

Computing Geodesic Distances on Triangular Meshes

Marcin Novotni and Reinhard Klein

Institut für Informatik II.
Römerstr. 164.
D-53117 Bonn
Germany
{marcin,rk}@cs.uni-bonn.de

ABSTRACT

We present an approximation method to compute geodesic distances on triangulated domains in the three dimensional space. Our particular approach is based on the Fast Marching Method for solving the Eikonal equation on triangular meshes. As such, the algorithm is a *wavefront propagation method*, a reminiscent of the Dijkstra algorithm which runs in $O(n \log n)$ steps.

Keywords: geodesic distances, computational geometry

1 INTRODUCTION

Computation of shortest paths is a well known problem in Computational Geometry. In this paper we present a method of approximating the length of shortest paths between two vertices on a 2-manifold surface represented by a triangular mesh. This problem arises in a number of applications such as robot motion planning, navigation and geographical information systems. Our work in this field was motivated by very recent developments in the area of computer graphics. Numerous new approaches emerged in this field where an efficient computation of geodesic distances on triangular meshes is needed: Zigelman, Kimmel and Kiryati [Zigel01] presented a method for distortion minimizing texture mapping; a fundamental component of this algorithm is the computation of geodesic distances. Another important contribution was also recently presented by Hilaga et al. [Hilag01] where the accumulated geodesic distances were used to construct multiresolution Reeb Graphs serving as search keys in a database of 3D objects. Last but not least Praun et al. [Praun01] applied geodesic paths to construct topologically equivalent base domain triangle meshes for consistent mesh parameterization, which in turn may be applied to a spectrum of further applications, like finding correspondences between pairs of objects, morphing of 3D objects, texture transfer, etc.

In most of these applications the efficiency of computations is preferred over accuracy. Instead of optimal solutions efficiently computable high quality approximations are called for. Our method is based on the work of Kimmel and Sethian [Kimme98], who showed an application of the Fast Marching Method, in which they approximate geodesic distances on triangulated domains. The Fast Marching Method is essentially a wavefront propagation algorithm, the central idea is to advance a front in an upwind fashion to produce new distance values at the vertices of the mesh. As shown later, the *update step* computation used by Kimmel and Sethian [Kimme98] produces good results if the front propagates from a straight line seed but fails when the seed is a single point. This is already true for the simplest planar cases. In addition to this observation, the main contribution of this paper is a new update step for the case of a single seed point. The relevance of this case is indicated by the fact that in all applications mentioned above the seed is a single point and not a straight or polyline.

1.1 PREVIOUS WORK

The general problem of finding the shortest Euclidean path is defined as follows: given a set of disjoint polyhedra in \mathbb{R}^3 , find the shortest path between two points s and t avoiding the interior

of these polyhedra. This problem was shown to be NP-hard by Canny and Reif [Canny87]. In our case, we look for a shortest path $\pi(s, t)$ between the points s and t on a 2-manifold mesh, which is a special case of the above problem. The algorithms elaborated so far are usually formulated for convex polytopes, which is sufficient for a number of applications. The solution by Sharir and Schorr [Shari86] yields an algorithm with $O(n^3 \log n)$ complexity; an important additional result here is the proof, that every shortest path on a polyhedron unfolds to a straight line on a plane. Mitchell et al. [Mitch87] showed an improved $O(n^2 \log n)$ running time algorithm working on non-convex polyhedra as well. The best algorithm known to the authors for the case of non-convex polyhedra was presented by Chen and Han [Chen90] with $O(n^2)$ time complexity.

An important class of algorithms for geodesic distance computation comprises of approximating and so-called ε -approximating algorithms. The latter family computes shortest paths on convex polytopes with controllable accuracy: denote $d(s, t)$ the length of $\pi(s, t)$, the ε -approximating algorithms compute paths with maximal length of $(1 + \varepsilon)d(s, t)$. An $O(n \log 1/\varepsilon + 1/\varepsilon^3)$ algorithm solving this problem was presented by Agarwal et al. [Agarw97]. Approximating algorithms for the general case of non-convex objects were presented in [Lanth97]. A nice and comprehensive overview over the techniques and problems in shortest path searching can be found in [Mitch98].

