge wl_{out_right} into wl_{right} on e_{right}

If the first window in wl_{left} changed

 update geodesic distance at start vertex of e_{left}

End if

If the last window in wl_{left} changed

 update geodesic distance at end vertex of e_{left}

End if

If the first window in wl_{right} changed

 update geodesic distance at start vertex of e_{right}

End if

If the last window in wl_{right} changed

 update geodesic distance at end vertex of e_{right}

End if

If the opposite vertex v_{opp} of e is covered

 update geodesic distance at v_{opp}

$v_queue \leftarrow v_{opp}$

End if

End while

$wl \leftarrow \emptyset$

End while

End while

Chapter 5

Application

This chapter presents an automatic method for computing anisotropic 2D shape distribution on arbitrary 2-manifold meshes. Our method allows user to specify the direction as well as the density of the distribution. Using a pre-computed lookup table, our method can efficiently detect collision among the to-be-distributed shapes on 3D meshes. In contrast to the existing approaches, which usually assume the 2D objects are isotropic and of simple geometry, our method applies for complex 2D objects and can guarantee the distribution is conflict-free, which is a critical constraint in many applications. It is able to compute multi-class shape distribution can be implemented in parallel. Our method does not require global parameterization of the input 3D mesh. Instead, it computes local parameterization on the fly using geodesic polar coordinates. Thanks to the recent breakthrough in geodesic computation, the local parameterization can be computed at little cost. As a result, our method can be applied to models of complicated geometry and topology. Experimental results on a wide range of 3D models and 2D anisotropic shapes demonstrate the good performance as well as the effectiveness of our method.

5.1 Geodesic Polar Coordinates

A key component for computing the shape distribution is to map the 2D object onto 3D models. Global parameterization, adopted in the existing anisotropic sampling algorithm [35], is not a good choice, since global parameterization is computationally expensive and it also produces large distortion, which may lead to numerical issues and/or visual artifacts. A possible solution is the exponential map induced local parameterization, which builds a geodesic polar coordinate system on 3D surfaces. Given a *smooth* surface $\mathcal{S}$ and consider a point $p \in \mathcal{S}$. Each tangent direction $\mathbf{v} \in$ tangent plane T_p corresponds to a unique geodesic $\gamma_{\mathbf{v}}$ passing through p in direction $\mathbf{v}$, i.e., $\gamma_{\mathbf{v}}(0) = p$ and $\gamma'_{\mathbf{v}}(0) = \mathbf{v}$. Thus, a point $q \in \gamma_{\mathbf{v}}$ can be represented by a 2-tuple (ρ, θ) corresponds to the tangent direction $\mathbf{v}$, where ρ is the geodesic distance between p and q and θ is the polar angle. Differential geometry can guarantee that on a sufficiently small neighborhood of

Figure 5.1: Given a 3D triangle mesh M and a set of 2D anisotropic shapes, user first specifies a vector field for directional control (see the top-right inset) and a scalar field for density control (see the bottom-left inset). Then our method automatically distributes the given 2D shapes onto M , satisfying the user-specified directional and density constraints. It takes only 2.9 seconds to distribute 4 classes of objects on this 400K-face Fertility model. Timing was measured on a quad-core CPU at 2.66GHz.

(5.2.a)

(5.2.b)

Figure 5.2: Geodesic polar coordinates. (a) Our LC-enhanced SVG method is able to compute the geodesic distances for curved sources. Each offset curve is then parameterized by arc-length. (b) The offset distance δ together with the normalized arc-length $\hat{s}$ uniquely determine a point on the 3D surface. The geodesic polar coordinate system, formed by the 2-tuple $(\delta, \hat{s})$, defines a bijective map between $\mathbb{R}^2$ and the patch on 3D surface.

p , the exponential map is a diffeomorphism, i.e., both the function and its inverse are smooth.

Discrete exponential map has many applications in computer graphics and digital geometry processing, for example, surface decaling [57], Poisson disk sampling [71], intrinsic CVT computation [65], e.g. Moreover, exponential map can be extended to a general setting, where the source is a curve [61]. In spite of its popularity, the exponential map, in general, is not bijective due to two reasons: first, geodesics are not unique when their lengths exceed the injective radius. Consider if two geodesic paths γ_{v_1} , γ_{v_2} of equal length ρ meet at a point q , then both (ρ, θ_1) and (ρ, θ_2) refer to the same point q . Second, as shown in [78], when a geodesic path passes through a saddle vertex (whose cone angle is more than 2π), it splits into many outgoing geodesic paths, meaning that all the outgoing geodesic paths share the same tangent direction. To fix the above-mentioned issues, Sun *et al.* [61] adopted a two-step strategy: They observed that the non-bijective issue occurs usually on part of the to-be-parameterized patch. Thus, they proposed a method for quickly detecting such regions. Then they computed a harmonic function

to send these regions to rectangular domain. Since the target domain is convex, the harmonic map is guaranteed to be bijective. Schmidt [57] applied Dijkstra’s algorithm to spread out the local parameters and his method can parameterize self-intersecting strokes.

Both Sun *et al.*’s method and Schmidt’s method require local remeshing, which is a significant overhead, especially for the to-be-parameterized patch is big. In this section, we propose a simple-yet-effective method for computing the polar coordinates. Our method is completely integrated into the geodesic computation framework, and the induced parameterization is guaranteed to induce a bijective map.

Note that the combined SVG and GTU method can compute geodesic distances with both point sources and curve sources. Here we discuss the case of curve source, since the point source is a special case.

Figure 5.3: Polar coordinate on iso-curves.

Consider a curve C_s as the source. The u -axis is along the source curve C_s and u values range $[0, 1]$ after the normalization by the arc-length of C_s . Each geodesic offset curve is then parameterized by the arc-length to get the corresponding u values. As shown in Figure 5.3, we define C_i and

C_j as two iso-curves with a, c are the two corresponding start points of curve C_i and b, d are two arbitrary points on curve C_j . The u_b value at point b is $\frac{l(a,b)}{l(C_i)}$, where $l(a,b)$ represents the length of curve between a and b . Similarly, the u_d value at point d is $\frac{l(c,d)}{l(C_j)}$. The u value of each point are only related to the arc-length on the corresponding iso-curve, meaning that the polar coordinate is guaranteed to be bijective. In contrast to [61], our method does not solve any linear system, thus, it is numerically stable and efficient. Computational results show that our method is up to 10 times faster than [61](see Figure 5.4).

