Path Planning on Quadric Surfaces and Its Application

Chi-Chia Sun, Gene Eu Jan, Chaomin Lu and
Kai-Chieh Yang

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.72573

Abstract

In this chapter, recent near-shortest path-planning algorithms with $O(n\log\ n)$ in the quadric plane based on the Delaunay triangulation, Ahuja-Dijkstra algorithm, and ridge points are reviewed. The shortest path planning in the general three-dimensional situation is an NP-hard problem. The optimal solution can be approached under the assumption that the number of Steiner points is infinite. The state-the-art method has at most 2.81% difference on the shortest path length, but the computation time is 4216 times faster. Compared to the other $O(n\log\ n)$ time near-shortest path approach (Kanai and Suzuki, KS's algorithm), the path length of the Delaunay triangulation method is 0.28% longer than the KS's algorithm with three Steiner points, but the computation is about 31.71 times faster. This, however, has only a few path length differences, which promises a good result, but the best computing time. Notably, these methods based on Delaunay triangulation concept are ideal for being extended to solve the path-planning problem on the Quadric surface or even the cruise missile mission planning and Mars rover.

Keywords: Delaunay triangulation, Dijkstra algorithm, ridge point, near-shortest path, mission planning, NP-hard

1. Introduction

In the Euclidean plane with obstacles, the shortest path problem is to find an optimal path between source and destination. Shortest path algorithms have already been applied to motion planning of robots and path planning of navigation. Furthermore, it can be applied to electronic design automation (EDA), biological cell transportation and operation research (OR) [1–3].

In [4–6], Jan et al. proposed two $O(n\log\ n)$ time path-planning algorithms to obtain the near-shortest path in the Euclidian and quadric planes, respectively. Compared to the other approaches of reduced visibility graph, this fast method outperforms the rest of $O(n\log n)$ algorithms in the general two-dimensional situation, except the path length compared to the shortest $O(n^2)$ time shortest algorithm of visibility graph.

In the quadratic plane, a survey of the shortest path problem concerning a two or higher dimensional geometric object (e.g. a surface, a polyhedron, space, network) can be found in [7]. The shortest path problem in the general three-dimensional situation is non-deterministic polynomial-time hard (NP-hard) problem [8], and only exponential time algorithms are known. In [9], the shortest path on a polyhedron is its local, which has an important property called unfolding, where the path must enter and leave at the same angle to the intersecting edge.

It follows that any locally optimal shortest path joining two consecutive obstacle vertices can be unfolded at each edge along its edge sequence, thus obtaining a straight segment. Sharir and Schorr [10] proposed an $O(n^3\log n)$ algorithm, which first applied this property to find the exact shortest path on a convex surface, where n is the number of edges. Later, Mitchell et al. [11] proposed an $O(n^2\log n)$ algorithm for propagating the shortest path map over a surface by a continuous Dijkstra method for general polyhedron. Chen and Han [12] improved it to an $O(n^2)$ algorithm. Faster algorithms than these cannot be found by far.

Kimmel and Sethian [13] presented a fast searching method for solving the Eikonal equation on a rectangular orthogonal mesh in $O(M\log M)$ steps, where M is the total number of grid points. They extended the fast marching method to triangulated domains with the same computational complexity. As an application, they provide an optimal time algorithm for computing the geodesic distances and thereby extracting shortest paths on triangulated manifolds.

Helgason et al. [14] presented a heuristic algorithm based on geometric concepts for the problem of finding a path composed of line segments from a given destination in the presence of polygonal obstacles. The basic idea involves constructing circumscribing triangles around the obstacles to be avoided. Their heuristic algorithm considers paths composed primarily of line segments corresponding to partial edges of these circumscribing triangles and uses a simple branch-and-bound procedure to find a relatively short path of this type.

Kanai and Suzuki proposed a near-shortest path approach (Kanai and Suzuki, KS's algorithm [9]) based on the Delaunay triangulation, the Dijkstra algorithm, and Steiner points, with computational complexity of $O(k^2n\log k^2n)$, where k is the number of Steiner points and n is the number of the triangles. Although KS's algorithm is an approximation, it has the significant advantages of easy implementation, high approximation accuracy, and numerical robustness. However, to obtain a shorter path, the computation time required by the path planning will increase rapidly when the Steiner points increases. A detailed comparison can be found in **Table 1**.