1.2 PAPER OVERVIEW

In the next section we review the fast marching method formulated for the case of arbitrary triangulation and discuss the arising problems. An improvement of the update step of this scheme is presented in section 3. The results are presented in section 4, which is followed by a short discussion in section 5.

2 FAST MARCHING METHOD

The fast marching method on triangulated domains was introduced by Kimmel and Sethian [Kimme98]. The main idea of this algorithm is to simulate a front advancing from a source point and store the computed distance values at the vertices; the update procedure is monotonic, the vertices are processed in increasing order of the distance value. Mathematically, the above problem can be formulated as the solution of the Eikonal

equation:

$$|\nabla T| = 1.$$

Intuitively, we want the distance T to grow from the seed point s where $T(s) = 0$ by a gradient with unit magnitude.

2.1 ALGORITHM OVERVIEW

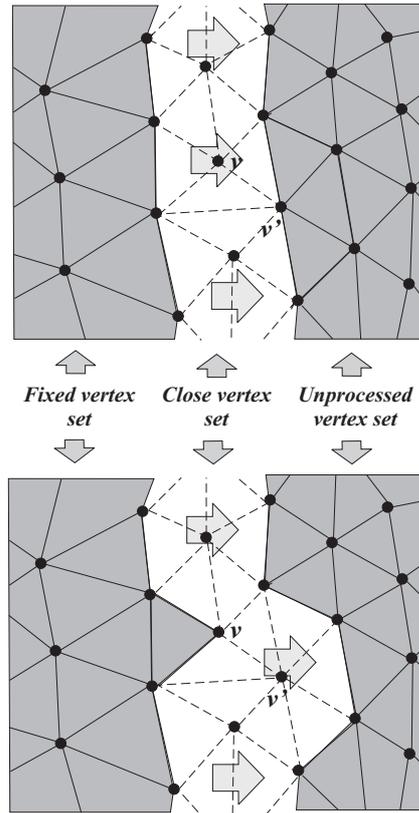


Figure 1: Demonstration of a step of the algorithm, the wavefront advances from left to right in this example. The gray areas on left and right side indicate the *Fixed* and *Unprocessed* vertex sets, respectively. The remaining vertices in between belong to the *Close* set. In the depicted step, vertex v having the smallest distance value in *Close* is included into *Fixed*. As a consequence of this, v' is included into *Close*.

The algorithm proceeds by propagating a wavefront outwards from a set of seed points. The advancing front can be thought of as a brushfire advancing with constant velocity in all directions in which the mesh has not yet been "burnt": we start the fire in the seed points and store the time values for each vertex at which the fire front has reached it. This is accomplished in a fashion very similar to the well known Dijkstra algorithm [Dijks59] for computation of shortest paths

in graphs. We essentially maintain and incrementally compute distance values for candidate vertices being adjacent to vertices, which already have according computed distance values. As in the Dijkstra algorithm, we include the candidate vertex with the smallest distance value into the set of vertices having final distance values and propagate and maintain the front of candidate vertices in the vicinity of the newly fixed vertex.

The overall algorithm is as follows (cf. figure 1: the vertices are initialized with the prescribed values, e.g. a zero distance value is assigned to the given seed vertices; these vertices are included to the *Fixed* vertex set which will contain the vertices with a final valid distance value. Then the according distance values for all the vertices, which are incident to triangles containing exactly two *fixed* vertices are computed, and included into the *Close* vertex set indicating that the values of these vertices may change. If there is only one *fixed* vertex as in the case of one seed point, Euclidean distances are used during initialization. As a final initialization step, all the remaining vertices are included to the *Unprocessed* set.

1. Begin Loop: Let *Trial* be the vertex in *Close* with the smallest *T* value. Add this vertex to *Fixed* and remove it from *Close*.
2. Compute the distance values for all vertices from $Close \cup Unprocessed$, which are incident to triangles containing *Trial* and another vertex in *Fixed*. If such a vertex was in *Unprocessed*, remove it from this set and add it to *Close*.
3. Return to Begin Loop.

