Compression and Querying of Arbitrary Geodesic Distances

Rosario Aiello[1], Francesco Banterle[1](✉), Nico Pietroni[1], Luigi Malomo[1,2], Paolo Cignoni[1], and Roberto Scopigno[1]

[1] ISTI - CNR, Pisa, Italy
francesco.banterle@isti.cnr.it
[2] Universitá degli Studi di Pisa, Pisa, Italy

Abstract. In this paper, we propose a novel method for accelerating the computation of geodesic distances over arbitrary manifold triangulated surfaces. The method is based on a preprocessing step where we build a data structure. This allows to store arbitrary complex distance metrics. We show that, by exploiting the precomputed data, the proposed method is significantly faster than the classical Dijkstra algorithm for the computation of point to point distances. Moreover, as we precompute exact geodesic distances, the proposed approach can be more accurate than state-of-the-art approximations.

1 Introduction

Determining the shortest path between two points on a surface, geodesic, is a fundamental task of many geometry processing algorithms. Mitchell, Mount and Papadimitriou (MMP) [12] proposed the first algorithm on polygonal surfaces with a practical computational complexity. Since then, several approaches have been presented to improve specific characteristics of geodesic computation.

In this paper, we introduce a method to speed up the computation of geodesic distances over triangulated manifold surfaces. Our method is based on a precomputation step where geodesic distances between vertices of an input triangular mesh are computed and efficiently stored in a data structure that allows fast querying. Storing all possible geodesics becomes practically infeasible in the real case, as the required memory rises quadratically with the number of vertices of the input mesh. Instead, our hierarchical data structure allows to retrieve geodesic distances between arbitrary vertices in constant time and, contemporarily, the required memory rises linearly with the number of vertices of the mesh.

Differently from previous methods, which are implicitly limited to compute the plain geodesic distance, the proposed method can be used to compute the shortest paths on a surface independently by the chosen distance metric. Therefore, it can be efficiently used to integrate any kind of function/signal over a surface. Our approach introduces an approximation error in geodesic computations, but this can be bounded in the preprocessing step.

V. Murino and E. Puppo (Eds.): ICIAP 2015, Part I, LNCS 9279, pp. 282–293, 2015.
DOI: 10.1007/978-3-319-23231-7_26

2 Related Work

Exact Methods. The MMP algorithm, proposed by [12], is the first algorithm that allows to compute exact geodesic distances on polyhedral meshes in $\mathcal{O}(n^2 \log n)$ time (where n is the number of vertices of a mesh). This algorithm subdivides each edge into intervals or *windows*. For each window, the distance is exactly computed and then propagated in the wavefront order. Chen and Han (CH) [3] proposed uses hierarchical windows to lower the complexity to $\mathcal{O}(n^2)$ time. However, Surazhsky et al. [16] showed that in practical cases this algorithms is slower and the MMP which typically runs sub-quadratic. Xin and Wang [19] discovered that 99% of the propagated windows in the CH algorithm do not contribute to shortest distance computations. So, they proposed to filter these useless windows improving efficiency of the CH algorithm (ICH). This has been implemented in the GPU [22] and extended for handling meshes with holes [14].

