A Simple Sweep-line Delaunay Triangulation Algorithm

Liu Yonghe*1, Feng Jinming2, Shao Yuehong3

1Institute of Resources and Environment, Henan Polytechnic University, Jiaozuo 454000, P. R. China
2Key laboratory of Regional Climate-Environment Research for Temperate East Asia, Institute of Atmospheric Physics, Chinese Academy of Sciences, Beijing 100029, P. R. China
3Applied hydrometeorological research institute, Nanjing University of Information Science and Technology, Nanjing 210044, P. R. China

*1 sucksis@163.com

***Abstract*-A simple sweep-line algorithm for constructing 2D Delaunay triangulation is proposed in this paper. It includes following steps: sort the given points according to x coordinates or y coordinates, then link a point each time into the triangulation sequentially, and legalize the new triangles using Lawson's method. Each point is processed by constructing triangles linking the point with all valid edges on the border - the convex hull of points passed by the sweep-line. The border edges are collected using a linked list and the point in it can be searched without consuming much time, since usually only 20-30 edges are in it. The algorithm is faster than incremental insertion algorithm and Fortune's sweep-line algorithm. It is slower than the Zalik's sweep-like algorithm, but is simpler and easier to implement, and can be regarded as another alternative choice to the Zalik's algorithm or the divide-and-conquer algorithm.**

Keywords- Delaunay Triangulation; Sweep-line Algorithm; Digital Elevation Models; Triangulated Irregular Networks

I. INTRODUCTION

Planar Delaunay triangulation is very useful for creating artificial terrains, which are generally called triangulated irregular networks (TIN) in the field of GIS. Nowadays, remote sensing technologies are rapidly producing large amount of surveyed spatial data, which need to be visualized and analyzed using techniques of TIN or other digital elevation models. Therefore, it is very important to manipulate these large datasets efficiently and correctly. There are many algorithms proposed for constructing Delaunay triangulations based on given planar and randomly distributed points, and they are generally classified into several groups: advancing front algorithms [1, 2], incremental insertion algorithms [3-8], gift-wrapping algorithms [9], divide and conquer algorithms [10, 11] and sweep-line algorithms [3, 12] . Except for some algorithms like advancing front method and the Fortune's sweep-line method, many other methods are based on the idea proposed by Lawson: split the points covered area into any non-overlap triangulation firstly and then transform it into Delaunay triangulation by applying Lawson's local optimization procedure (LOP, or named as legalization) [13]. Incremental insertion algorithms are the earliest algorithms applying LOP procedure. In these algorithms, searching the triangle containing the selected point is very time-consuming. Nevertheless, there are some auxiliary approaches like the walking method [14] or DAG tree [6] that are proposed for accelerating the task. Divide-and-conquer algorithms are usually faster than other classical algorithms, but they need complex procedures for merging the sub-triangulations, and the triangles still need to be optimized using Lawson's method.

The first and well known sweep-line algorithm for constructing planar Delaunay triangulation as well as Voronoi diagrams is invented by Fortune [12]. A faster algorithm is proposed by Zalik [3], and an algorithm of sweep-circle version is proposed by Ahmad [15].

A new algorithm will be presented in this paper and it is compared mainly with Zalik's one in this paper. Similar to Zalik's one, it is also based on the sweep-line paradigm combined with Lawson's recursive local optimization procedure, but it is simpler, although slower than Zalik's algorithm.

II. SWEEP-LINE APPROACH

The idea of sweep-line approach is that the points in the plane are sorted horizontally (or vertically in many other papers), and then incrementally the next point is extracted each time from the point set and new triangles associated with this point are constructed, which is imagined as a vertically sweep line glides from left to right over the points. The sweep line can only pass one point simultaneously each time. All the points behind the sweep line are linked in the Delaunay triangulation and the remaining points are to be processed. The triangles need to be locally optimized using Lawson's legalization. Fortune designed a way for directly creating triangles satisfying the Delaunay criteria for constructing Voronoi diagram-a dual graph of Delaunay triangulation [12].

Similarly, Zalik presented another sweep-line algorithm [3]. The edges on the outer border of the existing triangulation are divided into two parts: the back hull and the advancing front. When a new point is to be handled (here denoted by p), the edge on the advancing front hit by p's horizontal projection is selected to construct a triangle with p, and then starting from this edge, other neighboring boundary edges that can be triangulated with p need to be searched. Since some long and thin triangles need legalizations, which make the algorithm inefficient, Zalik proposed some heuristic angle criteria.

Zalik's algorithm has an expected time complexity of O (n log n), and is very fast. However, this algorithm is still complex since there are many branch cases that have to be considered and processed using different procedures. For example, the angle thresholds used for avoiding creating tiny triangles and the dividing of advancing front from back hull make the algorithm somewhat complex. Furthermore, the edges in the advancing front must be collected by a linked list combined with a specially designed and uncommon hashtable-like structure. Finally, when all points are linked to the triangulation, some additional triangles have to be supplemented along some concave part of the advancing front to make the final border become the convex hull of the points.