(5.4.a) Time for local parameterization is 1.7s. (5.4.b) Time for local parameterization is 0.18s.

Figure 5.4: Comparison with extended exponential map [61]. Our result (b) has comparable quality as Sun *et al.*'s method [61] (a) and much better time performance.

5.2 Parallel Distribution of Anisotropic Shapes

Two simple circular shapes do not conflict if the sum of the radii of their circumcircles is less than the distance between their centers. In [35], ellipsoidal shapes are represented with an anisotropic metric and the conflict checking process can be done as for circular ones. Our algorithm handles more flexible and complicated shapes, such as birds, fishes and branches, which are not always able to be represented as circles or ellipses. More than that, our algorithm is designed for parallel computing to guarantee the efficiency of the distribution process. The pipeline of our algorithm can be described by two stages: a) user specifies the input shape(s), the global vector field and the density characteristic on the 3D mesh surfaces; b) following user's specification, the input shapes can be randomly distributed and rendered in parallel. We will explore more details of these steps in the rest of this section.

Figure 5.5: Algorithmic pipeline of single-class shape distribution. (a) User specifies a scalar field to control the shape density and the locations of singularities and their indices. (b) We adopt Crane *et al.*'s method [13] to compute the global vector field which controls the shape orientation. (c) Our method automatically determines the locations, sizes and orientations of the 2D objects. Each blue curve-segment represents the central line of one object. (d) All distributed objects are guaranteed to be collision-free.

5.2.1 User Input

In our system, user inputs are consisted of three types: shape(s), global vector field and density characteristic.

Shape(s): User can apply arbitrary images as the input shapes. If only one simple circular shape is applied, our algorithm works as same/similar as the traditional blue noise sampling does. Multiple shapes are also supported in our system. By increasing the number and changing the types of the shapes, user is able to create more artistic and vivid results.

The global vector field: The global vector field is capable of guiding of the shapes' directions and is created spreading all over the input mesh surfaces. The direction of a shape is determined by a curve generating along the part of the global vector field. Note that the direction here is not a simple three-element vector in 3D space, it means the track (curve-segment) on the mesh surfaces that could be pasted by a shape.

We adopted the optimal method provided by Crane *et al.* [13] to generate the global vector field. It is controlled directly by setting singularities and direction constraints

on the surface, as shown in Figure 5.5(a). Each singularity has an index indicating the number of full rotations along a small loop around its self. These indices are disposed in such a way as to guarantee their sum equal to the Euler characteristic of the input mesh. User can also sketch strokes on the mesh surfaces to specify the general direction. After that, the global vector field is computed by solving a convex optimization problem with linear constraints.

The density characteristic: In our system, user is allowed to specify/modify the density characteristic globally and locally. Different from the traditional sampling algorithms, we use the distribution of curve-segments following the global vector field to represent the density characteristic. User inputs a sampling rate at first to globally define the density characteristic. Then a set of curve-segments are generated in parallel following the global vector field during the sampling process: each time we throw a sampling point, we start tracing a new curve-segment from it along the direction, which is interpolated from the global vector field. The density characteristic changes locally if the number of the curve-segments inside the selected area changes, which can also be thought as the length of the corresponding curve-segments is increased/decreased. User can interactively scale the length of curve-segments in a selected area with different density factors by using a customized brush. Note that the number of generated curve-segments could be more than the actual shapes distributed on the mesh surfaces. The main reason is that, besides the density characteristic, the distribution of shapes is also affected by our conflict checking scheme.

5.2.2 Single-Class Shape Distribution

When 2D shapes are mapped to the input mesh surfaces along curve-segments, to make sure they do not intersect each other we need an efficient method for conflict checking.

If a shape is long, we split it into equal size parts and check each part using precomputed distance table. Figure 5.6 gives a brief illustration.

Figure 5.6: For long shapes, we divide them to several parts and test the intersection separately.

Now we introduce the details of distance table technique. Given a 2D object S , we compute the smallest covering circumcircle $\odot(c, r)$, where c is the center and r is the radius of S (see Figure 5.8). The circle can be computed in linear time using [39]. We then choose a ray L from the center c to a fixed direction as the polar axis. Clearly, objects are guaranteed to be free of conflicts if the distance between their centers greater than $2r$. Objects within $2r$ may or may not conflict, depending on their orientations.

To efficiently determine whether or not two objects conflict, we pre-compute a look up table, named *Minimum Safe Distance Table*, or **MSDT** for short.

As shown in Figure 5.8, the relation between two objects s_i and s_j is determined by the distance between two center points $d(c_i, c_j)$ and the orientation of each object. The latter is characterized by the angles between the local polar axes at c_j, c_j and the line $c_i c_j$, denoted by α_i and α_j , respectively.

We uniformly divide $[0, 2\pi)$ into k_a angle intervals, and build a $k_a \times k_a$ look up table.

Figure 5.7: Comparison with Poisson disk sampling. Row 1: Poisson disk sampling considers each sample as a disk and does not allow overlapping disks (a). There are 103 stars distributed in (b) Row 2: Our method allows a more dense distribution than Poisson disk sampling, as long as the stars are not overlapping. It randomly distributes 221 stars in (c) and greedy distributes 333 stars in (d).

For each pair of angles $\{\alpha_i, \alpha_j\}$, the entry $d_{(\alpha_i, \alpha_j)} \leq 2r$ is the minimum distance between the center points that guarantees no conflict, which is computed as follows:

If the object is a 2D polygon with e edges, we can check the polygon intersection in $O(e^2)$ time. In the interval $[0, 2r]$ if we pick k_r checking points, then the binary search for intersection of two polygons at these points has $O(e^2 \log k_r)$ time complexity. The pre-computing step of the *Minimum Safe Distance Table* can be done in $O(k_a^2 e^2 \log k_r)$ time and $O(k_a^2)$ space complexities.

If the object is a 2D image with b boundary pixels, the intersection of two objects s_i and s_j is detected by checking each of their boundary pixel. s_i and s_j are in conflict if

any boundary pixel of s_i locates inside s_j . Assume that these pixels are sorted by their index, we can test whether two objects are conflicted in $O(b)$ time complexity. Putting it all together, the construction of MSDT takes $O(k_a^2 b \log k_r)$ time and occupies $O(k_a^2)$ space complexities.

Figure 5.8: The circumcircle of shape (a) and the minimum safe distance between two shapes (red line) (b).