Approximated Methods. Surazhsky et al. [16] presented a version of MMP (AMMP) for the approximated computation of geodesic distances. By merging adjacent windows, the AMMP algorithm gain a significative speed-up in the computation (up to an order of magnitude), introducing only a 0.1% of relative approximation error.Xin et al. [20] proposed GTU, a method based on a precomputation step where the mesh is decomposed in triangular patches. The distance between each pair of point belonging to the same patch is precomputed and stored in a table. Geodesic distances are obtained in a two step process: first a Dijkstra shortest path computation is used to compute patch to patch distance, a second step completes the distance computation by using direct access tables. The Saddle Vertex Graph (SVG) approach recently proposed by Ying et al. [21] consists of a sparse undirected graph that encodes complete geodesic information so that every shortest path on the mesh corresponds to a shortest path on the SVG. Both GTU and SVG produce a considerable speedup in the computation, however the proposed method is limited to the computation of a specific geodesic measure. Fast Marching Method (FMM), proposed by Sethian [15] is a special case of the Level Set Method [13] for solving the *boundary value problems* of the Eikonal equation. This original method is limited to work for the case of regular grids. This method has been extended for the general case of triangular meshes [7], modified for fast computations on graphics hardware [18], or extended to work in meshes with holes [2]. Further extensions and modifications to the original algorithm were proposed in [10,11], Kirsanov's thesis [8], Martinez et al. [9], and Bertelli et al. [1]. Crane et al. [5] proposed a novel approach called *Heat Method* (HM). The main idea is to exploit the relationship between the heat kernel function $k_{t,x}(y)$ and distance function; i.e. image to touch a point x on the mesh surface with a scorching hot needle.The method is straightforward to implement and produces good approximations for relatively smooth surfaces; quality decreases in presence of fine details.

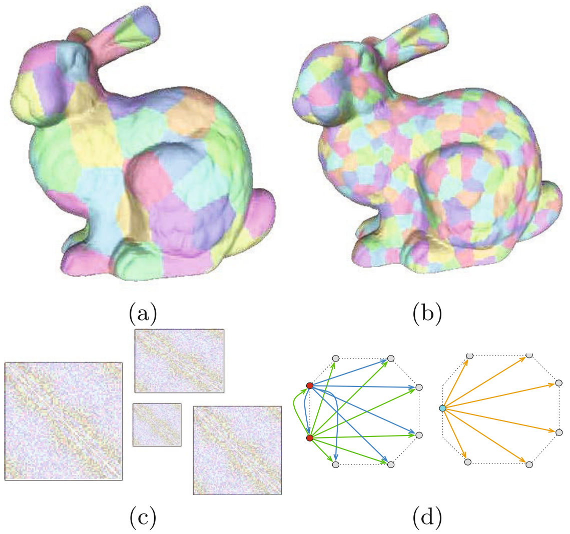


Fig. 1. Overview of the preprocessing pipeline: (a) build a Voronoi partitioning of the input mesh; (b) recursively subdivide each patch; (c) precompute for each region the geodesic distances and compress it as a set of images; (d) simplify the vertex-patch connecting graph by using a greedy pruning algorithm.

3 Algorithm

The preprocessing pipeline (see Fig. 1) to derive the hierarchical structure used to compute geodesics is composed by the following steps:

– We build a hierarchical partitioning of the input mesh. We recursively use a Voronoi-based approach using a set of uniformly sampled seeds (Fig. 1.a and 1.b). See section 3.1.
– For each region, we precompute the geodesic distances and we store them in a compressed graph based representation that allows for direct fast access (Fig. 1.c). See section 3.2.
– We simplify this graph by using a greedy pruning algorithm that minimizes the amount of introduced error (Fig. 1.c). See section 3.3.
– We assemble the graph that interconnects all the precomputed per patch information, in order to perform geodesic queries between any vertex pair.

3.1 Patch Subdivision

We want to subdivide the mesh into a set of disk-like patches that have approximately the same size. Moreover, in order to minimize the introduced error, the border between patches should be reasonably smooth and possibly convex with respect to the distance metric defined over the surface. To conform to our constraints, we used a strategy based on centroidal voronoi partitioning of the initial surface. Given a distance metric M defined over the input surface, we start from a Poisson Disk sampling of the surface under the metric M by using the algorithm proposed by [4] and choosing as seeds a subset of the mesh vertexes. Then, we perform Voronoi partitioning of the initial mesh by using the poisson samples as initial seeds; i.e. each vertex is associated to the closest seed under the distance metric M. In order to improve the partitioning, we interleave this step with a Lloyd relaxation step choosing as centroid of each patch the farthest point from boundary of the patch as in [17]. For this partitioning step when computing all the distances (for computing Voronoi partitioning and for Lloyd relaxation) we assume the use of the same distance metric of the geodesic we want to approximate.