In this chapter, an $O(n \log n)$ time near-shortest path planning is introduced. It combined with the Delaunay triangulation, Ahuja-Dijkstra algorithm, and ridge points for path planning on a quadratic surface. Experimental results show that the average path length of the Delaunay triangulation-based algorithm is 0.28% longer than the KS's algorithm; however, the speed is 31.71 times faster. Furthermore, when performing KS's algorithm with 29 Steiner points, the NP-hard shortest path will be found (extremely close approximation of the shortest path planning). Although the length is 2.81% longer than the shortest, the computation time is

Euclidean	**Algorithms**				
	Polyhedron			**Polyhedron or quadric**	
	Sharir [10]	**Mitchell [11]**	**Chen [12]**	**KS's [9]**	**Delaunay method**
Connection	No	No	No	$6k^2n$	$9n$
Time complexity	$O(n^3\log n)$	$O(n^2\log n)$	$O(n^2)$	$O(k^n\log k^2n)$	$O(n\log n)$
Is the path shortest?	N/A	N/A	N/A	Near-shortest	Near-shortest

Table 1. Comparison of different shortest path algorithms in the three-dimensional space, $\boldsymbol{n}$ denotes the number of triangle mesh.

4216 times faster. Therefore, it can not only obtain a good near-shortest path length on the quadratic surface, but also improve the computation time. Furthermore, it is worth noting that Delaunay triangulation-based fast algorithms are ideal for being extended to solve the path planning in the polyhedron plane or be applied to cruise missile mission planning in the quadratic plane.

This chapter is organised as follows. Section II briefly introduces the concept of shortest path algorithms. In Section III, we will describe the idea of the triangulation-based near-shortest path algorithm, the performance of which is analysed in Section IV. The experimental results are shown in Section V. Section VI explains a possible application for cruise missile mission planning, and Section VII concludes the chapter.

2. Algorithm backgrounds

In this section, basic concepts of the Delaunay triangulation algorithm on the quadratic surface will be introduced, such as Euclidean plane, Delaunay triangulation, ridge points, and Dijkstra's single-source shortest path algorithm. Euclidean space is the Euclidean plane and three-dimensional space of Euclidean geometry [15], as well as the generalisations of these notions to higher dimensions. It can be used to distinguish these spaces from the curved spaces of non-Euclidean geometry and Einstein's general theory of relativity [16].

For the quadratic surface, there is essentially only one Euclidean space with three real number coordinates from the modern viewpoint.

A Quadric surface is the locus of the points (x, y, z), which satisfy a second-degree equation in three variables, $Ax^2 + By^2 + Cz^2 + 2Dxy + 2Exy + 2Fyz + Gx + Hy + Iz + J = 0$, and the different types of Quadric surfaces, a total of 15, are obtained by varying the coefficients of it [17].

The classification of the different types of Quadric surfaces is made, first, on the basis of the matrix of the quadratic from determining by the symmetric matrix:

$$A_Q = \begin{pmatrix} A & D & E \\ D & B & F \\ E & F & C \end{pmatrix} \tag{1}$$

A triangle mesh is a type of polygon mesh in computer graphics. It comprises a set of triangles (typically in three dimensions) that are connected by their common edges or corners [18]. With individual triangles, the system has to operate on three vertices for every triangle. In mathematics and computational geometry, the triangle mesh can be expressed in a Delaunay triangulation for a set V of vertices in the plane is a triangulation DT(V) such that no vertex in V is inside the circle of any triangle in DT(V) [19].

A path that interconnects the two vertices v and v' of a graph G with minimal length for all paths is called the shortest path. Finding a shortest path in a graph G can be done in $O(n\log n)$ with Ahuja-Dijkstra's single-source shortest path algorithm by using Fibonacci heaps (F-heaps) and radix heaps [20].

A ridge is a curve consisting of ridge point: A point lies on a ridge if its neighbourhood can be subdivided by a line passing through it, and such that the surface in each half-neighbourhood is monotonically decreasing when moving away from the line [21].

3. Algorithm and illustration

In this section, a Delaunay triangulation-based method will combine the concepts of Ahuja-Dijkstra algorithm and ridge points to construct a directed graph and to obtain the shortest possible path length on the quadratic surfaces. Compared to another Delaunay triangulation method [4], Fermat points are replaced by the ridge points; this is mainly due to the fact that Fermat points cannot connect the shortest line between two neighbour triangles on the quadratic surface. The initial step of the algorithm is to build a triangle mesh G to simulate the earth's surface (the GIS map). Next, a source point and a destination point are spotted, then three ridge points in the same triangle will be connected together to generate an extra small triangle and three extra path segments between the vertices and neighbour triangles diagonal vertices.

These extra connections as shown in **Figure 1(b)** will be used to search for the near-shortest path by using the Ahuja-Dijkstras algorithm (expressed by E_f). As we have constructed the directed connected graph $G' = G \cup E_f$, we can obtain the near-shortest path P in graph G' if it exists.