III. THE PRINCIPLES OF THE PROPOSED APPROACH

In the proposed algorithm, the points in a triangle are always recorded in a counter-clockwise (CCW) order, and the edges are defined as directional edges, thus the edges on the border are always oriented in a CCW direction. There is no need to distinguish the border edges into advancing front and back hull, since maintaining the advancing front as in Zalik's algorithm is somewhat complex. Due to the fact that the border edges are CCW oriented, only the edges facing the next point with its right side can be used to construct new triangles with that point. By the sorted horizontal order, the points are progressively linked into the triangulation, and it can be proved that the new triangles constructed by all the satisfying edges with the point will not intersect any other hull edges, so no intersection tests are needed. By searching all such satisfying edges fit to the new point on the border hull and constructing triangles directly, the point can be linked into the triangulation correctly. After each point is triangulated, all the boundary edges facing the point with their right side are linked to that point. This guarantees the border always keep to be the convex hull of the point set. The above idea is very simple.

The idea of abandoning the distinguishing of advancing front from convex back-hull as in Zalik's algorithm definitely enlarges the searching scope of satisfying border edges. Theoretically, in the worst cases the scope can be very large, but in practice it never becomes too large, since the border is always convex and is comprised by a small number of edges. In most cases, the total number of boundary edges is nearly 30 for a dataset close to 106 points, as convinced by many experiments in this study, which is far less than the number of edges in the advancing front in Zalik's algorithm.

IV. IMPLEMENTATION OF THE ALGORITHM

A. Data Structure

The data structures used by this algorithm include three types: point, triangle and directional edge, only the points and triangles need to be recorded, and the directional edges are only auxiliary. The point type has two members needed: the x coordinate and y coordinate. The triangle type has two array-type members, one for recording the three pointers to its three vertices, another for recording the pointers to its three edge-sharing neighbor triangles. Directional edge type only has two pointer types for its starting vertex and end vertex respectively.

The vertices and neighboring triangles recorded in a triangle record as well as the 3 unrecorded directional edges of the triangle must have an implicit linking relationship. For example, as shown in Fig. 1, the triangle p0p1p2 must have the pointers to points be saved in the array with order p0-p1-p2, and its pointers to neighboring triangles be saved in the array with order T0-T1-T2, where T0 shares the edge p0p1, T1 shares the edge p1p2, and T2 shares the edge p2p0.

All input points are collected by an array, and all generated triangles are saved in a triangle-type array. These two arrays are called 'points list' and 'triangles list', respectively. A bidirectional linked list is used to record the border edges of the triangulation. All edges in the linked list (the hull) are sequentially saved just as their connected sequence in the graph.

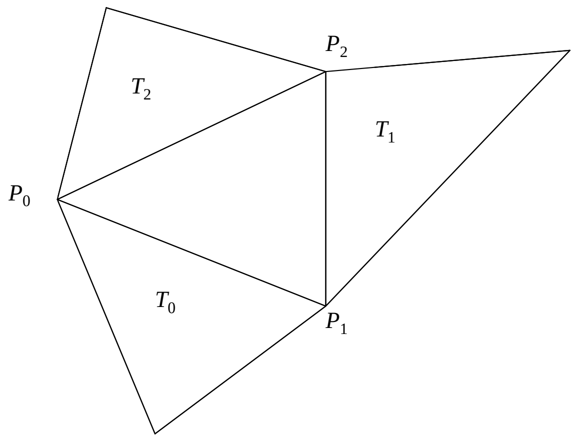

Fig. 1 The recording order and implicit linking relationship of neighbor triangles and vertices with a triangle

B. Initialization

In this paper, it is presumed that the randomly sampled points are sorted according to x coordinate in an ascending manner, thus the beginning of the points list are corresponding to the left side of the spatial area covered by the points. The initialization of the algorithm includes creating the first triangle using the first 3 points in the sorted points list and initializing the convex hull. Since all the point coordinates are added a generated small random number, there is no need to consider the cases of collinear points and coincidental points.

The first 3 points are denoted by p0(x0, y0), p1(x1, y1) and p2(x2, y2). If p2 lies in the left side of the vector p0p1, then the first triangle is constructed as p0p1p2 directly and edges p0p1, p1p2 and p2p0 are added into the hull, otherwise, the first triangle is constructed as p1p0p2 and edges p1p0 , p0p2 and p2p1 are added into the hull (see Fig. 2). In this paper, the side of p0p1 as well as its extended line the p2 lies in can be determined using

$$v=(x2-x0)\ (y1-y0)-(x1-x0)\ (y2-y0). \quad (1)$$

If $v \le 0$, p2 is on the left side of p0p1, otherwise it is on the right side, and there is no need to consider the collinear cases.