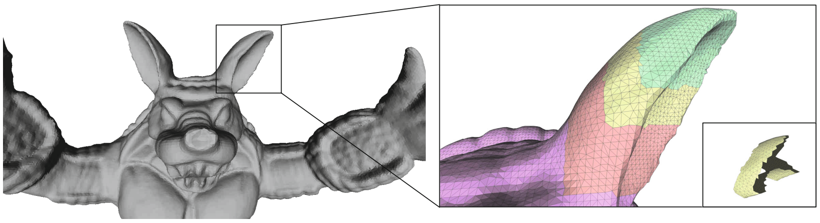


Fig. 2. A possible patching of the right ear of the ARMADILLO model. This partitioning creates a patch with two border rings.

This strategy does not guarantee that each patch is homomorphic to a disk. Indeed, it may produce patches with multiple borders. In the example shown in figure 2, the yellow patch has two border rings. This topological issue may commonly rise in correspondence of high curvature regions of the input surface (e.g. the ear of figure 2 or the tail which is also present in the ARMADILLO model). To overcome this issue, after a first partitioning step, we check each patch to have a single border. When this condition is not satisfied for a patch p_i, we re-run the Voronoi partitioning algorithm on the portion of the surface embedded by p_i. We repeat this step until every newly generated patch has exactly one single border.

This decomposition procedure is repeated in an hierarchical manner. We decompose each patch using the same Voronoi strategy, keeping tracks of the hierarchical relation between levels of subdivisions. In our experiments, we noticed that two levels of subdivision are enough to our purposes. Figure 3, on the left side, shows two examples of the patch decomposition.

The density of the patch subdivision is controlled by the number of seeds for the first level and the number of seeds used for each patch on the finest levels of the hierarchy. This constant can be set by the user at the beginning of the precomputation and it can be optimized to maximize the performance.

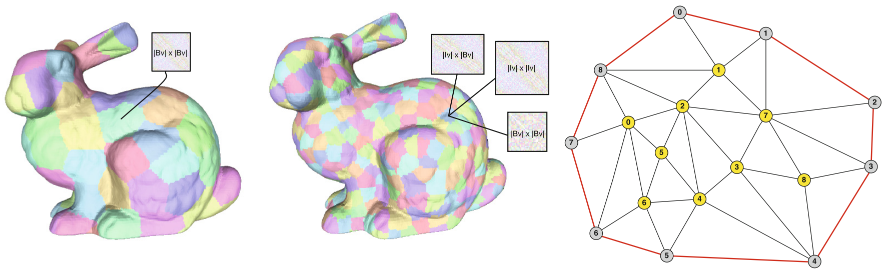


Fig. 3. On the left side, the first two levels of Voronoi decomposition of the bunny model, together with the precomputed geodesics distances. On the right side, an example of a triangulated patch. Border vertices (Bv) are shown in gray, internal vertices (Iv) are colored in yellow.

For each patch, we classify the vertices among two categories: *border* vertices (Bv) and *internal* vertices (Iv). In figure 3, on the right side, is shown an example: border vertices are colored in gray while internal vertices are in yellow. Border edges are highlighted in red.

3.2 Geodesic Precomputation

We precompute the geodesic distances by using the implementation of the MMP algorithm [12] as defined in Surazhsky [16]. This algorithm, despite the high computational cost, provides a very accurate estimation of the geodesic distance. As this is a preprocessing offline step, we are more interested in increasing the provided accuracy with respect to the time required by the process. The computation for each patch p can be summarized in the following steps:

1. For all the patches at any level, for each pair of border vertexes $v_i, v_j \in p$, we compute and store their relative distances in a $|\text{Bv}| \times |\text{Bv}|$ triangular matrix.
2. For each patch at the final level of the hierarchy, we compute the distance among border to internal vertices geodesic distance and store it in a $|\text{Iv}| \times |\text{Bv}|$ triangular matrix.
3. For each patch at the final level of the hierarchy, we compute the geodesic distances between each pair of internal vertices and store the values in a $|\text{Iv}| \times |\text{Iv}|$ triangular matrix.