In the algorithm, the newly computed distance values cannot be smaller than the values supporting this computation. This *monotonicity property* ensures that the solution is always propagated outwards by selecting the vertex with smallest *T* value. In other words no values corresponding to vertices in *Fixed* set will have to be recomputed. Eventually, every non-fixed vertex will have to be chosen as the one with smallest *T* value in *Close*, the complexity of the algorithm is therefore mainly influenced by this operation. In our implementation, we applied a pairing heap for this purpose, the above vertex location procedure takes $O(\log n)$ time using this technique. Since all vertices in the domain have to be processed, the overall time complexity of the algorithm is $O(n \log n)$.

Note that the above procedure is suitable only for triangles having an acute angle at the queried

vertex. If this is not the case, the situation may occur, that the supporting values for the distance computation are not yet available while processing the vertex, as the propagated front could not "reach" the second supporting vertex, cf. figure 2. As a solution to this problem *virtual vertices* are introduced for such obtuse triangles, see [Kimme98] for details on that.

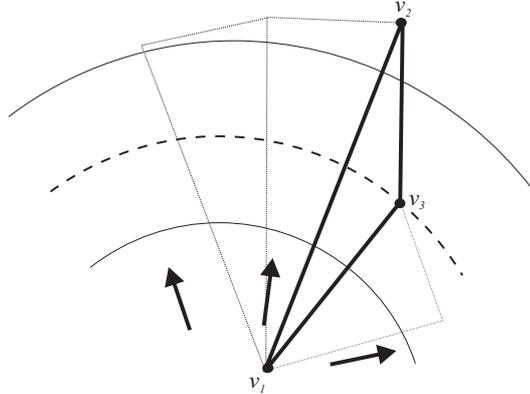


Figure 2: As the front indicated by the dashed arc arrives at the obtuse vertex v_3 , only the vertex v_1 is already processed and has a valid distance value. Therefore, special handling is needed for this triangle, see [Kimme98] for details.

2.2 UPDATE STEP

The main operation in the above procedure is the *update step*, which determines the distance value for a vertex based on distance values of two other vertices within the same triangle. The update step in the original work is computed as follows: given a triangle comprising of vertices v_1 , v_2 , and v_3 assume that the values $T(v_1)$ and $T(v_2)$ are known, we are looking for $T(v_3)$. We may further assume without loss of generality that the vertices v_1 , v_2 and v_3 lie in the xy -plane, thus we can depict the according distances by new points $v'_i = ((v_i)_x, (v_i)_y, T(v_i))$ above each of the vertices, cf. figure 3.

The value of $T(v_3)$ is computed in such a way that the angle between the normal \mathbf{n}' of the plane defined by v'_1 , v'_2 , and v'_3 and the normal \mathbf{n} of the plane defined by v_1 , v_2 , and v_3 is $\pi/4$. This can be formulated as follows:

$$\langle \mathbf{n}, \mathbf{n}' \rangle = \cos(\pi/4),$$

where $\langle \cdot, \cdot \rangle$ denotes the inner vector product. In general, there will be two solutions of the quadratic equation defined in such a way and the one larger than both $T(v_1)$ and $T(v_2)$ has to be

chosen in order to enforce the monotonicity of the solution.

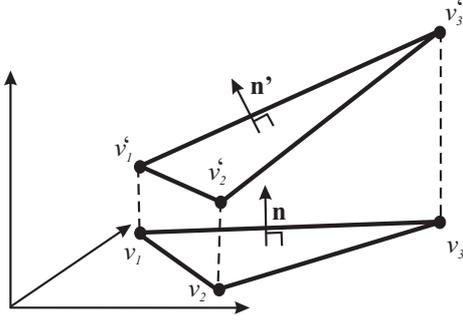


Figure 3: The plane defined by v_1' , v_2' and v_3' has to be tilted so that the angle between \mathbf{n} and \mathbf{n}' is $\pi/4$.

