Regular Article

A Complete Method for Reconstructing an Elevation Surface of 3D Point Clouds

Van Sinh Nguyen, Manh Ha Tran, Ba Cong Nhan

School of Computer Science and Engineering, International University, Ho Chi Minh City, Vietnam

Correspondence: Van Sinh Nguyen, nvsinh@hcmiu.edu.vn

Manuscript communication: received 18 December 2014, accepted 8 May 2015

Abstract– **Reconstructing the surface of 3D point clouds is a reconstruction from a cloud of 3D points to a triangular mesh. This process approximates a discrete point cloud by a continuous/smooth surface depending on the input data and the applications of users. In this paper, we propose a complete method to reconstruct an elevation surface from 3D point clouds. The method consists of three steps. In the first step, we triangulate an elevation surface of 3D point cloud structured in a 3D grid. In the second step, we remove the outward triangles to deal with concave regions on the boundary of the triangular mesh. In the third step, we reconstruct this surface by filling the hole of triangular mesh. Our method could process very fast for triangulating the surface, preserve the topology and characteristic of the input surface after reconstruction.**

Keywords– **Surface reconstruction, 3D point cloud, Delaunay triangulation, voxel traversal search, hole filling.**

1 Introduction

Reconstructing the surface of 3D point clouds is one important step in the researched field of geometric modeling. This process includes simplifying, triangulating, holes filling and refining the surface. The computing time of algorithms and the quality of output surfaces are two essential objectives that the researchers are facing. Some existing methods for processing the surface of triangulation have been studied since many years. But for the surface of 3D point clouds, it is still seem a challenge for researchers. In order to save processing time and capacity of memory, some methods aim to reduce the number of 3D point clouds by simplifying [1, 2]. Surface triangulation of a 3D object is a fast and efficient way to reconstruct such an approximating surface [3, 4]. Filling the holes and refining the reconstructed surface are also proposed in [5, 6].

As mentioned in [7], our purpose is to build an optimal geological triangulated surface in order to get the best simulation of the oil reservoir. The problem in our study comes from the seismic data, with a very large number of 3D points (that can reach several millions of points). Therefore, the necessary time to triangulate these data points with "classical" methods can be very high (and hence unacceptable in terms of time and memory). For this reason, the input data are first structured in a sparse 3D volume as defined by Philippe Verney [8]. In order to preserve the shape of the surface, the first step is extracting and simplifying the boundary of the surface [9]. The proposed method for simplifying the surface inside its boundary [10] is an important step for the further meshing step. Indeed, simplification produces smaller data sets whose density varies regularly over the surface. The work in this paper is the next step in which we propose a complete method to reconstruct the surface of 3D point clouds. In the first step, our simplified surface is triangulated by using a fast search algorithm (VTS) that was presented in [11]. In the second step, we propose a method to process the outward triangles on the boundary to reconstruct a part of the generated surface [12]. The third step in this paper, we suggest a method to fill the holes of triangulated surface based on [13] in order to completely reconstruct the output surface.

The remainder of the paper is organized as follows. We first present the existing methods for filling the holes of a triangular mesh in Section 2. Our full method is presented in detail in Section 3. Section 4 is the implementation and results. Discussion and evaluation are presented in Section 5. The last (Section 6) is our conclusion.

2 Related Works

The methods for triangulating the surface of 3D point clouds have been widely studied in the field of geometric modeling and computational geometry. The complexity of algorithms for meshing and the quality of generated triangular meshes are two essential objectives that the researchers are facing. The effectiveness of methods for triangulating a surface of 3D point clouds depends not only on the types of the surface, the input data structure, but also on the characteristics of the data. Most of these methods are classified into three categories: contour tracing approach (called implicit surface approach) [3], region growing [14], and sculpting-based approach (called Delaunay-based approach) [15–17]. They are detailed in [11, 12].

In this section, we study some existing methods for filling the holes of a triangular surface that can be

applied in our research to completely reconstruct our surface. Wu *et al.* [5] described a method to fill the holes automatically by using radial basis function (RBF); this function is also shown in [18, 19]. In the first step, the boundary of the hole is found by extracting the edges which belong to one triangle only (they share only with one triangular face). In the next step, all boundary points of the holes are marked. In the last step, these holes are filled based on the marked interpolation points by using RBF. However, one of drawback of this method is that the projection from 3D to 2D of the object does not contain the fold of surface.