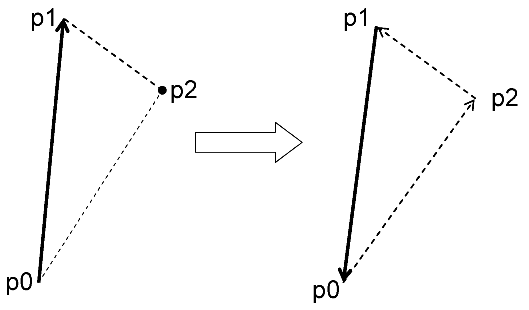

Fig. 2 Initialization of the first triangle

C. Expanding the Triangulation

From the fourth point in the points list, each time one point is linked into the triangulation, until all points are processed, as in Fig. 3.

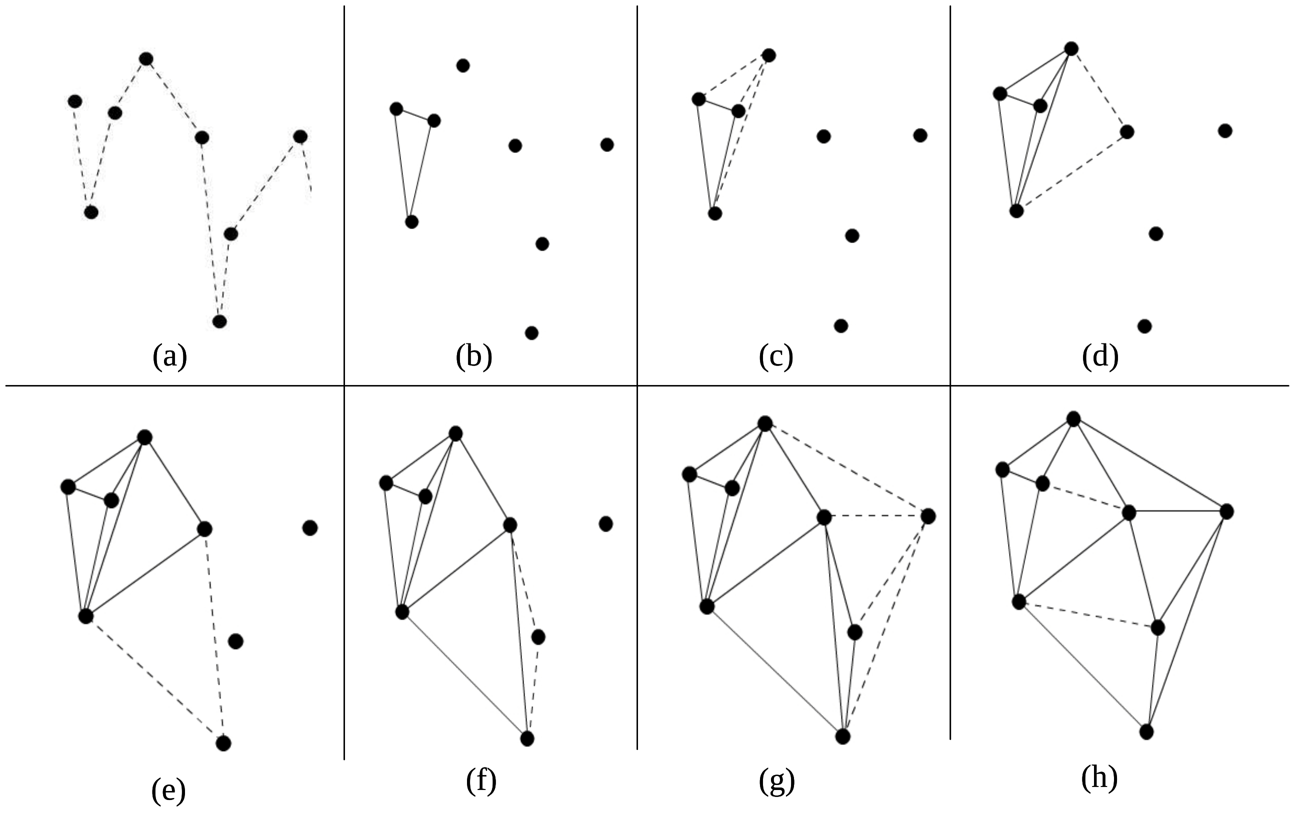

Fig. 3 The process of constructing a triangulation (a): sorting the points regarding the x coordinate, (b)-(g): each time expanding one point from the left triangulation and constructing new triangles, (h): legalizing final triangulation using LOP swaps