These matrices represents graphs connecting various subsets of the nodes and whose arcs represents the precomputed distances. We keep these matrices as compressed images (we used "*.png*" compression), so that they can be efficiently stored to disk. We use *floats* to maintain distance values and the `RGBA8888` format to encode each float value into a pixel. Intuitively, assuming 32bit (4 bytes) floating point, we map each of the 4 bytes of the float value into each channel of the `RGBA` image format.

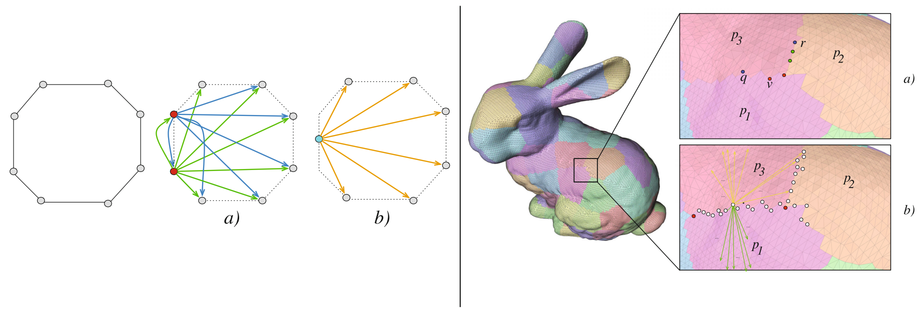


Fig. 4. Left: Pruning of two nodes: a) Two red vertices are selected for pruning. b) Pruning produces a new dummy node; Right:a) An example of a pair of nodes eligible for pruning (green) and two pairs not eligible (blue and red). b) An example of a border vertex and its |Bv| × |Bv| edges in both the patches it is contained. To have a clear picture, edges are only sketched.

As each patch has an unique border, we can sort the vertices along the border while internal vertices are sorted by considering their minimal distance to the border. Since close vertices in the patch will correspond to closer rows in the image, the produced image will vary smoothly allowing good compression ratios.

3.3 Graph Pruning

To reduce the size of the matrices described in the previous section, we simplify the graphs they represent by using a greedy pruning procedure that decreases the number of vertices actually used to generate all the pairs for which we store distances. The pruning procedure is done by iteratively merging adjacent patch boundary vertices: the intuition is that if two vertices are geometrically close (e.g. two adjacent vertices on a patch border) then their geodesics distances to other nodes will be "similar".

The error introduced by a merging operation can be estimated as the average difference of geodesic distances with respect to the other nodes of the adjacency matrices. The merging operation is shown in the left part of figure 4.a. Obviously, two nodes can be merged only if they belong to the same set of patches as shown in the right part of figure 4.

Merging operations are executed in a greedy fashion by prioritizing the operations with respect to the introduced error. The process halts when the introduced error exceeds a given threshold, δ. We express δ as a percentage of mesh bounding box diagonal. Results have shown that on small meshes (e.g. in the 70K-faces BUNNY model), a very small δ, such as $\delta = 0.01\%$, does not produce a significant pruning. In the case of high resolution meshes, the error introduced by every merging operation becomes smaller because the mesh is more densely sampled. Furthermore, the pruning becomes effective producing a considerably speedup.

3.4 Query Step

The Dijkstra shortest path computation is modified to exploit all the advantages of the hierarchical structure. There are two possible class of geodesic queries: The *SSSD geodesic computation* which stands for Single-Source to a Single-Destination and the *MSAD geodesic computation* which stands for Multiple-Source to All-Destination.

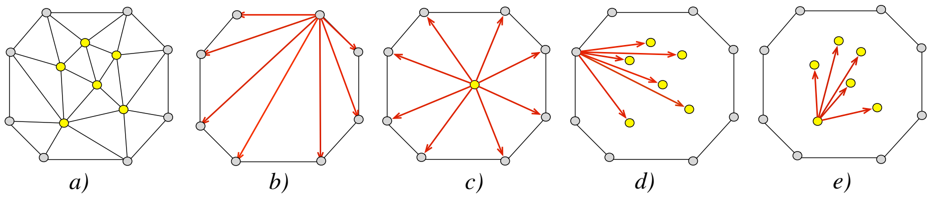


Fig. 5. An overview of the different class of relations and nodes that are contained by our hierarchical structure: (a) Border and internal nodes; (b) Border to Border arcs; (c) Internal to Border Arcs; (d) Border to Internal arcs; (d) Internal to Internal arcs

Figure 5 shows an overview of the different class of relations and nodes that are contained by our hierarchical structure.