Alexandra *et al.* [13] suggested a method to remesh and fair the holes of a triangular mesh. At first, the hole boundary is identified based on the density of point clouds. This hole is filled in the next step by connecting the boundary points to create new triangles. After that, the created triangles are subdivided by using a partial differential equation scheme in order to restore the local curvature and guarantee the smoothness of the hole boundary. At the end, the hole is repaired by minimizing a discrete thin-plate energy. This method obtains a good quality mesh (reconstructed surfaces are smooth and very close from the initial model), but leads to low running times because of using the multistep approach.

The above methods can be applied for filling the holes of a triangular mesh on both 2D and 3D. In our case, the input data are structured in the 3D grid in voxels. Therefore, we based on this advantage for building a complete method to reconstruct the surface of 3D point clouds. We detail our method in the next section.

3 Methods for Reconstruting the Surface

In this section, we present our full method for reconstructing the surface of 3D point clouds. The input data are structured in the 3D grid in voxels [7]. After simplifying these data [10] to reduce the number of input data, we triangulate this surface by using a fast algorithm to speed up the Delaunay Triangulation [11]. The triangular surface is then reconstructed by removing some outward triangles on the boundary to obtain a model that approaches the input surface [12]. However, the generated surface has some holes that need to fill in order to obtain completely a reconstructed surface. Next, we will describe in detail for each part.

3.1 Triangulating the Surface

As presented in [10], we have simplified the input surface. After simplifying, the point distribution is constrained (with respect to its density) from the boundary to the inside of the surface. Moreover, a large number of points have been removed while keeping the initial shape of the surface. This is one of the important steps for further triangular surface processing, because part of computation in the algorithm depends on the number of input data points. Our method is detailed below.

Our method is based on the 2D Delaunay triangulation of the projected point cloud which has been done in [11]. The elevation surface of 3D point clouds (after simplifying [10]) is first projected onto a natural 2D grid in the x, y plane. Then, we triangulate the surface (actually, we compute a Delaunay triangulation of the 2D point clouds taking advantage of its regular structure). The main novelty of our approach is that the neighboring points are searched in a rectangle supported by the edge e_i (of previous triangle) under consideration. Therefore, our method consists of three steps. In the first step, we create a first Delaunay triangle. Starting from the first Delaunay boundary edge, we find and connect to a neighboring point to create the first triangle. In the second step, we create the next triangle, adjacent to the first triangle based on a criterion of the Delaunay triangle. In the third step, we triangulate the surface by repeating the process from the next triangle. After creating the next triangle, this triangle will become the first triangle for the next iteration.

3.1.1 Building a seed triangle:

The existing methods for creating the first (seed) triangle are introduced in [12]. In our case, we start from the left-most boundary point (p_{b1}) to find the closest neighboring boundary point (p_{b2}) on the boundary. They are connected to create the first Delaunay boundary edge: $e_b(p_{b1}, p_{b2})$. Then, we find a neighboring point p_3^i of e_b such that p_3^i must satisfy the Delaunay criterion. Therefore, the triangle $\triangle(p_{b1}, p_{b2}, p_3^i)$ is our seed triangle: T_{first}. In the previuos work [11], we have also proved that the Delaunay boundary edge always exists in our case.

3.1.2 Creating the next triangle:

In order to create the next triangle for each edge e_i of the previous triangle, we need to find a neighboring point p_3^i. The main novelty of our method is that we apply the voxel traversal search (VTS) [11] to find p_3^i. The advantage of this method is described as follows and illustrate in Figure 1. For each edge $e_i(p_1^i, p_2^i)$, the neighboring point p_3^i is searched in one side of e_i; in a rectangle supported by $e_i(Re(e_i))$, size k starting from voxel which intersect to the line L. Then, the searching process is repeated by extending toward L_1 and L_2 respectively until finding p_3^i. If there is no point p_3^i on the rectangle of edge e_i, we enlarge the searching by iterative dilations of Re(e_i), and stop when a neighboring point p_3^i satisfying the Delaunay criterion is found.