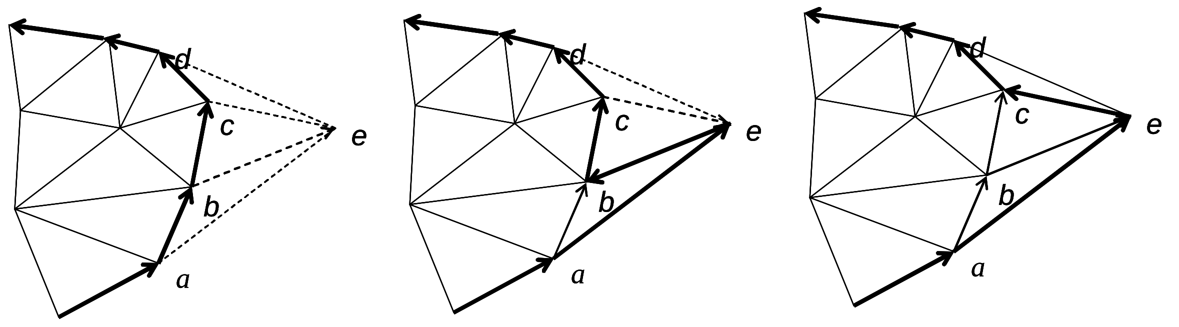

Fig. 4 A case of expanding one point

Once a next point is selected as current point to be linked, all the directional edges in the hull are tested for whether they are facing current point with their right side; if true, the tested edge is selected as the base edge to be linked to current point and correspondingly new triangles are to be created. As shown in Fig. 4, point e is the current point, it lies on the right side of edges *ab*, *bc* and *cd,* thus these 3 edges are used to create triangles with point *e*.

Once a new triangle is constructed, the record of base edge in the hull is to be removed. Correspondingly two new edges linking current point to each vertex of the base edge are created, and then each new edge is to be checked to see whether its reversed edge already exists in the hull; if true, the reversed edge is removed from the hull, otherwise, the new edge is inserted into the hull. As all hull edges are saved in the linked list sequentially and circularly, the above checking of reversing edges can be performed directly; only the precursor or next link of the base edge need be checked.

It should be noted that two new edges associated with a new triangle always are an upper one and a lower one. As shown in Fig. 4(a), for triangle Δ *bae*, the upper edge is *eb* and lower edge is *ae*. Thus, a convenience rule is available: the reverse edge of the lower edge only can be the precursor of the base edge, while the upper edge only can be the next node of the base edge; meanwhile, if no reverse edge is found, the upper edge should be inserted in the hull as the next node of the base edge, the lower edge should be inserted in the hull as the precursor of the base edge. Therefore, such rule can guarantee the operations be performed without any searching.

Fig. 4(b) is used to illustrate the above operations. The triangle Δ *bae* has been completed, the base edge *ab* is removed from the hull, and new edges *ae* and *eb* are inserted. The hull is non-convex but is still circularly connected. Then, triangle Δ*cbe* is constructed with the base edge *bc*. It should be noted that the upper and lower edges are *ec* and *be* respectively, and *be*’s reverse edge-*eb* exists in the hull, so *eb* should be removed from the hull. The upper edge *ec* should be inserted as the next node to *bc*, since *ec* has no reversed edge exists. After *bc* is removed, the hull is completed, as shown in Fig. 4(c).

After each triangle is created, it should be locally legalized with its neighbors. This legalization also needs to spread to other neighbors in a recursive way. This recursive process after creating each triangle can guarantee that all triangles meet the empty circle criteria without using very deep recursions.

D. Avoiding Errors

One common problem encountered when using Lawson’s legalizations is the collinear point, which often gives rise to wrong triangulation, especially when the given dataset is large, many tiny triangles with collinear vertices may exists. There also exist four cocircular points, which have multiple choices for creating different triangulation.

Actually, only under very strict conditions does such error occur, for example when the points’ coordinates are represented by float numbers directly transformed from integer values. In reality, such data errors are elsewhere, but in most times, very small errors are negligible. For the collinear and coincide points, they can be resolved very easily by adding some very small random errors: representing the points coordinate using double precision types, and then adding generated random values smaller than the resolution of the data to the coordinates, thus the occurring probability of collinear three-points in a local space tends to be zero. The small random values added to the point’s coordinates will not influence the final triangulation. The problem of four cocircular points also can be resolved by this method. Furthermore, if a third point is cocircular with the vertices of an existed triangle, it can be regarded out of the circle.

V. ANALYSIS OF THE ALGORITHM

A. Time and Space Complexity

Before the triangulation construction, points must be sorted, which need be done in T1=O(nlogn). When adding each point to the right of the triangulation, the satisfying edges need to be searched on the border, and this searching time can be regarded as O(logn), because the border edges increase very slowly with increasing datasets. Then, for each triangle created, its legalization can be done in O(logn) according to Zalik (2005). Thus the time complexity for the triangulation construction is T2=O(n(logn+logn))= O(nlogn). Then, the total complexity of the algorithm is T1+T2= O(nlogn).

The space complexity is O(n), which is consumed by n points, 2n optional edges, log n border edges and 2n triangles.

B. Theoretical Comparisons with the Zalik's Algorithm

This new algorithm has time complexity of O (n log n) and linear space complexity. In Zalik's method, there are many edges exist in the nonconvex advancing front, and its space complexity is O (n^1/2) (n is the total number of input points). Therefore, if there are one million input points, the edges in the advancing front may be one thousand, which is often between 800 and 1000 when 10^6 points are given, as evidenced by the experiments performed in this study. In the algorithm of this paper, for every swept point, the searching of satisfying base edges needs traverse in the whole convex hull, instead of searching only in the advancing front part as in Zalik's method. The number of edges in the convex border hull is usually smaller than 30 in real applications, even when the points attain one million. When larger datasets are given, the border edges may increase, but it can be imagined that the number will not grow too large.