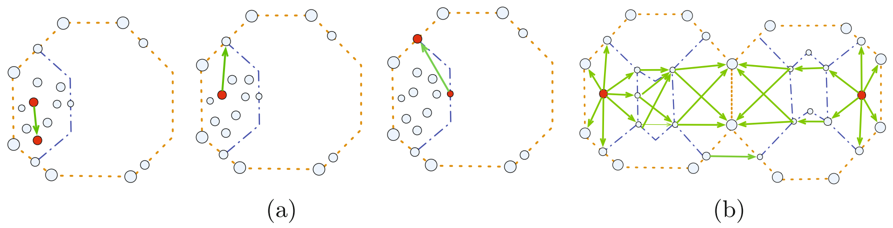


Fig. 6. a) Different cases of geodesic computation between vertices that belong to the same patch at the final level; b) The set of arcs enabled for Dijkstra shortest path computation for the general case.

SSSD Distance Computation. Given two arbitrary vertices, v_1 and v_2, we first retrieve the patches containing them along all the level of the hierarchy. If v_1 and v_2 belong to the same patch at the last level of the hierarchy p_i (as in figure 6.a left) then their distance can be simply retrieved by performing a direct access to the matrix relating internal to internal vertices of p_i.

Otherwise, when v_1 and v_2 do not belong to a common patch at the last lever we implicitly build a subset of all the arcs on which runs Dijkstra shortest path algorithm. We first retrieve the level L_{v_1,v_2} of hierarchy where v_1 and v_2 belong to a common patch. In the case there is no common patch, we consider the first level of the hierarchy. We only consider the arcs that relate each vertex and its ancestors up in the hierarchy until the level L_{v_1,v_2} is reached. We also enable all the arcs relating border to border relation at level L_{v_1,v_2}. In this way, we allow the propagation algorithm to make jumps as long as possible when the nodes are far away. Then, we execute a finer propagation when the front approach its final destination, see figure 6.b.

Moreover, we took advantage of the A* algorithm to further optimize the search. The A* algorithm requires a knowledge-plus-heuristic cost function of node v (denoted with $f(v)$) to determine the order to examine the visited nodes. The cost function $f(v)$ is a sum of two functions:

– the *past* path-cost function, which is the known distance from the starting node to the current node v (denoted with $g(v)$)
– a *future* path-cost function, which is an admissible "heuristic estimate" of the distance from v to the goal (denoted with $h(v)$).

The $h(v)$ part of the $f(v)$ function has to be an admissible heuristic; that is, it must not overestimate the distance to the goal. In our case, it is simply the Euclidean distance to the destination vertex. Therefore, we obtain a heuristic that is monotone (i.e. satisfies $h(v) \leq d(v, u) + h(u)$ for every edge (v, y)) and it is a lower bound. Indeed, the length of the shortest path between two points cannot be shorter than the norm of the vector connecting them. As exposed in [6,23], the A* algorithm achieves better timing when using admissible monotone heuristics.

MSAD Distance Computation. The MSAD distance computation algorithm is a generalization of the SSSD computation. We first use patch internal geodesic matrices to propagate all the distances to the vertices that belong to the same patch. Then, we propagate distances to the border of the patch. We use all the border to border adjacency matrices to propagate distances. Finally, we patch internal geodesic matrices to propagate the distances to the final nodes.

4 Results

We implemented our algorithm in C++ using the VCG library [1], an open source portable C++ template library for geometry processing. All our tests were performed on a machine equipped with an Intel i7 processor with 8 cores at 2.6GHz. Only the preprocessing part was parallelized.