Function 1: *FindingExtraConections*():

Step 1. Create a ridge point between two neighbouring triangles as shown in **Figure 1(a)**.

Step 2. Obtain the shortest path by connecting these two vertices E, F with the ridge point.

END {Function of *FindingExtraConections*}.

Function 2: *PathShortening*(P):

Step 1. Generate the shortcuts of any two consecutive segments by performing the Function 1.

Step 2. Sort the shortcuts by their corresponding length improvements in a descending order.

Step 3. Shorten the original path in a descending order.

END {Function of *PathShortening*}.

Algorithm: The triangle mesh-based shortest path on the quadric surfaces.

Init Load the data from a GIS map.

Step 1. Construct a triangle mesh *G* by the Delaunay triangulation on the data, locate a source point and a destination point.

Step 2. Compute the shortest path between the neighbouring vertices based on the ridge points on the quadric surface by performing Function 1.

Step 3. Construct the directed connected graph G' with the extra connections.

Step 4. Obtain the shortest path in graph G' if it exists by Ahuja-Dijkstras algorithm.

Step 5. Call the Function 2 *PathShortening*(*P*).

END {Algorithm of the triangle mesh-based shortest path}.

Figure 2 illustrates the detailed process. **Figure 2(a)** shows the initiation of a triangle mesh *G*, then a source point *S* and a destination point *D* are depicted in **Figure 2(b)**. Next, ridge points will be inserted into the triangle mesh in order to generate extra path connections

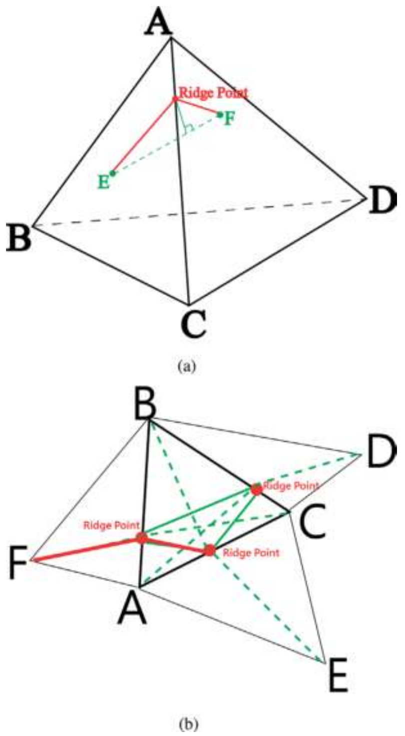


Figure 1. Illustration of the ridge points. (a) A ridge point of ΔABC and ΔACD. (b) The connections between the ΔABC and neighbour triangles vertices and ridge points.