This procedure can be interpreted as follows: the intersection between the planes described above defines a straight line. The distance value assigned to v_3 is exactly the distance of v_3 from this line. In other words the propagated wavefront is a straight line and not a circular arc, which is desired for a single seed point.

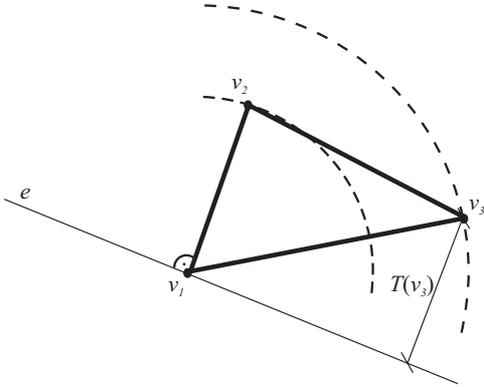


Figure 4: The situation if v_1 is the seed point and v_2 is assigned the Euclidean distance to v_1 . The plane will be tilted in a way that the direction of the gradient will be rectangular to the line e . Therefore, v_3 will be assigned its distances to e instead of the correct Euclidean distance indicated by the dashed arc.

Considering for instance a single seed point v_1 with distance value zero and two neighboring points v_2 and v_3 forming a triangle. Assume that v_2 has a distance value equal to Euclidean distance between v_1 and v_2 , see figure 4. In this case, there is only one solution of the quadratic equation, the plane can only be tilted in such a way, that the gradient points from v_2 towards v_1 .

$T(v_3)$ will coincide with the Euclidean distance to the seed point if and only if all the vertices are collinear, which is a pathological case.

3 NEW UPDATE STEP

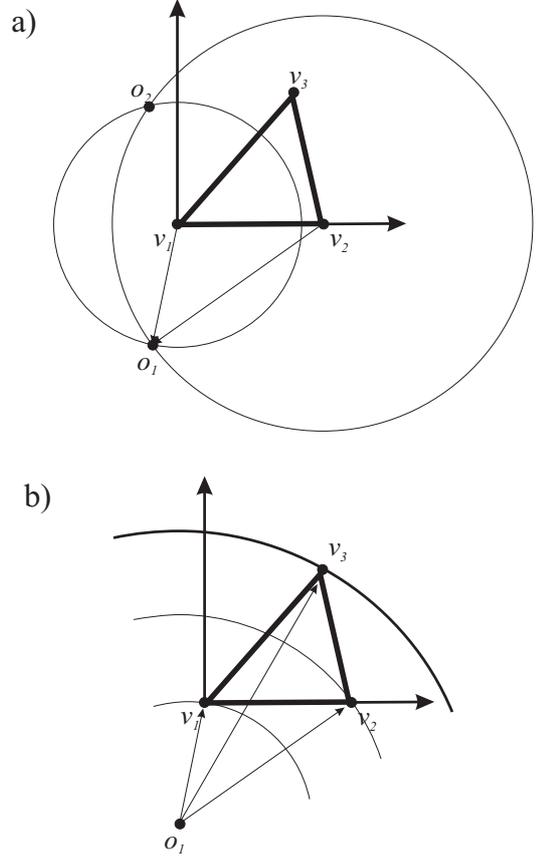


Figure 5: Computation of $T(v_3)$ in the plane of the triangle. a) The origin is in $T(v_1)$ and $T(v_2)$ distance from v_1 and v_2 , respectively. The possible origins are given by intersections of circles with loci at the vertices and radii of according distance values. b) $T(v_3)$ is computed as the larger of distances to the computed origins; the arcs depict the wavefront advancing from the origin.

The apparent reason for the failure of the algorithm in cases described in the previous section is that in the case of a single seed point, the propagated wave should not be modelled as a plane but in a circular or conical manner around the seed vertex. Therefore, in our new approach we model the wavefront as a circle around a virtual center, defined by the distance values of the already processed vertices.