Figure 7 shows the results of the computation of MSAD distance on different models (distance isolines are visualized). Our results are compared to those obtained by applying the MMP algorithm. We can see that our method produces smooth geodesic fields. Moreover, by checking the isolines, no artifacts are visible. The achieved accuracy in our results makes our results hardly distinguishable from those computed by the MMP algorithm.

[1] http://vcg.isti.cnr.it/vcglib/

Table 1. The average query time (left) and mean average error (right) statistics when varying n_1 and n_2 on the Armadillo ($173Kfaces$) Busto ($255Kfaces$) and Ramesses (826K faces) models.

Armadillo

	n_1		
n_2	100	200	300
5	0.01126s	0.01128s	0.01145s
10	0.01134s	0.01137s	0.01176s
15	0.01166s	0.01164s	0.01215s

Busto

	n_1		
n_2	100	150	200
5	0.01989s	0.01979s	0.01978s
10	0.01983s	0.01983s	0.01986s
15	0.01986s	0.01990s	0.01991s

Ramesses

	n_1		
n_2	950	1000	1200
5	0.06295s	0.06088s	0.06112s
10	0.06335s	0.06027s	0.06202s
15	0.65986s	0.06077s	0.06243s

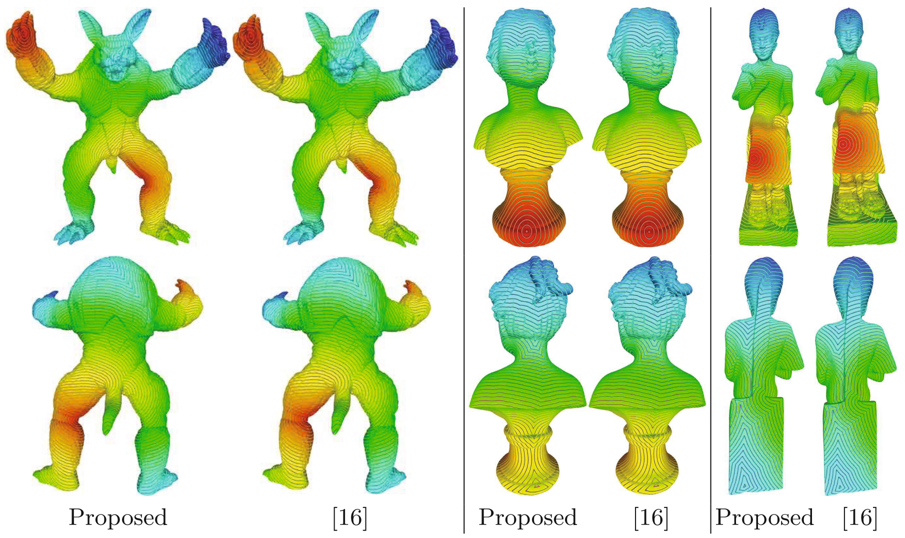
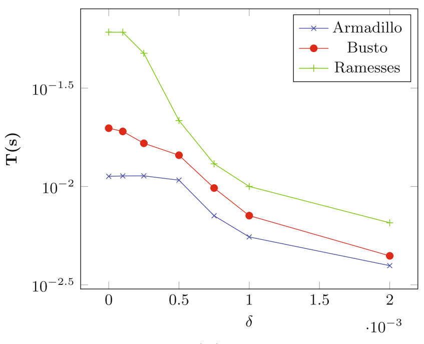


Fig. 7. Comparison of geodesics computed with our method with respect to the exact MMP algorithm by Surazhsky [16].