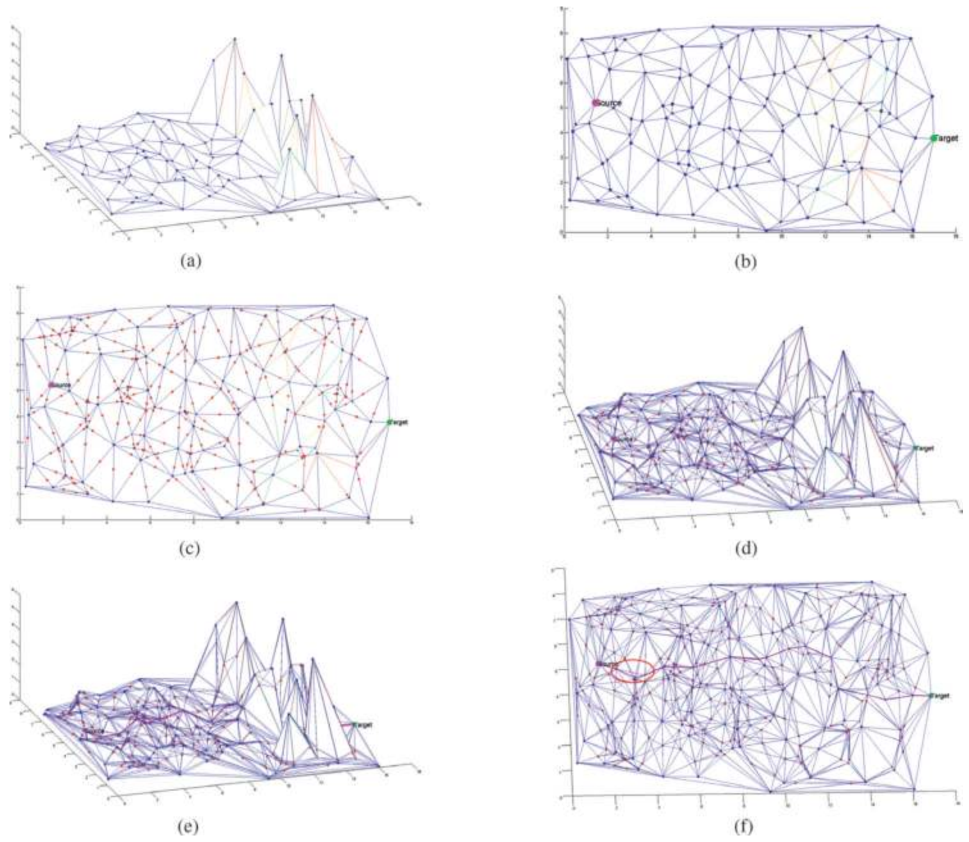


Figure 2. Illustration of the Delaunay triangulation algorithm. (a) Initialise (aerial view). (b) Spot source point and destination point (top view). (c) Insert ridge points (top view). (d) Extra connections (aerial view). (e) Obtain the shortest path (aerial view). (f) PathShortening (top view).

as shown in **Figure 2(c)**. **Figure 2(d)** shows that each ridge point will connect with the others in the same triangle and three extra path segments between the vertices and neighbour triangles diagonal vertices. Thereafter, the shortest path can be obtained in the graph G' by Ahuja-Dijkstras algorithm presented as a red line shown in **Figure 2(e)**. Finally, **Figure 2(f)** shows the final result after the *PathShortening*.

4. Performance analysis

A near-shortest path algorithm on the Quadratic surface is the fastest in the literature.

Theorem 1. The time complexity of the algorithm in the triangle mesh G is $O(n\log n)$, where n denotes the number of triangles.

Proof: We can generate Delaunay triangulation as a triangle mesh with time of $O(n\log n)$ [22], as a result, time complexity in Step 1 is $O(n\log n)$ [23].

The number of the ridge points is bounded by $O(n)$; thus, the number of connections in Step 3 is also bounded by $9 \times O(n)$. We therefore know that all the time complexity for Steps 2 and 3 are $O(n)$. In Step 4, the time complexity of the Ahuja-Dijkatras algorithm using Fibonacci heaps and radix heaps is $O(n + n\log n) = O(n\log n)$ [20]. In Step 5, as the number of points are $n - 3$, the time complexity will be dominated by $O[(n - 3)log(n - 3)] = O(n \log n)$. Therefore, the overall time complexity for *PathShortening* is $O(n\log n)$.

In conclusion, the time complexity is $9 \times O(n) + O(n \log n) = O(n \log n)$ from Steps 1–5.

Theorem 2. The space complexity of constructing the triangle mesh in the quadratic surface is $O(n)$, where n is the number of triangles.

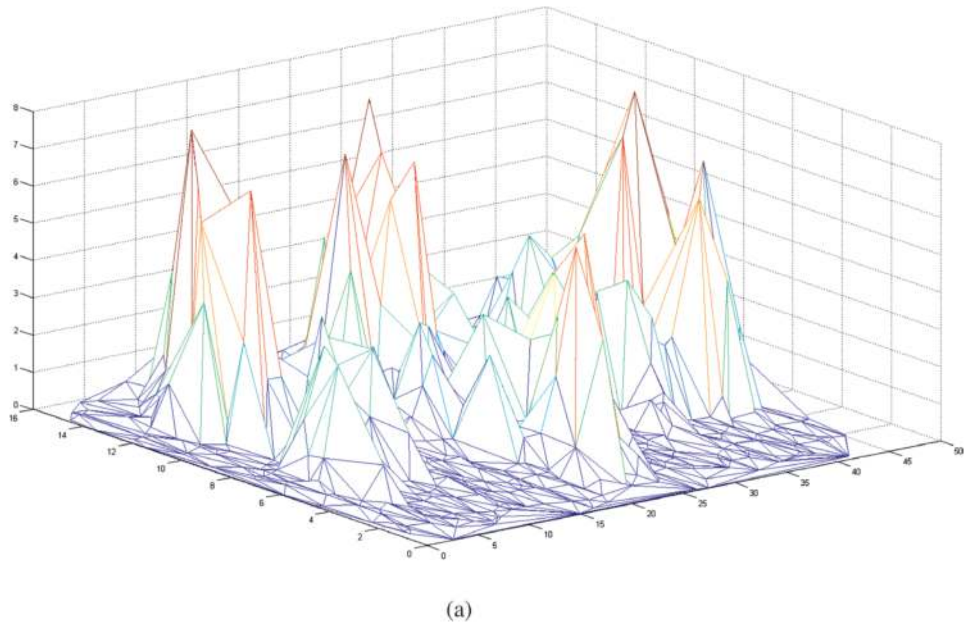
Proof: According to Euler characteristic, the number of triangles is less than $T = 2 \times (k_C \times n) - h - 2$ once the triangle mesh is generated, where h is the number of corners of the triangles. Therefore, the space complexity is bounded by $O(n)$. Furthermore, the number of edges including the original edges of triangles, edges connecting ridge points to the vertices of triangles, and connection between ridge points is also bounded by $(6 + 3/2) \times T = 7.5\ T$. Hence, the space complexity is $O(n)$.