More space and time are required for pre-computing MSDT if we increase the value of k_a or k_r to distribute more shapes (see Figure 5.7).

By default, the density factor is set as 1 on the whole input mesh. If the density factor is changed, then an adaptive shape distribution is defined. Let f_i, f_j be the density factors of the local regions around points c_i and c_j , respectively. A new min safe distance d'_{α_i, α_j} between s_i and s_j can be approximated from the old distance d_{α_i, α_j} as below:

$$d'_{(\alpha_i, \alpha_j)} = \max\{f_i, f_j\} * d_{(\alpha_i, \alpha_j)}.$$

Assume that there are curves generated on the input mesh and we want to distribute shapes following these curves. Since the SVG graph is pre-computed for the input mesh, the distance between center points c_i, c_j and the angles $\{\alpha_i, \alpha_j\}$ can be computed efficiently. Thus in our algorithm, the conflict detection for a curve c_i can be done

Algorithm 5 Simplified Parallel MSDT Generation

Input: input shape S with radius r .**Output:** minimum safe distance table D .

```

function ParallelMSDTGeneration()
   $b \leftarrow$  find boundary pixels of  $S$ 
   $c \leftarrow$  random 2D point
  parallel for  $i$  from 0 to  $k_a - 1$  do
     $b' \leftarrow$  rotate  $b$  with angle  $\frac{i*2\pi}{k_a}$  around  $c$ 
     $d \leftarrow 0$ 
    for  $j$  from 1 to  $k_r$  do
       $d \leftarrow \frac{j*2r}{k_r}$ 
       $b_1 \leftarrow$  affine translate  $b'$  from  $c$  with distance  $d$ 
      if  $b \cap b_1 == \emptyset$  then
        break
      end if
    end for
    for  $j$  from 0 to  $k_a - 1$  do
       $\alpha \leftarrow \frac{j*2\pi}{k_a}$ 
       $\beta \leftarrow \alpha + \frac{i*2\pi}{k_a}$ 
       $D(\alpha, \beta) \leftarrow d$ 
       $D(\beta, \alpha) \leftarrow d$ 
    end for
  parallel end for
  return  $D$ 

```

without computing the geodesic polar coordinates of whole local disk $\odot(c_i, 2r)$ around the curve-source.

The pseudo-code in Algorithm 7 describes a simplified version of our method for single-class distribution.

5.2.3 Multi-Class Shape Distribution

Our system can distribute input shapes of multiple classes without overlapping. For a group of m input shapes $\mathcal{S} = \{S_1, S_2, \dots, S_m\}$, we have to compute $\binom{m}{2}$ MSD table between different classes and m MSD tables of the same class. Thus for a shape with

Algorithm 6 Simplified Conflict Detection using MSDT

Input: minimum safe distance table D , a set $\mathcal{C}$ of curves with density factor f , a candidate curve c_i , an input shape S with radius r .

Output: a set $\mathcal{C}_i^1$ of idle curves $\in \odot(c_i, 2r)$, a set $\mathcal{C}_i^2$ of active curves $\in \odot(c_i, 2r)$, a set $\mathcal{C}_i^3$ of accepted curves $\in \odot(c_i, 2r)$

```

function DetectConflictMSDT()
   $\mathcal{C}_i^1 \leftarrow \emptyset$ 
   $\mathcal{C}_i^2 \leftarrow \emptyset$ 
   $\mathcal{C}_i^3 \leftarrow \emptyset$ 
  for each  $c_j \in \odot(c_i, 2r)$ 
    if  $d_{(i,j)} \leq \max\{c_i.f, c_j.f\} * D(c_i.\alpha, c_j.\alpha)$  then
      if  $c_j.status == ACCEPTED$  then
         $\mathcal{C}_i^3 \leftarrow c_j$ 
        return
      else if  $c_j.status == IDLE$  then
         $\mathcal{C}_i^1 \leftarrow c_j$ 
      else if  $c_j.status == ACTIVE$  then
         $\mathcal{C}_i^2 \leftarrow c_j$ 
      end if
    end if
  end for
  return

```

b boundary pixels, the time complexity of our method is $O(m^2 k_a^2 b \log k_r)$ and it takes $O(m^2 k_a^2)$ memory space.

In multi-class case, a point/curve could be rejected because of conflict by one class and re-accepted later by others. If an object s_i of class $S_i \in \mathcal{S}$ is a candidate for conflict checking with its neighboring objects $\{s_j\}$ of corresponding classes $S_j \in \mathcal{S}$, we can use a general min safe distance $d_{(\alpha_i, \alpha_j)} = \max\{d_{(\alpha_i, \alpha_j)}^{S_i, S_j}\}, S_j \in \mathcal{S}, j = 1 \dots k_a$ for fast distribution or use all min safe distances $d_{(\alpha_i, \alpha_j)}^{S_i, S_j}, S_j \in \mathcal{S}, j = 1 \dots k_a$ for a denser distribution. We can also approximate $d_{(\alpha_i, \alpha_j)}$ from $\{d_{(\alpha_i, \alpha_j)}^{S_i, S_j}\}$ by different priority for each class to get nonuniform shape distributions of all classes.

The impact of the density factor in multi-class case is the same as that in the single-class

case.

5.3 Experimental Results

Performance. We implement the algorithm in C++ and test it on an Intel Xeon quad-core CPU at 2.66GHz. Thanks to the nice parallel structure of our method, we also use OpenMP to parallelize our algorithm so that it can take the advantage of all CPU cores. We observe that our parallel implementation on a quad-core CPU is up to 3 times faster than that of a single-core and it allows interactive manipulation on 3D models with 100K faces. See Tab. 5.1 for running time performance and the accompanying video demonstration.

Table 5.1: Mesh complexity and running time performance of sharp distribution process. T_1 and T_4 : running time (in seconds) on a quad-core CPU using a single core and all cores, respectively.

Model	$ F $	# of 2D shapes	T_1	T_4
Bump Sphere	50K	458	3.1	1.1
Bunny	80K	561	4.2	1.4
Fertility	400K	1,150	8.1	2.9
Kitten	250K	864	9.5	3.7

Figure 5.9, 5.10 and 5.14 demonstrate our results on single-class and multi-class shape distribution.

Robustness. Our method is intrinsic since both the collision detection and shape distribution depend on the metric only. Thanks to the intrinsic nature of the geodesic algorithms [75, 78], our method is insensitive enough to mesh resolution and tessellation. See Figure 5.11 for our results on the Bimba model with various resolutions.