The basic idea of our solution is depicted in fig-

ure 5. Using again the previously introduced notation, we assume without loss of generality that $v_1 = (0, 0, 0)$ and $v_2 = ((v_2)_x, 0, 0)$, which means that v_1 is in the origin and v_2 is on the x -axis of the coordinate system. This yields the following system of equations:

$$\begin{aligned} x^2 + y^2 &= T(v_1)^2, \\ (x - (v_2)_x)^2 + y^2 &= T(v_2)^2. \end{aligned}$$

The solution of this system is fairly simple, and gives two solutions in general. In order to enforce the monotonicity, we choose the solution yielding larger Euclidean distance between v_3 and the solution point. Let

$$\begin{aligned} A &= 2T(v_1)^2(v_2)_x^2 - (v_2)_x^4 + 2T(v_2)^2(v_2)_x^2, \\ B &= (T(v_1)^2 - T(v_2)^2)^2. \end{aligned}$$

The solution of the above system of quadratic equations:

$$\begin{aligned} (v_3)_x &= \frac{1}{2} \frac{(v_2)_x^2 + T(v_1)^2 - T(v_2)^2}{(v_2)_x}, \\ (v_3)_y &= \pm \frac{1}{2} \frac{\sqrt{A-B}}{(v_2)_x}. \end{aligned}$$

The intuition behind the above idea is to "project" the seed point to the plane of the triangle to be processed. In our approach, this projected origin is given by the solution of the above equation system, which is at larger distance from v_3 , see figure 5. This is exactly the distance $T(v_3)$ we are looking for.

Note however, that in the general case this is only an approximation to the exact shortest path length. If we wanted to exactly backtrack the shortest paths for the vertices, we would have to maintain a more complex data structure for each edge passed by the wavefront, see [Mount85a, Mount85b, Mitch87] for more details. On the other hand, the shortest path unfolds to a straight line only if it does not pass through a vertex (cf. [Shari86]), which is always the case for convex objects. In case of non-convex objects the unfolded shortest path may pass through vertices and therefore it may be not a straight line segment, but a polyline. Thus, similarly to the method of Kimmel and Sethian, our algorithm gives only an approximation of geodesic distances, which, however, is considerably more accurate than the former.

It should be mentioned that in some cases the usage of a polyline as wave front yields exactly the same solution as our new update step. This situation occurs if the plane can be tilted correctly, i.e. using the notation of previous section, v'_1 and v'_2 are in the plane and the gradient points from the v'_3 into the triangle. However, as we saw, this is not always the case.

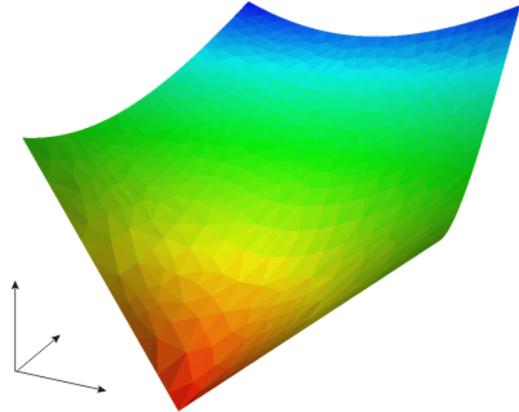


Figure 6: Result for the new update step in the case of a single seed point. We processed a triangulated planar square domain lying in the xy plane and raised the vertices along the z coordinate by the corresponding distances. All vertices lie on a cone having a vertical axis and its apex in the seed point.

4 RESULTS

To analyze the difference between the described update step computations, we applied the methods to a triangulated planar quadrilateral with one seed point. In this example, the geodesic distances between vertices are equal to Euclidean distances, thus making the verification of results easy in this simple case. The new update step introduced in this paper yields the result depicted in figure 6. The vertices lying originally on the xy plane were risen by the corresponding distances lie on a cone. The axis of the cone is parallel with the z axis and its apex is in the seed point. The angle between lines on the cone passing through the apex and the axis is $\pi/4$, which indicates that the distances grow by a gradient $|\nabla T| = 1$. Therefore, the computed distances are exactly the Euclidean distances from the original vertices to the seed point.