5. Experimental result

The performance of Delaunay triangulation-based path algorithm has been analysed for evaluating the near-shortest path with several real GIS maps in the Matlab Language. The analysis was performed on an Intel Core2 Quad CPU Q9550@2.83 GHz processor with 8 GB memory. **Figure 3** shows one of the experimental results with a GIS map, where the solid line is the near-shortest path and dashed lines are the shortcuts.

Next, we have compared this algorithm to the KS's algorithm with 1, 3, 5, 7, 9, 19 and 29 Steiner points and summarised the comparison results on the average path length and the average runtime in **Table 2**. In KS's algorithm, each edge of the triangle has been divided into multiple segments to generate more connections for path searching. **Figure 4(a)** and **(b)** illustrates the average running time and path length between two algorithms.

When compared to one Steiner point, the average path length difference of the Delaunay triangulation-based algorithm is 6.14% better than the KS's algorithm, and computation time between the Delaunay triangulation-based algorithm and the KS's algorithm is same. When it increased three Steiner points, the length difference is only 0.28%, but the computation time is 31.71 times faster. When 29 Steiner points for the KS's algorithm are applied, the KS's results can be assumed as the shortest path; however, the length difference is 2.81% longer and computation time is 4216 times faster. This proves that the Delaunay triangulation-based algorithm can solve the NP-hard problem and also obtain fast computing features. From the statistical view, **Figure 5** shows the prediction of the average computation time and length difference if the number of KS's Steiner points is infinity.



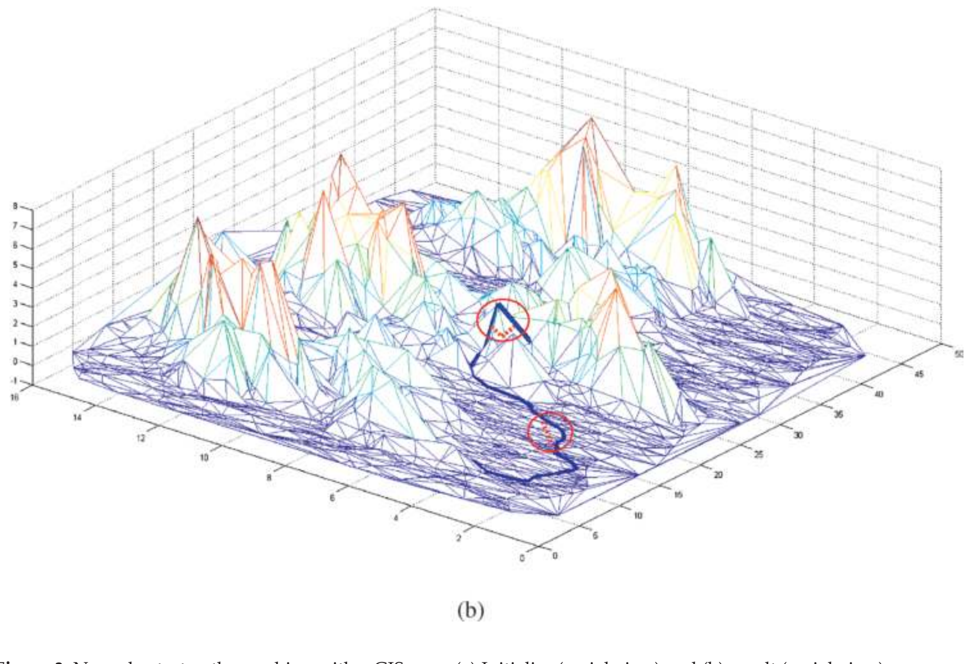


Figure 3. Near-shortest path searching with a GIS map. (a) Initialise (aerial view) and (b) result (aerial view).