Indeed, the accuracy of our local parameterization still depends on the geometric characteristics of the input meshes (mesh resolution, triangulation,...). When the input meshes have very low resolution, there could be small distortion appear (see Figure 5.12).

Figure 5.9: Single-class shape distribution on the Kitten model with different density values and textures. We control the density values f by using a brush with the center at the left eye and the user-defined radius r . The default value of f is 1.0. For the region inside the brush we have $f = \min\{0.25, d/r\}$ where d is the geodesic distance to the center.

Figure 5.10: Multi-class shape distribution on a bumpy sphere with 50K faces.

In Figure 5.4, we compared the time performance and accuracy of our method with Sun *et al.* [61]. Since their method computes a harmonic map to fix the non-bijective issue of the exponential map, it is more computationally expensive than ours. It takes their method 1.7 seconds to parameterize the 10K-vertex patch, whereas our method spends only 0.18 seconds with acceptable distortions due to the geodesic approximation. Here we set $K = 50$ for the SVG generation. Increasing K to 100 will almost double the time for pre-computing step in SVG but provides insignificant improvement on our accuracy. Clearly, if the local area is larger then the quality of our local parameterization is lower because of the geodesic approximation on long distances.

In our scenario, the time performance has higher priority and we focus to run our local parameterization on local area around given curves on large-scale models, thus the bad cases above could be mostly avoided.

Figure 5.11: Our method works well on the Bimba model with various resolutions: 30K faces in (a) and (b), 150K faces in (c) and (d).

Comparison to [35]: Our method is based on the dart throwing algorithm, so it fulfills the blue noise sampling property. Our method and the anisotropic blue noise sampling method [35] differ in aspects below: First, their method is mainly designed for the 2D problem. Although it can be extended to 3D surfaces via global parameterization, computing a high quality parameterization for high-genus model is technically

Figure 5.12: Distortion of the local parameterization on model Bimba with low resolution 7k faces in (a). Our results become smoother when resolution increases to 30k faces in (b) and 150k faces in (c).

challenging. Our method is based on local parameterization, which can be computed efficiently on the fly. Therefore, our method can be applied to models of arbitrary geometry and topology. Second, their parallelization techniques are hard to be extended to fulfill anisotropic criterion, since anisotropy poses a challenge in domain partition. Third, their method abstracts the anisotropic shapes by bounding ellipses so that collision detection and quality evaluation can be done in an analytical manner. However, ellipse is not an effective representation if the 2D shape has a highly concave boundary. As a result, their method may produce a distribution which is far from maximal. Thanks to the minimal safe distance table, our method can faithfully keep the geometry of the 2D shapes hereby allow a much denser distribution. Last but not the least, their method supports single-class sampling only, whereas our method can compute multi-class shape distribution (see Figure 5.13).

Comparison to [27]: Both our method and the Hu *et al.*' optimization-based method densely distribute irregular tiles of multiple classes with variable size without overlapping (see Figure 5.7). However, our method has some advantages: it supports long irreg-

Figure 5.13: Comparison to [35]. The 2D Eagle shape is highly concave, so a simple bounding ellipse cannot capture its geometric features. As a result, requiring non-overlapping ellipses is very pessimistic. Li *et al.*'s method [35] can distribute only 115 eagles (left). Obviously, the distribution is not maximal. Our method produces a much more dense distribution, containing 224 eagles (right), since it allows overlapping ellipses as long as the adjacent eagles do not collide.

ular tiles, which are not suitable for Voronoi diagrams approximation. The optimization process in [27] requires to compute local parameterization multiple times, meaning it is surely outperformed by our pre-computed approach. In our method, user can flexibly control density, direction of tiles without re-computing local parameterization.

Figure 5.14: More results of multi-class sharp distribution on Gargoyle (a) and Sculpture (b) models.

Algorithm 7 Simplified Parallel Single Shape Distribution with T threads

Input: triangle mesh $\mathcal{M}$, MSD table D , curve-set $\mathcal{C}$, input shape S .

Output: A set $\mathcal{P}$ of parameterized regions.

```

function ParallelSingleClassShapeDistribution()
     $\mathcal{P} \leftarrow \emptyset$ 
     $\mathcal{C}.status \leftarrow IDLE$ 
    while  $\mathcal{C} \neq \emptyset$  do
        for each thread  $i$  from 1 to  $T$ 
             $c_i \leftarrow \mathcal{C}.front()$ 
             $\mathcal{C}.pop\_front()$ 
             $c_i.status \leftarrow ACTIVE$ 
             $c_i.priority \leftarrow \frac{rand()*T+i}{RANDMAX*T}$ 
        end for
         $r \leftarrow$  compute radius of  $S$ 
        parallel for each thread  $i$  from 1 to  $T$ 
             $\mathcal{C}_i^1 \leftarrow \emptyset$  //idle curves  $\in \odot(c_i, 2r)$ 
             $\mathcal{C}_i^2 \leftarrow \emptyset$  //active curves  $\in \odot(c_i, 2r)$ 
             $\mathcal{C}_i^3 \leftarrow \emptyset$  //accepted curves  $\in \odot(c_i, 2r)$ 
             $\{\mathcal{C}_i^1, \mathcal{C}_i^2, \mathcal{C}_i^3\} \leftarrow \text{DetectConflictMSDT}(c_i, r, D)$ 
            if  $\mathcal{C}_i^3 \neq \emptyset$  then
                 $c_i.status \leftarrow REJECTED$ 
                 $\mathcal{C}_i^1, \mathcal{C}_i^2 \leftarrow \emptyset$ 
            end if
        parallel end for
        parallel for each thread  $i$  from 1 to  $T$ 
            CheckStatus( $c_i, \mathcal{C}_i^3$ )
            if  $c_i.status == ACCEPTED$  then
                 $P \leftarrow$  find parameterized region centered at  $c_i$ 
                atomic  $\mathcal{C} \setminus \{\mathcal{C}_i^1, \mathcal{C}_i^2, \mathcal{C}_i^3\}$ 
            end if
        parallel end for
    end while
    return  $P$ 

function CheckStatus( $c_i, \mathcal{C}_i^3$ )
    if atomic  $c_i.status \neq ACTIVE$  then
        return
    end if
    for each  $c_j \in \mathcal{C}_i^3$ 
        if  $c_j.priority > c_i.priority$  then
            CheckStatus( $c_j, \mathcal{C}_j^3$ )
            if  $c_j.status == ACCEPTED$  then
                atomic  $c_i.status \leftarrow REJECTED$ 
                return
            end if
        end if
    end for
    atomic  $c_i.status \leftarrow ACCEPTED$ 
    return

```

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This thesis aims at developing discrete geodesic algorithm to address the challenges of computing geodesics on the large-scale models. One sample applications is implemented to show the performance and accuracy of our methods.