The result for the original update step is depicted in the upper image of figure 7. It can be seen that the distances do not advance according to a circular arc front, which would be the correct solution. Some directions can be distinguished where the update clearly fails to compute the correct values – there are mesh areas, where the distances grow slower, and there are regions where they grow faster. After taking a closer look, it can be seen that this is due to the already indicated fact that the computations are inaccurate if the edges are aligned with the direction of the gradient. In fact, the distances along the edges will

grow slower than the correct Euclidean distances.

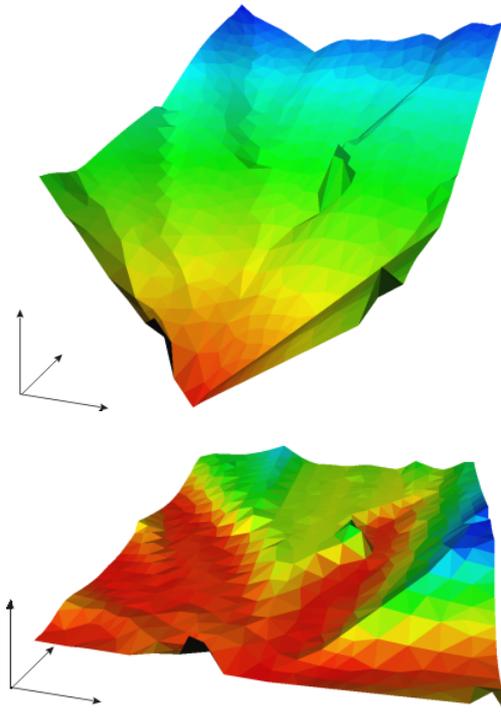


Figure 7: The result of applying the original update step for a triangulated planar square domain lying in the xy plane can be seen in the upper image. Similarly to the figure 6, the vertices have been risen along the direction parallel with the z axis according to their distance from the seed point to visualize the advancing wavefront. The lower image depicts the error of values computed by this procedure, the z values of vertices indicate the error. The distance values are inaccurate especially in cases where the edges are aligned with the gradient of the advancing front.

The lower image of figure 7 illustrate the error of computations using the original update step. The picture is a difference image between the correct solution (cf. figure 6) and the result yielded by the original procedure from upper image of figure 7.

We applied both algorithms to a number of 3D triangle meshes. The efficiency of the method comes from the fact that we need to conduct only a simple computation in the update step. The *Close* set always contains the vertices being adjacent to *Fixed* vertices in the vicinity of the advancing front which is only a small fraction of all vertices of the mesh. This way locating the vertex with smallest T value is very fast as well. Previous methods computing exact geodesic

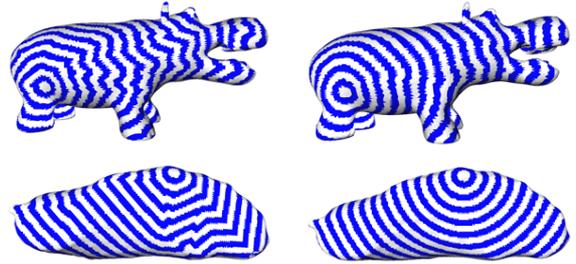


Figure 8: Results of running both methods for a hippo model and a foot scan viewed from below. On the left side we depicted the distance iso-values from one seed point computed by the algorithm of Kimmel and Sethian. As it can be seen, in some areas the front advances slower, in some others faster, which is due to the already mentioned alignment of triangle edges. Results using the new update step are shown on the right side, the wavefronts in this case are propagated with approximately constant velocity.

distances and paths did not reach the $O(n \log n)$ time complexity, some of the best methods are still quadratic. Kimmel and Sethian proposed a method, which utilized an incremental approach similar to Dijkstra's algorithm, however, their method does not compute accurate distance values. On the other hand, as indicated in the introduction, they provided an important alternative for geodesic distance computation, which has already been successfully used in a number of computer graphics applications. In this paper we showed how to improve their approach in terms of accuracy, which makes the method even more attractive. The data structure and implementation of the algorithm is very straightforward, the approach is inherently very fast due to incremental computations.