As proposed by Zalik, to accelerate the searching process of border edges, the advancing front can be implemented using a structure like AVL tree or a hash-table. This actually makes the procedure complex, and meanwhile, the searching in such additional structures will not bring overwhelming advantage comparing with the simple traverses in the whole linked-list hull in the algorithm of this paper.

Moreover, the proposed algorithm does not need managing of the contents of advancing front and the back hull as in Zalik's algorithm. Due to the fact that the advancing front of the triangulation is convex after each point is inserted in the proposed algorithm, there is no need to fill the concave basins on the border as done in Zalik's method.

The main idea of Zalik's algorithm is to minimize the number of calls to the empty circle tests as well as of the Lawson's local optimization procedure (LOP). On the contrary, the LOP is called more often in the proposed algorithm due to the more tiny triangles constructed. Its time complexity is O(log n) for each point insertion, as claimed by Zalik (2005), but it is nearly a constant in reality, since many experiments showed that the number grows very slowly for the proposed algorithm, and in Zalik's algorithm the number also seems a constant. The tiny triangles give the main shortcoming of the proposed algorithm and much time is spent on this procedure, but it is not a big problem if the legalization and diagonal swaps are improved.

C. Theoretical Comparisons with Other Algorithms

This algorithm is also similar to the divide-and-conquer algorithm without alternating recursions between horizontal cuts and vertical cuts, since both of the algorithms construct new triangles without using much calculation or searches comparatively, and both produce many similar tiny triangles requiring legalizations. Except for that it is simpler, the proposed algorithm may have very similar time consumption as the divide-and-conquer algorithm.

A common aspect of the proposed algorithm shared with randomly incremental insertion algorithms is that they both need Lawson's legalizations. The advantage of the proposed algorithm over incremental insertion algorithm is that it needs no searches for the point-containing triangles and no triangle-splitting operations, which are often time-consuming. However, it does not support the dynamical and flexible randomly point insertion supported by incremental insertion algorithms.

D. Experimental Tests of Running Time

The comparisons in this paper were made on the platform of a laptop of Intel i5 (2.53 GHZ), with Windows XP and Microsoft.net 3.5. All the algorithms were implemented in C# language by the authors. Except for the Fortune's algorithm, other algorithms were all based on the same directional-edge based data structures and the same defined types of edges and triangles. The edge-edge and triangle-edge topologies play an important role for maintaining a complete topology structure.

The other algorithms used for comparing with the proposed simple sweep-line algorithm (denoted as SSL) are: 1) A randomized incremental insertion algorithm (denoted as RII) using a 'walking' method; 2) A Fortune's sweep-line algorithm (denoted as F-S), which is implemented by Matt Brubeck at http://www.cs.hmc.edu/~mbrubeck/voronoi.html (2012); 3) A divide-and-conquer algorithm (denoted as D&C) without alternating recursion cuts (denoted as D&C), which is implemented using an efficient merging procedure. The improved Dwyer's version of D&C is not used here since it often behaves not much superior to simple D&C according to run time shown in [4]; 4) A simplified version of Zalik's algorithm (denoted as SZ), which is modified from SSL. The heuristic angle thresholds are reserved but the border edges are only represented by a simple linked list; 5) A completed version of Zalik's algorithms (denoted as CZ), which uses a kind of hashtable as Zalik (2005) proposed; 6) A version of Zalik's algorithm using a C# 'SortedList' structure (denoted as SLZ), instead of using a hash-table; 7) A version of Zalik's algorithm using a modified C# SortedDictionary structure (denoted as SDZ), instead of using a hash-table The SortedDictionary(SD) is a structure based on Red-black Tree, thus its operations are very similar to a AVL tree.

The four versions of Zalik's algorithm were all used for comparison because the choice of data structures for representing the advancing front has important influences on the performance.

The programs were tested multiple times, and the results showed that all of them never failed or create erroneous triangles. A case with one million points is shown in Fig. 5(a). Some LiDAR datasets and DEM datasets were used to test the correctness of the programs, and two examples are shown in Fig. 5(b)-(c).