In Chapter 3, we present an intrinsic algorithm based on saddle vertex graph to compute the approximate geodesic distances and paths on polygonal meshes, thus takes advantages of both the GTU and the SVG methods. Like the SVG algorithm, our method is accurate enough and numerically stable and still high efficient because there is no required retriangulation as in the GTU method. By using unfolding strategy into SVG, we can compute geodesics between any two points (not necessarily mesh vertices) with high accuracy. The unfolding process has constant time complexity, thus the geodesic distance field and geodesic paths can be approximated in linear time. It gives a big opportunity to apply our framework into a wide range of applications. The high time performance and accuracy are illustrated by experimental results on large models.

In our applications, the majority of the running time is for the computing of geodesic distances in terms of "multiple sources, multiple destinations" with priority queues. To

reduce the computing time, we replaced the classic Dijkstra’s algorithm by label correcting algorithm. The experiments show that the time performance of finding the shortest paths on SVG can be much improved and it makes our method run at least two time faster in comparison with the SVG method. This promising result leads to a belief that the performance can be improved much more with the implementation using techniques of the GPU parallel programming. We will further investigate this improvement in the future. Thanks to the advantage of time performance, our framework has the great potential for interactive operations, especially on large models and it could be applied in real-world problems.

In Chapter 4, we introduce IMMP and PIMMP algorithms for discrete geodesic computation with high performance in both time and memory aspects. The first major improvement is about designing an efficient approach for windows clipping, which can be used extensively in other geodesic algorithms. Instead of solving quadratic equations and computing Euclidean distance multiple times in all cases of window clipping as in classic exact geodesic algorithms, we describe this process through a system of equations and analyze their first order derivatives for fast window generation. In our approach, solving quadratic equations and computing Euclidean distance are required only in special cases, thus the number of calling expensive square-root function is significantly reduced. Plus an efficient window merging and front nodes management, we show that these improvements make our IMMP algorithm performs remarkably well on a variate range of models. We also propose PIMMP algorithm as a parallelized version of IMMP algorithm. We use two separate buffers for income windows on each window list and divide the IMMP sequential algorithm into four phases, front node selection, window list propagation, window list merging and vertex updating so that there is no data dependence or conflicts in each phase and the operations within each phase can be carried out in a parallel fashion on CPUs. Our PIMMP is able to propagate a large

number of windows simultaneously and independently, thus its performance is highly improved on real-world models.

The last contribution is about the applications in digital geometry processing. In Chapter 5, we present a shape distribution application, in which our method for computing geodesic distances plays an important role. Our application enables the user to control the direction field and density of shapes. It works well with both single-class and multi-class distribution. The input shapes can be mapped at arbitrary points with specific directions or followed curves. We proposed two approaches for fast conflict detection, which make our application run interactively and can produce nice result of texture mapping without overlapping between shapes. However, there are several limitations that we will improve in the future. First, our application requires the input meshes with enough high resolution. With more near sources selected around curves, the approximated mapping result will be smoother and more shapes will be distributed. Second, the curves cannot be self-intersecting. Assume that a curve can be considered as a continuous complex of line segments, if some segments from the same curve intersect each other, there are chances that their near sources are selected multiple times and then the geodesic distances from a point to them can be wrongly approximated. We will carry out further studies to improve our method and extend it for more applications. Some suggestions for improvements are referred in the next section of this report.

6.2 Limitations

Although our PIMMP algorithm can out-performs other exact algorithm running on CPUs with only a small additional memory usage, it is still challenging to fully parallel the window propagation process. In our PIMMP algorithm, picking correct value of factor f is an important task to guarantee the optimal performance, hence it is highly

desirable to theoretically estimate f . Another limitation is the lack of direct error/time controlling in the GTU/SVG combination because it may require a expensive re-build the geodesic graph. Finally, our shape distribution application could provides bad results when the quality of input meshes and the generated curves is low (input meshese with low resolution, bad triangulation or curves are short, with high curvature, self-intersecting,...), which can make the approximated parameterization become very inaccurate, even in a local area.

6.3 Future work

There are a few interesting future directions. First, we will remove the "one angle, one split" checking rule, which can work only on triangle meshes, to apply our algorithms to other discrete domains, such as quad meshes. And our PIMMP with good performance is highly desired in many time-critical applications which require extensive computation of geodesics. We will investigate such applications in the near future.

Another interesting direction is applying our work on dynamic meshes. Since our method has high performance, we can combine it with SVG algorithm to make it work incrementally on the input meshes, which are undergoing some non-isometric deformation. Assume that the time of deformation process is split into very short time steps. If the time steps are short enough, the changes on input mesh are only in small areas, thus we can reconstruct SVG locally. Thanks for the parallelable property used in SVG, the local update can be done efficiently. The whole process is repeated between continuous time steps to get the final result. Furthermore, when update results from previous frame, we have capacity to control error by time steps to make the whole deformation process smoother. We also can use result from the last time step to approximately anticipate results for the next future time steps. It would be more interesting to investigate

these techniques for working with incomplete data. We do believe it is very promising for us to make interesting applications for meshes matching, mesh reconstruction, mesh animation, skeletal animation, frame automatic generator, etc... with high performance.