3.1.3 Triangulating the surface:

We start form the first boundary Delaunay edge to create the first Delaunay triangle T_{first}. Then, put all edges of T_{first} into a list "EdgePool". Then, for each edge e_i in the EdgePool, we choose a neighboring point p_3^i based on step (3.2.2) to create a new adjacent triangle T_{next}. Then, we update the EdgePool by inserting the new edges of T_{next}. After that, the triangulating process is repeated from T_{next} by assigning ($T_{\text{first}} \leftarrow T_{\text{next}}$) for the next iteration, until finishing the triangulating process (i.e., the EdgePool is empty). Our method could triangulate a surface of 3D point clouds very fast (see

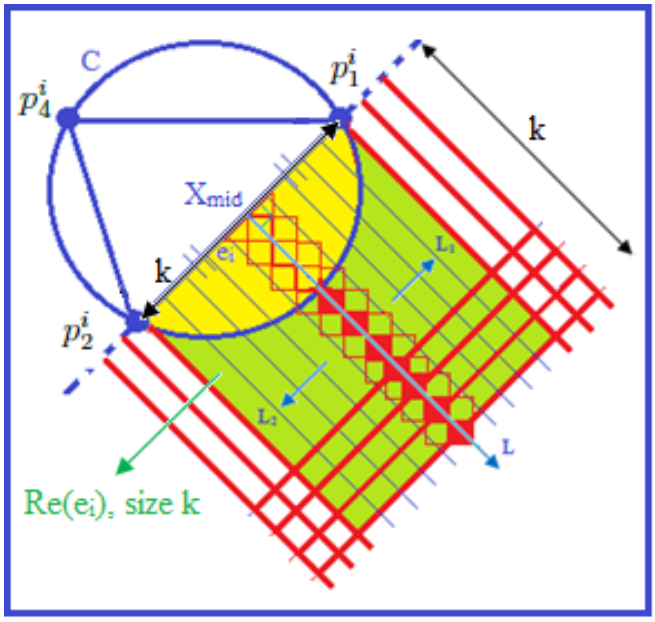

Figure 1. Finding a neighboring point to create the next triangle based on VTS [12].

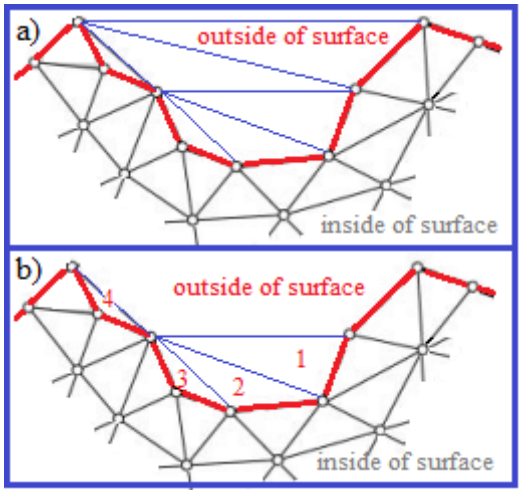

Figure 2. a) The convex hull of a triangular surface; b) The outward triangles.

Section 4). However, depending on the characteristic of the geological data, there still exist some holes. We will process these hole in the next section.

3.2 Removing the Outward Triangles

While triangulating the surface of point clouds, it normally produces a mesh on the convex hull of the initial set of points. As we can see in Figure 2, many triangles lies outside of the boundary (the red color line). In our case, the boundary of the surface has determined and processed in [9]. After triangulating the surface [11], it also generates many undesired triangles which are outside the exterior boundary (see Figure 2(b)). For this reason, we propose a method to delete these triangles (called outward triangles) in order to obtain an optimal triangular surface. We review our proposed method (which have been done in [12]) to remove the outward triangles as follows. In order