SPs	Length difference (%)	Runtime difference (X) (%)
1	−6.14	0.97
3	0.28	31.71
5	1.66	86.40
7	2.26	162.94
9	2.68	414.55
19	2.79	1968.62
29	2.81	4215.75
999 ≅ ∞	5.3	3.0E + 10

Table 2. Comparisons between our algorithm and KS's algorithm on average running time and length difference when the Steiner points are 1, 3, 5, 7, 9, 19, 29, …, ∞

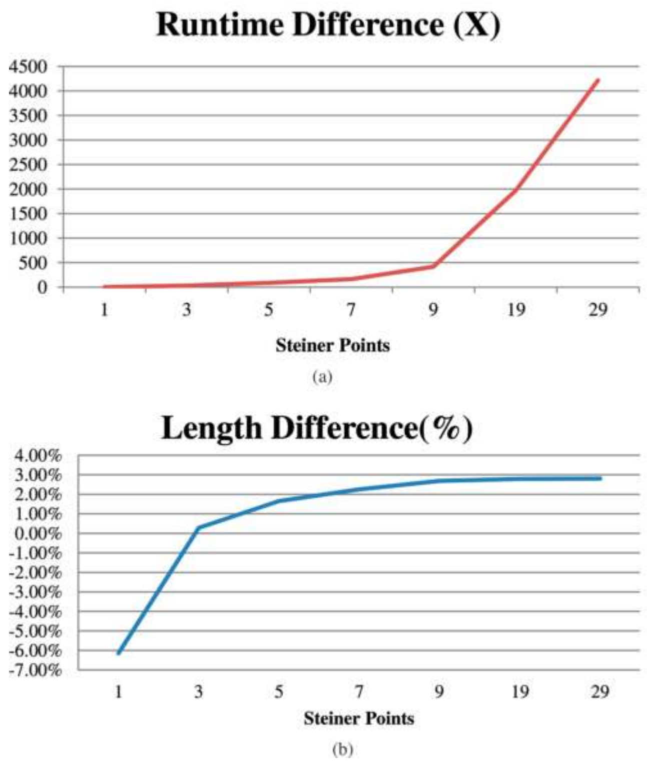


Figure 4. Comparison between Delaunay triangulation-based algorithm and KS's algorithm on average running time and path length. (a) Average computation time and (b) average length difference.

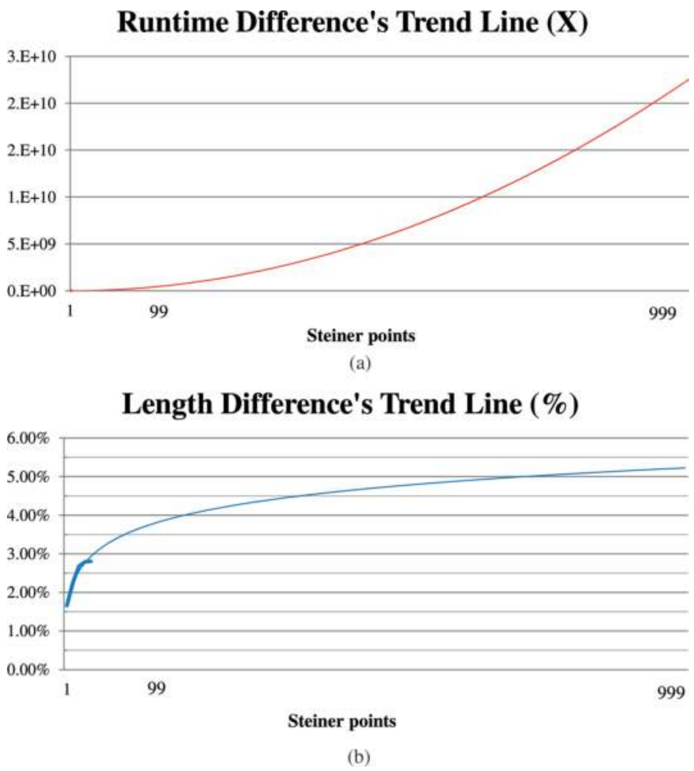


Figure 5. The prediction of the average computation time and length difference if the number of KS's Steiner points is infinity. (a) Average computation time and (b) average length difference.

6. The shortest path application on the quadric surface

This section explains an application that benefits from the Delaunay triangulation-based algorithm. Actually, it can be applied to shortest path planning for Mars rover and mission planning for cruise missiles in the quadric surface. For cruise missile mission planning, we steady up with the angle θ_1 at source point and down with the angle θ_2 at destination point, respectively. Furthermore, the z-coordinate is limited by the l altitude units to avoid the radar's scan as well as crash prevention, where l is a constant, as shown in **Figure 6(a)**. To verify the correctness and performance, we assume a cruise missile needs to move from the source position S to the destination position D, as shown in **Figure 6(b)**. In order to keep the safety margin between the cruise missile and quadric surface, virtual l altitude units are added up to the graph G' (e.g. 20 meters above the G). Once the virtual altitude and thresholds are applied, a shortest path is obtained. Apparently, **Figure 6** shows that this shortest path algorithm can be also applied to intelligently guide the cruise missile to pass a narrow passage and avoid radar's scan.