In our shape distribution application, the generated curves need to be long enough because we treat each curve as a serial of some line segments. When the areas around two segments are close to each other or they intersect, there will be some sources, which are appropriate to be used for both of the segments. To avoid overlapping, each source can be selected for only one segment, then in the worst case, there could be not enough sources around some segments, leading to wrong results. To fix this issue, we can duplicate sources if necessary. Assume that two line segments (v_1, v_2) and (v_3, v_4) belong to the same curve, namely c . If a vertex s can be selected as a source for both (v_1, v_2) and (v_3, v_4) , we duplicate s to generate a new source s' and then select s for (v_1, v_2) and s' for (v_3, v_4) . We need to make s and s' different. Since s is a source for (v_1, v_2) and (v_1, v_2) , we delete all pseudo-source windows from s to (v_3, v_4) and all integral windows, which cover s and go to (v_3, v_4) , hence s accepts only windows come to (v_1, v_2) , or other ways to say, s blocks all approximate geodesic paths to (v_3, v_4) and allows only approximate geodesic paths to (v_1, v_2) . Similarly, we delete all pseudo-source windows from s' to (v_1, v_2) and all integral windows, which cover s' and go to (v_1, v_2) , hence s' accepts only windows come to (v_3, v_4) . Assume that vertex v is inside the geodesic offset of c and it has two approximate geodesic paths to c : one goes cross s into (v_1, v_2) and one goes cross s' into (v_3, v_4) . We generate a new vertex v' as a duplicate of v . To avoid distortion or self-overlapping at v , we parameterize v by using precomputed parameters of s , and parameterize v' by using precomputed parameters of s' . Finally, we add v' and its adjoin faces into input mesh and get a new mesh for drawing. Note that the new mesh is generated only for last texture mapping step, so we do not need to care about changes of the geometry topology of duplicated mesh.

It is also interesting to extend our work on smooth surfaces. Given a smooth surface, we can apply an effective adaptive sampling technique on it to generate a set of surface points. When there is enough sampling points collected, we can build a SVG graph over them using our IMMP algorithm for efficient local geodesic computation. As mentioned above, it is promising to develop a version of our IMMP algorithm which works flexibly with extend/rolling-back possibility. Thus when the number of sampling points in a local area changes, we can re-build the SVG graph locally with low cost of computation. In a nutshell, when the embedding mesh tessellations get finer and finer, we can adaptively update the geodesic distances at the sampling points and directly control the approximation error.

References

- [1] AGARWAL, P. K., ARONOV, B., O’ROURKE, J., AND SCHEVON, C. A. Star unfolding of a polytope with applications. *SIAM Journal on Computing* 26, 6 (1997), 1689–1713.
- [2] AGARWAL, P. K., HAR-PELED, S., AND KARIA, M. Computing approximate shortest paths on convex polytopes. *Algorithmica* 33, 2 (2002), 227–242.
- [3] ALEKSANDROV, L., MAHESHWARI, A., AND SACK, J.-R. An improved approximation algorithm for computing geometric shortest paths. In *Fundamentals of Computation Theory* (2003), Springer, pp. 246–257.
- [4] BATTIATO, S., MILONE, A., AND PUGLISI, G. Artificial mosaic generation with gradient vector flow and tile cutting. *Journal of Electrical and Computer Engineering 2013* (2013), 8.
- [5] BERTSEKAS, D. P. A simple and fast label correcting algorithm for shortest paths. *Networks* 23, 8 (1993), 703–709.
- [6] BERTSEKAS, D. P., GUERRIERO, F., AND MUSMANNO, R. Parallel asynchronous label-correcting methods for shortest paths. *Journal of Optimization Theory and Applications* 88, 2 (1996), 297–320.
- [7] BOWERS, J., WANG, R., WEI, L.-Y., AND MALETZ, D. Parallel poisson disk sampling with spectrum analysis on surfaces. In *ACM Transactions on Graphics (TOG)* (2010), vol. 29, ACM, p. 166.

REFERENCES

- [8] CAMPEN, M., AND KOBBELT, L. Walking on broken mesh: Defect-tolerant geodesic distances and parameterizations. In *Computer Graphics Forum* (2011), vol. 30, Wiley Online Library, pp. 623–632.
- [9] CHEN, J., AND HAN, Y. Shortest paths on a polyhedron. In *Proceedings of the sixth annual symposium on Computational geometry* (1990), ACM, pp. 360–369.
- [10] COHEN, M. F., SHADE, J., HILLER, S., AND DEUSSEN, O. *Wang tiles for image and texture generation*, vol. 22. ACM, 2003.
- [11] COOK, R. L. Stochastic sampling in computer graphics. *ACM Transactions on Graphics (TOG)* 5, 1 (1986), 51–72.
- [12] CRANDALL, M. G., ISHII, H., AND LOUIS LIONS, P. Users guide to viscosity solutions of second order partial differential equations, 1992.
- [13] CRANE, K., DESBRUN, M., AND SCHRÖDER, P. Trivial connections on discrete surfaces. In *Computer Graphics Forum* (2010), vol. 29, Wiley Online Library, pp. 1525–1533.
- [14] CRANE, K., WEISCHEDEL, C., AND WARDETZKY, M. Geodesics in heat: a new approach to computing distance based on heat flow. *ACM Transactions on Graphics (TOG)* 32, 5 (2013), 152.
- [15] DIJKSTRA, E. W. A note on two problems in connexion with graphs. *NUMERISCHE MATHEMATIK* 1, 1 (1959), 269–271.
- [16] DU, Q., FABER, V., AND GUNZBURGER, M. Centroidal voronoi tessellations: applications and algorithms. *SIAM review* 41, 4 (1999), 637–676.
- [17] DUNBAR, D., AND HUMPHREYS, G. A spatial data structure for fast poisson-disk sample generation. In *ACM Transactions on Graphics (TOG)* (2006), vol. 25, ACM, pp. 503–508.
- [18] EBEIDA, M. S., DAVIDSON, A. A., PATNEY, A., KNUPP, P. M., MITCHELL, S. A., AND OWENS, J. D. Efficient maximal poisson-disk sampling. In *ACM Transactions on Graphics (TOG)* (2011), vol. 30, ACM, p. 49.