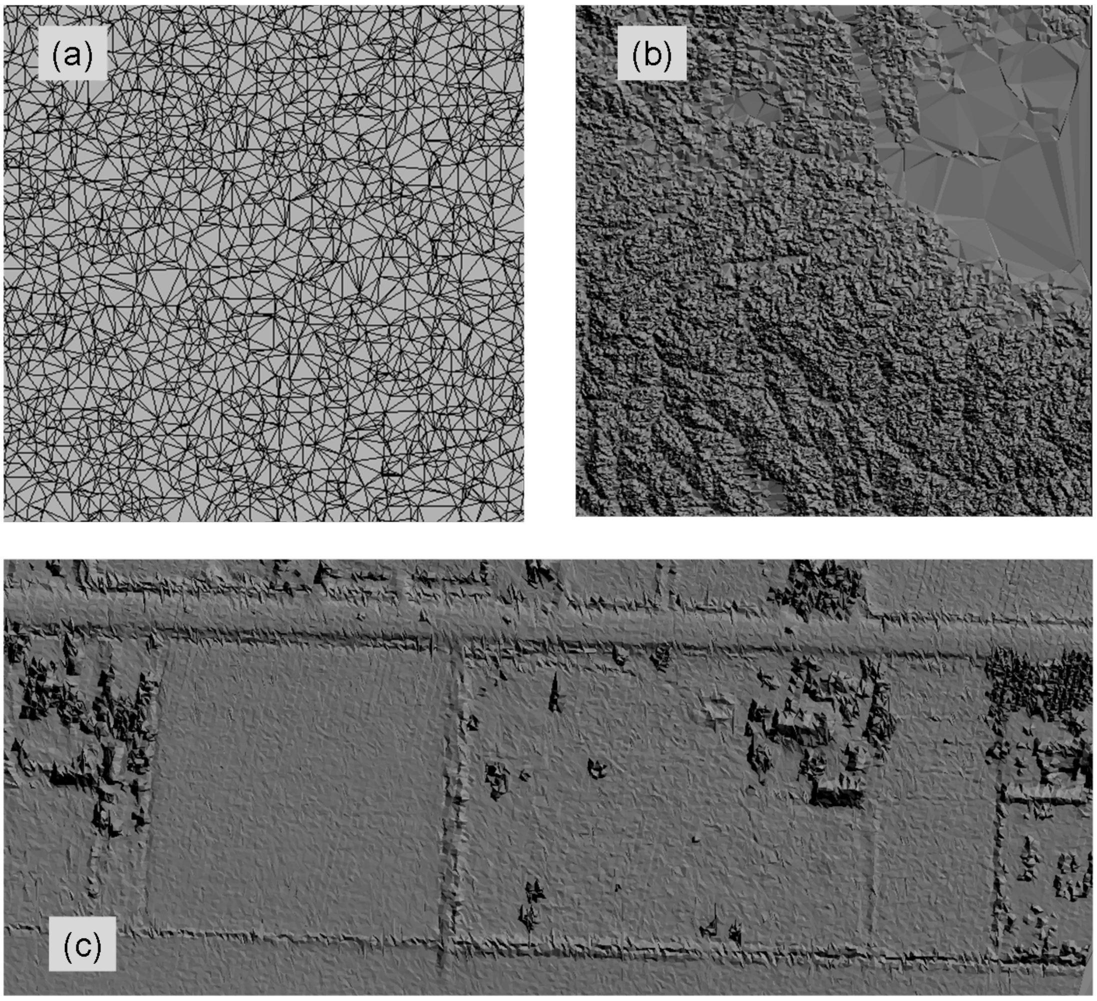

Fig. 5 The examples of using the proposed algorithm: (a) the center part of a triangulation of one million points, (b) the shading effect of a triangulation based on a randomly distributed DEM data contains 70000 points, (c) the shading effect of a triangulation based on a part of LiDAR dataset (http://inside.uidaho.edu/geodata/LiDAR/)

Artificial points with uniform distributions and normal (Gaussian) distributions were used for tests. The CPU time spent is shown in Table 1 and Table 2. For the points generated by the two distributions, these programs give very similar running time. The RII is the slowest, and the F-S is the second slowest. When the dataset contains more than 40,000 points, the time spent by RII is too long and become intolerable. The D&C seems slightly slower than SSL and their running time can be regarded as being equivalent.

For a dataset smaller than 100,000 points, SZ is faster than the SSL, and for larger dataset SSL become faster than SZ. The reason is that the SZ does not implement a structure supporting faster searching for the border hull. Among the different versions of Zalik's algorithms, the SLZ and SDZ are slower than SZ, and SDZ is slower than SLZ. Only when the dataset is larger than 100,000, the advantage of SLZ and SDZ becomes obvious. CZ appears to be the fastest among all the tested algorithms no matter how large the dataset is. For small dataset, the proposed SSL is only faster than SDZ. But for very large dataset, it is only faster than SZ. In summary, even SSL seems slower than most versions of Zalik's algorithm, but the contrasts are not very large.

TABLE 1 THE CPU TIME (S) SPENT FOR ARTIFICIAL POINTS IN UNIFORM DISTRIBUTION

Algorithm	RII	F-S	D&C	SZ	CZ	SLZ	SDZ	SSL
10’000	2.75	0.453	0.125	0.078	0.046	0.093	0.203	0.125
20’000	11.843	1.125	0.281	0.171	0.093	0.140	0.375	0.250
30’000	26.296	1.937	0.484	0.312	0.171	0.250	0.578	0.375
40’000	47.89	3.031	0.656	0.453	0.281	0.328	0.671	0.593
50’000	79.062	4.125	0.875	0.562	0.343	0.484	0.921	0.671
100’000	--	10.890	1.812	1.515	0.937	0.921	1.671	1.468
500’000	--	102	10.890	12.718	5.093	6.156	9.203	11
1’000’000	--	261	24	36.546	11.421	15	19	24
2’000’000	--	--	X	143	28	--	X	57

Note: The symbol ‘--’ denotes intolerably long time and ‘X’ denotes failed tests due to limited memory.