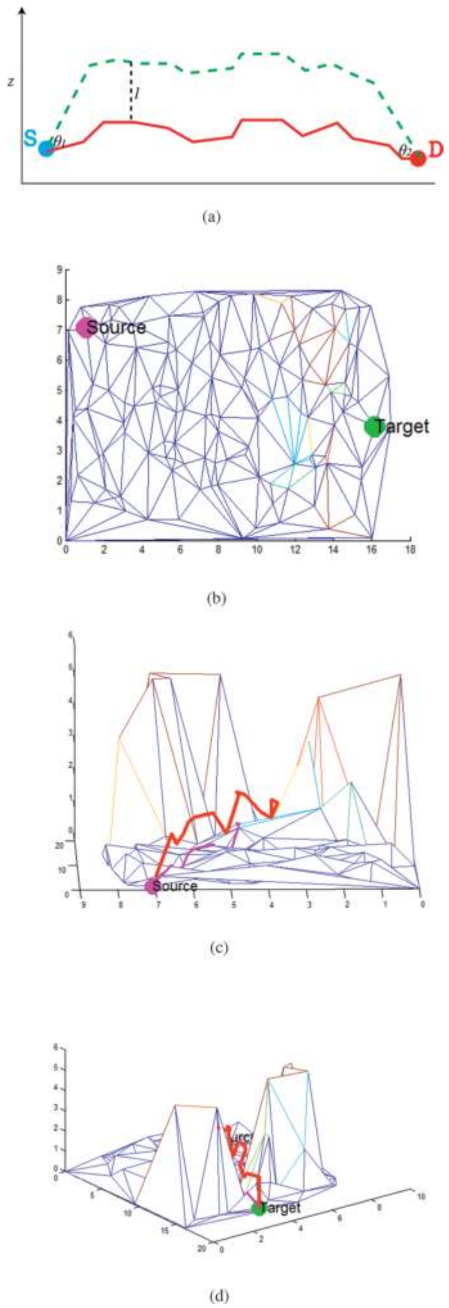


Figure 6. An illustration of the shortest path for planning a cruise missile on the landscape. (a) Cruise missile planning; (b) land scope (top view); (c) result (aerial view 1) and (d) result (aerial view 2).

7. Conclusion

In this chapter, an $O(n\log n)$ time near-shortest path planning based on the Delaunay triangulation, the Ahuja-Dijkstra algorithm, and ridge points on the quadric surface are introduced. Although the length of path obtained by Delaunay triangulation-based algorithm is 0.28% longer than another $O(n\log n)$ time KS's algorithm, the average computation time is 31.71 times faster. Furthermore, when the KS's Steiner point is 29, which means that the shortest path in the NP-hard problem will be obtained, the Delaunay triangulation-based algorithm has at most a 2.81% difference on the path searching, but the computation time is 4216 times faster approximately. Therefore, the Delaunay triangulation-based algorithm presents a good near-shortest path searching solution in the quadric surface with a very short amount of computation time.

Author details

Chi-Chia Sun[1], Gene Eu Jan[2]*, Chaomin Lu[3] and Kai-Chieh Yang[4]

*Address all correspondence to: geneeujan@gmail.com

1 Department of Electrical Engineering, National Formosa University, Taiwan, ROC

2 Tainan National University of the Arts, Taiwan, ROC

3 Department of Electrical and Computer Engineering, University of Detroit Mercy, USA

4 Department of Electrical Engineering, National Taiwan Ocean University, Taiwan, ROC

References

[1] Wu Y, Sun D, Huang W, Xi N. Dynamics analysis and motion planning for automated cell transportation with optical tweezers. IEEE/ASME Transactions on Mechatronics. 2012;**18**(2):706-713. DOI: http://dx.doi.org/10.1109/TMECH.2011. 2181856

[2] Harada K, Hattori S, Hirukawa H, Morisawa M, Kajita S, Yoshida E. Two-stage time-parametrized gait planning for humanoid robots. IEEE/ASME Transactions on Mechatronics. Oct 2010;**15**(5):694-703. DOI: http://dx.doi.org/10.1109/TMECH.2009.2032180

[3] Jan GE, Chang KY, Parberry I. Optimal path planning for mobile robot navigation. IEEE/ASME Transactions on Mechatronics. Aug 2008;**13**(4):451-460. DOI: http://dx.doi.org/10.1109/TMECH.2008.2000822