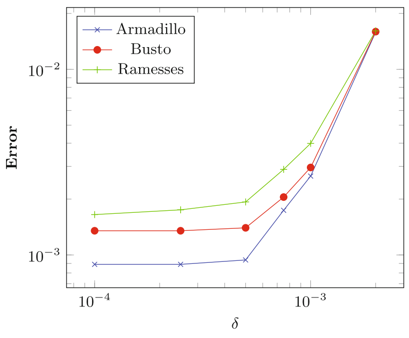
4.1 Parameters Tuning

In our algorithm, the user needs to specify the δ parameter which controls the introduced error during the pruning phase, see section 3.3). Additionally, the user has to specify the number of patch subdivisions used for each level of the hierarchy. We noticed that two levels of hierarchy are sufficient for high quality result. We refer to n_1 and n_2 as the number of subdivisions created for the first and second level of the hierarchy.

Some statistics relating the time needed to perform a query with respect to different values of n_1 and n_2 are reported in table table 1. To estimate the time needed to perform a query, we repetitively selected randomly pairs of nodes on the mesh to perform SSSD searches. From experiments, we noticed that the optimal number of seeds is proportional to the number of faces/vertices of the input mesh.



(a)



(b)

Fig. 8. Effect of parameter δ on query time and introduced error. δ and ϵ are expressed as % of the diagonal of the bounding box.(a) Average query time $T(s)$ plotted for variable pruning threshold δ. (b) Average mean error ϵ plotted for variable pruning threshold δ.

In figure 8, we report the average query time when varying δ. The error ϵ is relative to the geodesic evaluated by using the exact MMP algorithm [16]. Both δ and ϵ are expressed as a percentage of the diagonal of the model bounding box. As expected, the query time is inversely proportional to δ which governs the amount of merging operation performed during the pruning. Conversely, the amount of introduced error is proportional to δ.

4.2 Preprocessing Time

We report in table 2 timings for the preprocessing step when varying n_1 and n_2. Since our implementation runs computations for each patch in parallel, the preprocessing time tends to decrease when increasing the total number of regions in the first and second layers. On the other hand, as we discussed in section 4.1, this does not necessarily imply a decrease in the query time.

A comparison of geodesics computed with our method with respect to the exact MMP algorithm by Surazhsky [16] can be found in the additional material.

Table 2. Timings for the preprocessing step when varying n_1 and n_2 on the ARMADILLO model.

	n_1		
n_2	100	200	300
5	333.4s	218.7s	182.2s
10	287.3s	197.5s	202.1s
15	264.2s	191.1s	171.5s

Table 3. Comparisons of speedups of our algorithm with respect to exact MMP algorithm by Surazhsky [16] and the algorithm for computing geodesics included in the VCG library for the case of MSAD computations.

Model	MPP	VCG	VoroGeo	Speedup
Armadillo	1.3s	0.57s	0.011s	51x
Busto	8.1s	0.82s	0.019s	44x
Ramesses	47.3s	2.73s	0.061s	45x

4.3 Speedup and Comparisons

In table 3, we report the effective speedup achieved by our algorithm with respect to the geodesic algorithm employed in the VCG library. Through several experiments, we have also noticed that time performances became comparable when applying the VCG geodesic algorithm to a decimated version of the mesh. However, accuracy exponentially decreases making these results useless in practice.

5 Conclusions

We proposed a method to speed up the computation of geodesic distances on arbitrary manifold surfaces. The method is based on a preprocessing step where the mesh is decomposed into hierarchy of disk-like regions. Geodesic distances are precomputed on subsets of the vertices of those regions and stored for a later use.

Compared to previous methods, our method is independent from the metric used to compute distances over the surface. Additionally, our method allows to control the amount of introduced error with respect to the exact value.

We successfully integrated the exact geodesic computation proposed by Surazhsky [16] within our framework. This method [16] is very accurate (close to the exact geodesic value), but it unfortunately demands high computational costs. We showed several examples how our method can be used to speed up the entire process introducing a very small approximation error.

We believe that many geometry processing algorithms may benefit from the proposed method. Furthermore, our method may be used to integrate over manifold surfaces any arbitrary point-to-point functions, independently from their complexity, and make the querying very efficient. Thanks to this flexibility, our method may be used as an essential component in various application scenarios.