As a result of these, even for large meshes containing several tens of thousands of vertices, the computation of distance values can be performed in a fraction of a second, cf. table 1. An excerpt from our results for one seed point are depicted in figure 8. To visualize the advancing wavefront, we painted the triangles in these examples according to distance values of incident vertices in an alternating manner. In all of our experiments we encountered similar improvements.

Finally, we wish to indicate that the update step proposed in this paper is particularly well suited to geodesic distance computations from a single

Models	Vertex Count	Our Method	Kimmel's Method
Square	604	20.4	20.06
Foot	21817	755.1	747.1
Hippo	15446	558.3	553.8

Table 1: The timings in milliseconds measured for the algorithms. The results were measured on a 1200 MHz AMD Athlon PC with 256 MB RAM running Windows2000. We indicate the vertex counts of the models in the second column, in the third and fourth column are the corresponding timings. As it can be seen, there is practically no difference between timings of our and Kimmel's method, since both rely on a very similar procedure, only the update step has changed.

seed point. In case of multiple seed points, the wavefront will be the union of wavefronts starting from each seed point. The approach of Kimmel and Sethian may theoretically yield better results in this case, since they approximate the front by line segments. However, the issue of inaccuracy in case of edges being aligned with the wavefront gradient remains.

5 CONCLUSIONS

In this paper we analysed and improved the marching front method for geodesic distance computation proposed by Kimmel and Sethian. Especially, in the case of single seed points the results are better than the ones produced by the original algorithm. Although the algorithm presented here delivers good results if instead of single seed points arbitrarily shaped polygons are used as well, we are still working on further improvements for this case.

REFERENCES

- [Agarw97] Pankaj K. Agarwal, Sarel Har-Peled, Micha Sharir, and Kasturi R. Varadarajan. Approximating shortest paths on a convex polytope in three dimensions. *Journal of the ACM*, 44(4):567–584, 1997.
- [Canny87] J. Canny and J. H. Reif. New lower bound techniques for robot motion planning problems. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 1987.
- [Chen90] Jindong Chen and Yijie Han. Shortest paths on a polyhedron. In *Symposium on Computational Geometry*, pages 360–369, 1990.
- [Dijks59] Edsger W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik*, 1:269–271, 1959.
- [Hilag01] M. Hilaga, Y. Shinagawa, T. Kohmura, and T. L. Kunii. Topology matching for fully automatic similarity estimation of 3d shapes. In *ACM SIGGRAPH*, 2001.
- [Kimme98] R. Kimmel and J. A. Sethian. Computing geodesic paths on manifolds. *Proceedings of National Academy of Sciences*, 95(15):8431–8435, 1998.
- [Lanth97] Mark Lanthier, Anil Maheshwari, and Jörg-Rüdiger Sack. Approximating weighted shortest paths on polyhedral surfaces. In *6th Annual Video Review of Computational Geometry, Proc. 13th ACM Symp. Computational Geometry*, pages 485–486. ACM Press, 4–6 1997.
- [Mitch87] Joseph S. B. Mitchell, David M. Mount, and Christos H. Papadimitriou. The discrete geodesic problem. *SIAM Journal on Computing*, 16(4):647–668, 1987.
- [Mitch98] J. Mitchell. Geometric shortest paths and network optimization, 1998.
- [Mount85a] David M. Mount. On finding shortest paths on convex polyhedra. Technical Report CS-TR-1495, University of Maryland, 1985.
- [Mount85b] David M. Mount. Voronoi diagrams on the surface of a polyhedron. Technical Report CS-TR-1496, University of Maryland, 1985.
- [Praun01] E. Praun, W. Sweldens, and P. Schröder. Consistent mesh parameterizations. In *SIGGRAPH 2001 Conference Proceedings*, 2001.
- [Shari86] M. Sharir and A. Schorr. On shortest paths in polyhedral spaces. *SIAM Journal on Computing*, 15(1):193–215, 1986.
- [Zigel01] G. Zigelman, R. Kimmel, and N. Kiryati. Texture mapping using surface flattening via multi-dimensional scaling. *Accepted to IEEE Trans. on Visualization and Computer Graphics*, 2001.