[4] Jan GE, Sun CC, Tsai WC, Lin TH. An $O(n\log n)$ shortest path algorithm based on Delaunay triangulation. IEEE/ASME Transactions on Mechatronics. Apr 2014;**19**(2):660-666. DOI: http://dx.doi.org/10.1109/TMECH.2013.2252076

[5] Sun CC, Jan GE, Leu SW, Yang KC, Chen YC. Near-shortest path planning on a quadratic surface with $O(n\log n)$ time. IEEE Sensors Journal. Nov 2015;**15**(11):6079-6080. DOI: http://dx.doi.org/10.1109/JSEN.2015.2464271

[6] Jan GE, Fung K, Wu PY, Leu SW. Shortest path-planning on polygonal surfaces with $O(n\log n)$ time. In: IEEE International Conference on Control and Robotics Engineering. IEEEXplore. Apr 2016. pp. 1-5. DOI: http://dx.doi.org/10.1109/ICCRE.2016. 7476149

[7] Mitchell JSB. The Geometric Shortest Paths and Network Optimization in the Handbook of Computational Geometry. North Holland: Elsevier Science; 1998

[8] Canny J, Reif J. New lower bound techniques for robot motion planning problems. In: Annual Symposium on Foundations of Computer Science. IEEEXplore. 1987. pp. 49-60. DOI: http://dx.doi.org/10.1109/SFCS.1987.42

[9] Kanaia T, Suzuki H. Approximate shortest path on a polyhedral surface and its applications. Computer-Aided Design. Sep 2001;**33**(11):801-811. DOI: https://doi.org/10.1016/S0010-4485(01)00097-5

[10] Sharir M, Schorr A. On shortest paths in polyhedral spaces. SIAM Journal of Computing. 1986;**15**:193-215. DOI: http://dx.doi.org/10.1137/0215014

[11] Mitchell JSB, Mount DM, Papadimitriou CH. The discrete geodesic problem. SIAM Journal on Computing. 1987;**16**(4):647-668. DOI: https://doi.org/10.1137/0216045

[12] Chen J, Han Y. Shortest paths on a polyhedron. In: ACM Symposium on Computational Geometry. ACM Digital Library. 1990. pp. 360-369. DOI: http://dx.doi.org/10.1145/98524.98601

[13] Kimmel R, Sethian JA. Computing geodesic paths on manifolds. In: Proceedings of the National Academy of Sciences on Applied Mathematics. PNAS Online. July 1998;**95**:8431-8435

[14] Helgason R, Kennington J, Lewis K. Cruise missile mission planning: A heuristic algorithm for automatic path generation. Journal of Heuristics. Sep 2001;**7**(5):473-494. DOI: http://dx.doi.org/10.1023/A:1011325912346

[15] Byer O, Lazebnik F, Smeltzer DL. Methods for Euclidean Geometry. Washington D.C. USA: Mathematical Asso-ciation of America; 2010

[16] Einstein A. Die Feldgleichungen der Gravitation. Sitzungsberichte der Preussischen Akademie der Wissenschaften zu Berlin. 1915;**48**:844-847. DOI: http://dx.doi.org/10.1002/3527608958.ch5

[17] Galarza R, Irene A, Seade J. Introduction to Classical Geometries. Berlin, Germany: Springer; 2007. DOI: http://dx.doi.org/10.1007/ 978-3-7643-7518-8

[18] Jin J. Three Novel Algorithms for Triangle Mesh Processing: Progressive Delaunay Refinement Mesh Generation, MLS-based Scattered Data Interpolation and Constrained Centroid Voronoi-based Quadrangulation. IL, USA: UMI Dissertation Publishing; 2011

[19] Cheng S-W, Dey TK, Shewchuk J. Delaunay Mesh Generation. UK: Chapman and Hall/CRC; 2012

[20] Ahuja R, Mehlhorn K, Orlin J, Tarjan R. Faster algorithms for the shortest path problem. Journal of the ACM. Apr 1990;**37**:213-223. DOI: http://dx.doi.org/10.1145/77600.77615

[21] Sack J, Urrutia J. Handbook of Computational Geometry. North Holland: Elsevier; 1999

[22] Fortune S. A sweepline algorithm for voronoi diagrams. In: Proceedings of the Second Annual ACM Symposium on Computational Geometry. Berlin, Germany: Springer-Verlag; 1986. pp. 313-322. DOI: https://doi.org/10.1007/BF01840357

[23] Rohnert H. Shortest paths in the plane with convex polygonal obstacles. Information Processing Letters. 1986:**23**(2):71-76. DOI: http://dx.doi.org/10.1016/0020-0190(86)90045-1