TABLE 2 THE CPU TIME (S) SPENT FOR ARTIFICIAL POINTS IN GAUSSIAN DISTRIBUTION

Algorithm	RII	F-S	D&C	SZ	CZ	SLZ	SDZ	SSL
10’000	3.031	0.390	0.125	0.062	0.031	0.062	0.203	0.093
20’000	12.062	1.046	0.296	0.171	0.093	0.140	0.312	0.250
30’000	27.843	1.796	0.468	0.328	0.140	0.312	0.468	0.390
40’000	58.812	2.781	0.625	0.468	0.265	0.359	0.718	0.515
50’000	111	3.750	0.828	0.656	0.328	0.453	0.937	0.703
100’000	--	10	1.718	1.687	0.703	1.000	1.593	1.484
500’000	--	93	11	15.25	5.343	6.578	8.625	10
1’000’000	--	265	24	43	11.781	15	18	24
2’000’000	--	--	X	145	25.7	34	38	58

Furthermore, as shown in Table 1-2, when the Computer memory becomes a bottleneck, the algorithms perform differently. For example, D&C and SDZ, with large datasets of 2×10^6 points, could not complete the process, but SZ, CZ and SSL did. Thus SSL is also a memory economized algorithm.

E. Experimental Tests of Border Edges and Legalizations

The average number of edges reside in the border hull plays an important role in the comparison of SSL with Zalik’s algorithm. The larger the number of edges in the hull, the longer the time spent on searching. Here, the proposed algorithm was only compared to Zalik’s algorithm. SZ was used as the delegate, since the implementation choice of the Zalik’s algorithm does not influence the number of border edges. Given both datasets of uniform distributions and Gaussian distributions, the average numbers of edges in the hull with increasing datasets are almost the same, as evidenced by the experiments. The case with datasets of uniform distributions is shown in Fig. 6. For SZ, the numbers increase in an n^1/2 speed of the increase of points. With 10,000 points, the number of border edges is 108, while with 10^6 points the number is 924. For the SSL, the numbers in the convex hull are also increased with more points given, but it is in a very slow increasing rate: for 10,000 points, the number is 22, but for 10^6 points, the number is only 30. Thus, its influence on the run time is very small.

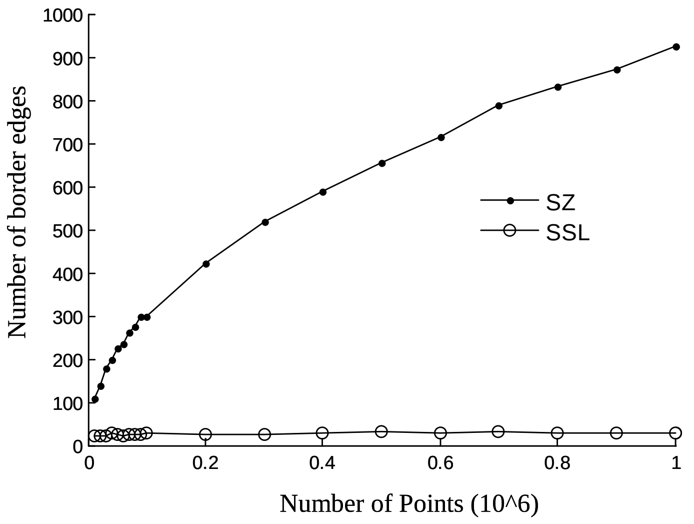

Fig. 6 The numbers of edges in the border hull with different numbers of given points

The execution of Lawson's legalization procedure including empty-circle tests and diagonal swaps in the SSL cost much time due to the tiny triangles. The average number of legalization calls per point is shown in Fig. 7. For SZ, the number of legalization calls per point is almost a constant number of 3.48, which is shown as a horizontal line in Fig. 7, even when the given points attains 10^6. Comparatively, the calls to legalization in SSL are actually more frequent than in SZ: the number ranges from 16.9 to 27.9 with the points increasing from 10^4 to 10^6, and it seems to increase in a logarithmic rate.

The proposed algorithm can be regarded as an implementation of Zalik's sweep-line algorithm without using heuristics for preventing tiny triangles. However, as shown in Table 1, CZ is only 2 times faster than SSL, not as contrastive as the comparison performed by Zalik. This discord may due to the different legalization implementations. Theoretically, if the efficiency of legalizations is improved, the tiny triangles will not become a serious problem.