REFERENCES

- [19] FATTAL, R. Blue-noise point sampling using kernel density model. In *ACM Transactions on Graphics (TOG)* (2011), vol. 30, ACM, p. 48.
- [20] FENG, L., HOTZ, I., HAMANN, B., AND JOY, K. I. Anisotropic noise samples. *Visualization and Computer Graphics, IEEE Transactions on* 14, 2 (2008), 342–354.
- [21] GAMITO, M. N., AND MADDOCK, S. C. Accurate multidimensional poisson-disk sampling. *ACM Transactions on Graphics (TOG)* 29, 1 (2009), 8.
- [22] GLOVER, F., GLOVER, R., AND KLINGMAN, D. Threshold assignment algorithm. *Netflow at Pisa* (1986), 12–37.
- [23] HAR-PELED, S. Constructing approximate shortest path maps in three dimensions. In *Proceedings of the fourteenth annual symposium on Computational geometry* (1998), ACM, pp. 383–391.
- [24] HAR-PELED, S. Approximate shortest paths and geodesic diameter on a convex polytope in three dimensions. *Discrete & Computational Geometry* 21, 2 (1999), 217–231.
- [25] HAUSNER, A. Simulating decorative mosaics. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), ACM, pp. 573–580.
- [26] HERSHBERGER, J., AND SURI, S. 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* (Philadelphia, PA, USA, 1995), SODA '95, Society for Industrial and Applied Mathematics, pp. 447–456.
- [27] HU, W., CHEN, Z., PAN, H., YU, Y., GRINSFELD, E., AND WANG, W. Surface mosaic synthesis with irregular tiles. *IEEE Trans. Vis. Comput. Graph.* 22, 3 (2016), 1302–1313.
- [28] HUANG, Q.-X., ADAMS, B., WICKE, M., AND GUIBAS, L. J. Non-rigid registration under isometric deformations. In *Computer Graphics Forum* (2008), vol. 27, Wiley Online Library, pp. 1449–1457.

REFERENCES

- [29] JIN, M., KIM, J., LUO, F., AND GU, X. Discrete surface ricci flow. *IEEE Transactions on Visualization and Computer Graphics* 14, 5 (Sept 2008), 1030–1043.
- [30] KANAI, T., AND SUZUKI, H. Approximate shortest path on a polyhedral surface based on selective refinement of the discrete graph and its applications. In *Geometric Modeling and Processing 2000. Theory and Applications. Proceedings* (2000), IEEE, pp. 241–250.
- [31] KIMMEL, R., AND SETHIAN, J. A. Computing geodesic paths on manifolds. *Proceedings of the National Academy of Sciences* 95, 15 (1998), 8431–8435.
- [32] KIMMEL, R., AND SETHIAN, J. A. Fast voronoi diagrams and offsets on triangulated surfaces. In *Proc. of AFA Conf. on Curves and Surfaces* (1999), University Press, pp. 193–202.
- [33] KOPF, J., COHEN-OR, D., DEUSSEN, O., AND LISCHINSKI, D. *Recursive Wang tiles for real-time blue noise*, vol. 25. ACM, 2006.
- [34] LAGAE, A., AND DUTRÉ, P. A procedural object distribution function. *ACM Transactions on Graphics (TOG)* 24, 4 (2005), 1442–1461.
- [35] LI, H., WEI, L.-Y., SANDER, P. V., AND FU, C.-W. Anisotropic blue noise sampling. In *ACM Transactions on Graphics (TOG)* (2010), vol. 29, ACM, p. 167.
- [36] LIU, Y.-J., CHEN, Z., AND TANG, K. Construction of iso-contours, bisectors, and voronoi diagrams on triangulated surfaces. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 33, 8 (2011), 1502–1517.
- [37] LIU, Y.-J., ZHOU, Q.-Y., AND HU, S.-M. Handling degenerate cases in exact geodesic computation on triangle meshes. *The Visual Computer* 23, 9-11 (2007), 661–668.
- [38] MARTÍNEZ, D., VELHO, L., AND CARVALHO, P. C. Computing geodesics on triangular meshes. *Computers & Graphics* 29, 5 (2005), 667–675.

REFERENCES

- [39] MEGIDDO, N. Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science* (1982), pp. 329–338.
- [40] MÉMOLI, F., AND SAPIRO, G. Fast computation of weighted distance functions and geodesics on implicit hyper-surfaces. *Journal of computational Physics* 173, 2 (2001), 730–764.
- [41] MÉMOLI, F., AND SAPIRO, G. Distance functions and geodesics on submanifolds of $\mathbb{R}^d$ and point clouds. *SIAM Journal on Applied Mathematics* 65, 4 (2005), 1227–1260.
- [42] MITCHELL, D. P. Spectrally optimal sampling for distribution ray tracing. *ACM SIGGRAPH Computer Graphics* 25, 4 (1991), 157–164.
- [43] MITCHELL, J. S. Geometric shortest paths and network optimization. In *Handbook of Computational Geometry* (1998), Elsevier Science Publishers B.V. North-Holland, pp. 633–701.
- [44] MITCHELL, J. S., MOUNT, D. M., AND PAPADIMITRIOU, C. H. The discrete geodesic problem. *SIAM Journal on Computing* 16, 4 (1987), 647–668.
- [45] MITCHELL, J. S., AND SHARIR, M. New results on shortest paths in three dimensions. In *Proceedings of the twentieth annual symposium on Computational geometry* (2004), ACM, pp. 124–133.
- [46] MOORE, E. F. The shortest path through a maze. *Proc. Int. Symp. Switching Theory, 1959* (1959), 285–292.
- [47] NEHAB, D., AND SHILANE, P. Stratified point sampling of 3d models. In *Proceedings of the First Eurographics conference on Point-Based Graphics* (2004), Eurographics Association, pp. 49–56.
- [48] NOVOTNI, M., AND KLEIN, R. Computing geodesic distances on triangular meshes.
- [49] OSTROMOUKHOV, V. Sampling with polyominoes. In *ACM Transactions on Graphics (TOG)* (2007), vol. 26, ACM, p. 78.

REFERENCES

- [50] ÖZTIRELI, A. C., ALEXA, M., AND GROSS, M. Spectral sampling of manifolds. In *ACM Transactions on Graphics (TOG)* (2010), vol. 29, ACM, p. 168.
- [51] PEYRÉ, G., AND COHEN, L. D. Geodesic remeshing using front propagation. *International Journal of Computer Vision* 69, 1 (2006), 145–156.
- [52] PEYRÉ, G., PÉCHAUD, M., KERIVEN, R., AND COHEN, L. D. Geodesic methods in computer vision and graphics. *Foundations and Trends® in Computer Graphics and Vision* 5, 3–4 (2010), 197–397.
- [53] PHAM-TRONG, V., SZAFRAN, N., AND BIARD, L. Pseudo-geodesics on three-dimensional surfaces and pseudo-geodesic meshes. *Numerical Algorithms* 26, 4 (2001), 305–315.
- [54] POLTHIER, K., AND SCHMIES, M. *Straightest geodesics on polyhedral surfaces*. ACM, 2006.
- [55] QIN, Y., HAN, X., YU, H., YU, Y., AND ZHANG, J. Fast and exact discrete geodesic computation based on triangle-oriented wavefront propagation. *ACM Transactions on Graphics (Proc. SIGGRAPH)* ?, ? (2016), ?
- [56] SANDER, P. V., WOOD, Z. J., GORTLER, S. J., SNYDER, J. M., AND HOPPE, H. H. Multi-chart geometry images, Sept. 8 2009. US Patent 7,586,488.
- [57] SCHMIDT, R., GRIMM, C., AND WYVILL, B. Interactive decal compositing with discrete exponential maps. *ACM Transactions on Graphics (TOG)* 25, 3 (2006), 605–613.
- [58] SETHIAN, J. A. Fast marching methods. *SIAM review* 41, 2 (1999), 199–235.
- [59] SETHIAN, J. A., AND VLADIMIRSKY, A. Ordered upwind methods for static hamilton–jacobi equations: Theory and algorithms. *SIAM J. Numer. Anal.* 41, 1 (Jan. 2003), 325–363.
- [60] SHARIR, M., AND SCHORR, A. On shortest paths in polyhedral spaces. *SIAM Journal on Computing* 15, 1 (1986), 193–215.