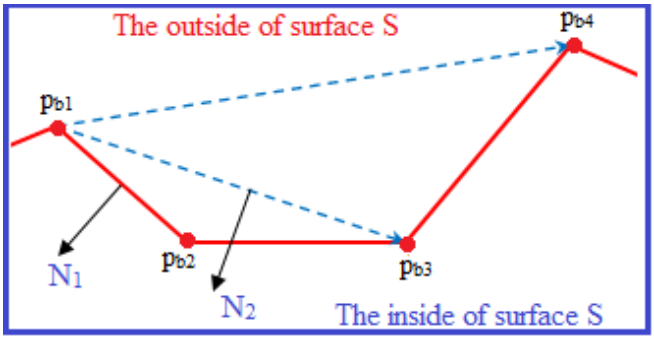

Figure 3. Determination of an outward triangle on the boundary.

to determine the outward triangles, we can base on computing the angle α formed between each pair of consecutive edges on the boundary (e.g., two boundary edges: $e_{b1}(p_{b1}, p_{b2})$ and $e_{b2}(p_{b2}, p_{b3})$; angle α at p_{b2}). If $\alpha < \pi$, the triangle $\triangle(p_{b1}, p_{b2}, p_{b3})$ is considered as an outward triangle. This method of course requires computing the boundary of the point set before meshing. In our case, the boundary has already been determined in [9]. Therefore, in order to compute and remove the outward triangles, we just need to follow the boundary in the clockwise direction and delete the outward triangles accordingly. Our method is described as follows.

We first determine the boundary triangles according to their vertices. For each triangle, if its three vertices are boundary points, it is considered as a boundary triangle. Following the boundary clockwise, we then check and delete the outward triangles. A boundary triangle is an outward triangle if the dot product (denoted as *.dot*) between two vectors of this triangle (an inward normal vector of an edge and a vector of an outside edge) is negative (see Figure 3). The red line is a boundary line of surface S; $p_{b1}, p_{b2}, p_{b3}, p_{b4}$ are boundary points. We check if $((p_{b3} - p_{b1}).dot(N_1) < 0)$ then the triangle $\triangle(p_{b1}, p_{b2}, p_{b3})$ is an outward triangle of S; similarly, if $((p_{b4} - p_{b1}).dot(N_2) < 0)$ then the triangle $\triangle(p_{b1}, p_{b3}, p_{b4})$ is also an outward triangle of S.

3.3 Filling the Holes of Triangular Surface

In this section, we present our method for filling the holes of a triangular mesh. We base on the idea in method [13] to remesh some undersampled areas (called "holes") in a triangular meshes. In fact, our triangular mesh is a surface of geological data. The particularity of such surface is that the 3D points are distributed irregularly. The density is often drastically lesser in some areas than in others: this leads to what we call "holes". We first determine the holes by considering the boundary triangles of the holes. We then fill and triangulate these holes to completely obtain a reconstructed surface.

3.3.1 Determining the boundary of the holes:
In order to determine the boundary of the holes, we base on the previous work [9] to find the boundary points of these holes first (see Figure 4(a): red points). For each boundary triangle around a hole, it is formed

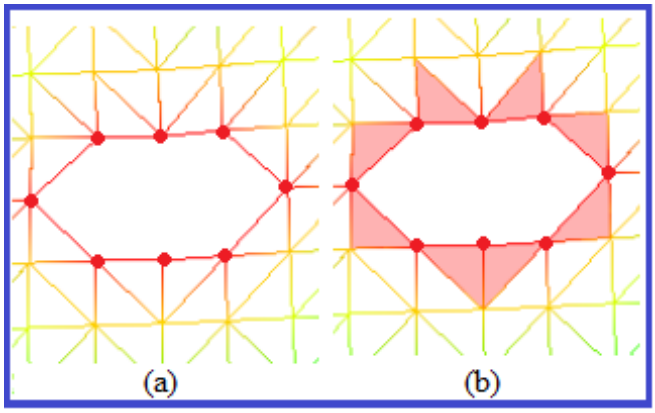

Figure 4. Determining the boundary of a hole.