References

1. Bertelli, L., Sumengen, B., Manjunath, B.S.: Redundancy in all pairs fast marching method. In: 2006 IEEE Int. Conf. on Image Processing, October 2006
2. Campen, M., Kobbelt, L.: Walking on broken mesh: Defect-tolerant geodesic distances and parameterizations. Computer Graphics Forum **30**(2) (2011)
3. Chen, J., Han, Y.: Shortest paths on a polyhedron. In: Proc. of the Sixth Annual Symp. on Computational Geometry, SCG 1990. ACM, New York 1990
4. Corsini, M., Cignoni, P., Scopigno, R.: Efficient and flexible sampling with blue noise properties of triangular meshes. IEEE Trans. on Visualization and Computer Graphics **18**(6) (2012)
5. Crane, K., Weischedel, C., Wardetzky, M.: Geodesics in heat: A new approach to computing distance based on heat flow. ACM Trans. Graph. **32**(5), October 2013
6. Dechter, R., Pearl, J.: Generalized best-first search strategies and the optimality of a*. J. ACM **32**(3), July 1985
7. Kimmel, R., Sethian, J.A.: Computing geodesic paths on manifolds. Proc. Natl. Acad. Sci., USA (1998)
8. Kirsanov, D.: Minimal Discrete Curves and Surfaces. PhD thesis, The Division of Engineering and Applied Sciences, Harvard University, September 2004

9. Martínez, D., Velho, L., Carvalho, P.C.: Computing geodesics on triangular meshes. Comput. Graph. **29**(5), October 2005
10. Mémoli, F., Sapiro, G.: Fast computation of weighted distance functions and geodesics on implicit hyper-surfaces. J. Comput. Phys. **173**(2), 730 (2001)
11. Mémoli, F., Sapiro, G.: Distance functions and geodesics on submanifolds of r and point clouds. Journal on Applied Mathematics **65**(4), August 2005
12. Mitchell, J.S.B, Mount, D.M, Papadimitriou, C.H.: The discrete geodesic problem. SIAM J. Comput. **16**(4), August 1987
13. Osher, S., Sethian, J.A.: Fronts propagating with curvature dependent speed: Algorithms based on hamilton-jacobi formulations. Journal of Computational Physics **79**(1) (1988)
14. Quynh, D.T.P., He, Y., Xin, S.-Q., Chen, Z.: An intrinsic algorithm for computing geodesic distance fields on triangle meshes with holes. Graph. Models **74**(4), July 2012
15. Sethian, J.A.: A fast marching level set method for monotonically advancing fronts. Proc. Nat. Acad, Sci. (1995)
16. Surazhsky, V., Surazhsky, T., Kirsanov, D., Gortler, S.J., Hoppe, H.: Fast exact and approximate geodesics on meshes. ACM Trans. Graph. **24**(3), July 2005
17. Valette, S., Chassery, J.-M.: Approximated centroidal voronoi diagrams for uniform polygonal mesh coarsening. In: Computer Graphics Forum, vol. 23 (2004)
18. Weber, O., Devir, Y.S., Bronstein, A.M., Bronstein, M.M., Kimmel, R.: Parallel algorithms for approximation of distance maps on parametric surfaces. ACM Trans. Graph. **27**(4), November 2008
19. Xin, S.-Q., Wang, G.-J.: Improving chen and han's algorithm on the discrete geodesic problem. ACM Trans. Graph. **28**(4), September 2009
20. Xin, S.-Q., Ying, X., He, Y.: Constant-time all-pairs geodesic distance query on triangle meshes. In: Proc. of the ACM SIGGRAPH Symp. on Interactive 3D Graphics and Games, I3D 2012. ACM, New York (2012)
21. Ying, X., Wang, X., He, Y.: Saddle vertex graph (svg): A novel solution to the discrete geodesic problem. ACM Trans. Graph. **32**(6), November 2013
22. Ying, X., Xin, S.-Q., He, Y.: Parallel chen-han (pch) algorithm for discrete geodesics. ACM Trans. Graph. **33**(1), February 2014
23. Zeng, W., Church, R.L.: Finding shortest paths on real road networks: The case for a*. Int. J. Geogr. Inf. Sci. **23**(4), April 2009