REFERENCES

- [61] SUN, Q., ZHANG, L., ZHANG, M., YING, X., XIN, S.-Q., XIA, J., AND HE, Y. Texture brush: an interactive surface texturing interface. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2013), ACM, pp. 153–160.
- [62] SURAZHISKY, V., SURAZHISKY, T., KIRSANOV, D., GORTLER, S. J., AND HOPPE, H. Fast exact and approximate geodesics on meshes. In *ACM Transactions on Graphics (TOG)* (2005), vol. 24, ACM, pp. 553–560.
- [63] TURK, G. Re-tiling polygonal surfaces. *ACM SIGGRAPH Computer Graphics* 26, 2 (1992), 55–64.
- [64] VARADARAJAN, K. R., AND AGARWAL, P. K. Approximating shortest paths on a nonconvex polyhedron. *SIAM Journal on Computing* 30, 4 (2000), 1321–1340.
- [65] WANG, X., YING, X., LIU, Y.-J., XIN, S.-Q., WANG, W., GU, X., MUELLER-WITTIG, W., AND HE, Y. Intrinsic computation of centroidal voronoi tessellation (cvt) on meshes. *Computer-Aided Design* 58 (2015), 51–61.
- [66] WANG, Y., SHI, J., YIN, X., GU, X., CHAN, T. F., YAU, S. T., TOGA, A. W., AND THOMPSON, P. M. Brain surface conformal parameterization with the ricci flow. *IEEE Transactions on Medical Imaging* 31, 2 (Feb 2012), 251–264.
- [67] WEBER, O., DEVIR, Y. S., BRONSTEIN, A. M., BRONSTEIN, M. M., AND KIMMEL, R. Parallel algorithms for approximation of distance maps on parametric surfaces. *ACM Transactions on Graphics (TOG)* 27, 4 (2008), 104.
- [68] WEI, L.-Y. Parallel poisson disk sampling. In *ACM Transactions on Graphics (TOG)* (2008), vol. 27, ACM, p. 20.
- [69] WEI, L.-Y. Multi-class blue noise sampling. *ACM Trans. Graph* 29, 4 (2010).
- [70] WILSON, D., AND LAXMINARAYAN, S. *Handbook of Biomedical Image Analysis: Volume 1: Segmentation Models Part A*. Topics in Biomedical Engineering. International Book Series. Kluwer Academic/Plenum Publishers, 2006.

REFERENCES

- [71] XIANG YING, SHI-QING XIN, Q. S., AND HE, Y. An intrinsic algorithm for parallel Poisson disk sampling on arbitrary surfaces. *IEEE Trans. Vis. Comput. Graph.* 19, 9 (2013), 1425–1437.
- [72] XIN, S.-Q., AND WANG, G.-J. Efficiently determining a locally exact shortest path on polyhedral surfaces. *Computer-Aided Design* 39, 12 (2007), 1081–1090.
- [73] XIN, S.-Q., AND WANG, G.-J. Improving chen and han’s algorithm on the discrete geodesic problem. *ACM Transactions on Graphics (TOG)* 28, 4 (2009), 104.
- [74] XIN, S.-Q., YING, X., AND HE, Y. Efficiently computing geodesic offsets on triangle meshes by the extended xin-wang algorithm. *Comput. Aided Des.* 43, 11 (Nov. 2011), 1468–1476.
- [75] XIN, S.-Q., YING, X., AND HE, Y. Constant-time all-pairs geodesic distance query on triangle meshes. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2012), ACM, pp. 31–38.
- [76] XU, C., LIU, Y.-J., SUN, Q., LI, J., AND HE, Y. Polyline-sourced geodesic voronoi diagrams on triangle meshes. In *Computer Graphics Forum* (2014), vol. 33, Wiley Online Library, pp. 161–170.
- [77] XU, C., WANG, T. Y., LIU, Y. J., LIU, L., AND HE, Y. Fast wavefront propagation (fwp) for computing exact geodesic distances on meshes. *IEEE Transactions on Visualization and Computer Graphics* 21, 7 (July 2015), 822–834.
- [78] YING, X., WANG, X., AND HE, Y. Saddle vertex graph (svg): A novel solution to the discrete geodesic problem. *ACM Transactions on Graphics (TOG)* 32, 6 (2013), 170.
- [79] YING, X., XIN, S.-Q., AND HE, Y. Parallel chen-han (pch) algorithm for discrete geodesics. *ACM Transactions on Graphics (TOG)* 33, 1 (2014), 9.
- [80] YING, X., XIN, S.-Q., SUN, Q., AND HE, Y. An intrinsic algorithm for parallel poisson disk sampling on arbitrary surfaces. *Visualization and Computer Graphics, IEEE Transactions on* 19, 9 (2013), 1425–1437.

REFERENCES

- [81] ZELINKA, S., AND GARLAND, M. Similarity-based surface modelling using geodesic fans. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing* (2004), ACM, pp. 204–213.
- [82] ZHOU, K., SYNDER, J., GUO, B., AND SHUM, H.-Y. Iso-charts: stretch-driven mesh parameterization using spectral analysis. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing* (2004), ACM, pp. 45–54.
- [83] ZIGELMAN, G., KIMMEL, R., KIRYATI, N., AND MEMBER, S. Texture mapping using surface flattening via multi-dimensional scaling. *IEEE Transactions on Visualization and Computer Graphics* (2002).