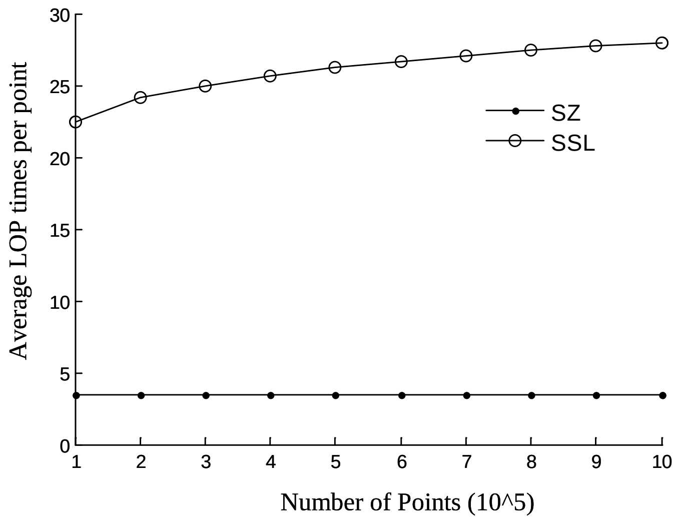

Fig. 7 The average LOP times for adding each point

VI.CONCLUSION

For visualizing large datasets like LiDAR point clouds, a simple, fast and stable algorithm for constructing triangulations from planar points is necessary. A sweep-line algorithm was introduced in this paper, and it can be regarded as a fast algorithm comparatively. It is faster than incremental insertion algorithms, and Fortune's sweep-line algorithm. It is only slower than the Zalik's sweep-line algorithm. It has equivalent running time with divide-and-conquer algorithm, but occupies less memory. It is also very stable as evidenced by the computer-program implementations and the experimental tests.

ACKOWLEDGEMENTS

This study was jointly supported by the General Project of the National Natural Science Foundation of China (Grant No. 41105074 and 41275108), the Open Research Fund of Key Laboratory of Digital Earth Science, Center for Earth Observation and Digital Earth, Chinese Academy of Sciences (grant number: 2011LDE010), and the Scientific Research Fund of Henan

Polytechnic University(grant number: B2011-038). The authors are very grateful to the website http://inside.uidaho.edu/geodata/LiDAR/ for providing free LiDAR dataset, which are used to test the algorithms. The authors are also indebted to the Microsoft Corporation for providing free Visual Studio express software and Dotnet Framework, which are used to implement the algorithms efficiently.

REFERENCES

[1] D. J. Mavriplis, "An advancing front delaunay triangulation algorithm designed for robustness," Journal Of Computational Physics, vol.117, pp. 90-101, 1995.

[2] C. J. Du, "An algorithm for automatic Delaunay triangulation of arbitrary planar domains," Advances in Engineering Software, vol. 27, pp. 21-26, 1996.

[3] B. Zalik, "An efficient sweep-line Delaunay triangulation algorithm," Computer-Aided Design, vol. 37, pp. 1027-1038, 2005.

[4] S. Zhou and C. B. Jones, "HCPO: an efficient insertion order for incremental Delaunay triangulation," Information Processing Letters, vol. 93, pp. 37-42, 2005.

[5] L. J. Guibas, D. E. Knuth, and M. Sharir, "Randomized incremental construction of Delaunay and voronoi diagrams," Algorithmica, vol. 7, pp. 381-413, 1992.

[6] I. Kolingerov and B. Zalik, "Improvements to randomized incremental Delaunay insertion," Computers & Graphics, vol. 26, pp. 477-490, 2002.

[7] B. Zalik and I. Kolingerova, "An incremental construction algorithm for Delaunay triangulation using the nearest-point paradigm," International Journal of Geographical Information Science, vol. 17 pp. 119-138, 2003.

[8] S. W. Sloan, "A fast algorithm for constructing Delaunay triangulations in the plane," Advances in Engineering Software and Workstations, vol. 9, pp. 34-55, 1987.

[9] T. P. Fang and L. A. Piegl, "Delaunay triangulation using a uniform grid," IEEE Computer Graphics and Applications, vol. 13, pp. 36-47, 1993.

[10] D. T. Lee and B. J. Schachter, "2 Algorithms for constructing a Delaunay triangulation," International Journal of Computer & Information Sciences, vol. 9, pp. 219-242, 1980.

[11] R. Dwyer, "A faster divide-and-conquer algorithm for constructing delaunay triangulations," Algorithmica, vol. 2, pp. 137-151, 1987.

[12] S. Fortune, "A sweepline algorithm for Voronoi diagrams," Algorithmica, vol.2, pp.153-174, 1987.

[13] C. L. Lawson, *Software for C1 Surface Interpolation.Mathematical Software III*, New York, USA, Academic Press, 1977.

[14] L. J. Guibas and J. Stolfi, "Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams," Acm Transactions on Graphics, vol. 4, pp. 75-123, 1985.

[15] A. Biniaz and G. Dastghaibyfard, "A faster circle-sweep Delaunay triangulation algorithm," Advances in Engineering Software, vol. 43, pp. 1-13, 2012.