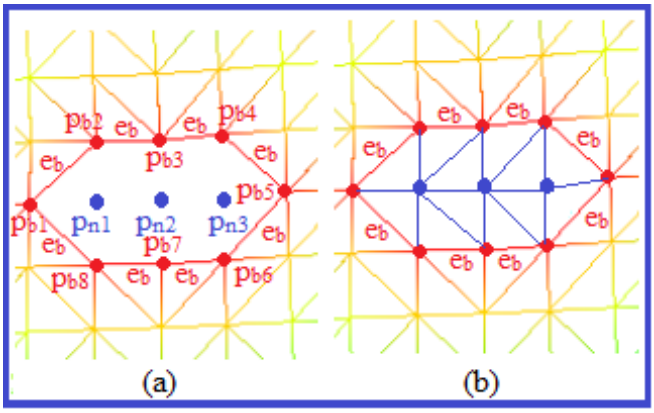

Figure 5. Inserting and triangulating the hole.

by two boundary points (called boundary edge) and one interior point (this is also the different point comparing to the outward triangles: they are formed by three boundary points as pointed out in the part 3.3). This boundary edge only share with one triangular face (see Figure 4(b): red triangular faces).

3.3.2 Filling and triangulating the holes:

After determining the hole of a triangular mesh, we fill by inserting the new points into the hole first; and then, we create a new triangulation between the boundary points of holes and new inserted points for this patch. As mentioned in the introduction, our input data are structured in a 3D grid. Therefore, for each boundary point of the holes we can base on its neighboring points to insert the new points. For each time to insert a new point p_n we compute the distance d from p_n to the nearest boundary points p_b of the holes by

$$d(\|p_n - p_b\|) \approx \frac{1}{N}\sum_{i=1}^{N} e_{bi}, \tag{1}$$

where N is the number of boundary edges e_{bi} of the hole.

After that p_n is formed with a boundary edge e_b to create a new triangle T_{new}; and T_{new} also satisfies the Delaunay criterion as presented in [11]. The process is then repeated from T_{new} until the hole is full.

In order to preserve the local curvature and guarantee the smoothness of the hole boundary, we also compute the value of z coordinates for each time inserting this new point $p_n(x, y, z)$ into the hole. The z value is computed as

$$z(p_n) = \frac{1}{m}\sum_{j=1}^{m} z_j(p_{\text{nei}}), \tag{2}$$

where m is the number of neighboring points p_{nei} based on 4-connectivity of p_n.

Figure 5 is an illustration. In Figure 5(a), we insert three new points, the distance from p_{n1} to p_{n2} and to p_{n3} approximates the distance between e_b. The value of coordinates z of p_{n1} is computed based on the z values of p_{b1}, p_{b2}, p_{b8}. Similarly, the z value of p_{n2} is calculated based on the z values of p_{n1}, p_{b7}, p_{b3}, etc.

Table I
Comparison of the Processing Times Between the Methods. We Use the Same Input Data Points and Run on the Same Computer (Intel 2CoreDue, 2GB of Ram).

Input points	CTC Output faces/ Time: ms	VTS Output faces/ Time: ms	BP Output faces/ Time: ms
15626	29871/14631	29871/13360	29990/18984
32402	61073/30911	61073/28984	58030/67871
60511	115828/68797	115828/63223	110982/225037
68956	125463/112117	125463/93246	118273/268276
98231	181851/194682	181851/162451	174582/859341
148317	266187/246435	266187/218713	267842/2017632
886639	1618624/1473188	1618624/1307466	1601158/11951403

4 Implementation and Results

We test our method on various data sets that are noisy, irregular, and with some holes on the surface. Our algorithms are programmed and integrated in Meshlab [20] as a plug-in. Therefore, we can test the running time between our method and the existing method in the Meshlab. We compare the processing times between our methods: VTS, CTC [11] and "Ball Pivoting" (BP) (the method in Meshlab). We also compare the shape of the triangulated surfaces with the input ones. The processing time of our method is faster and as we will see, the initial shape of the surface is well preserved. After triangulating the surfaces and processing their boundaries with our methods, the total processing times are presented in Table I. In this table, we used the same input surfaces to test with the three methods. Both our neighboring searched methods (VTS and CTC) are integrated and tested (one by one) in the same algorithm and on the same computer. They are only different from their ways to find a neighboring point. The processing time is a bit different between them (see Table I), and the reason has also been detailed in [11]. The BP method generated a very good mesh, but the processing time was higher than our methods. If the number of input data points increases, the running time to triangulate the surface is far from our method (see Figure 6).

We have processed the triangles on the concave parts of the boundary by removing some outward triangular faces of surface S (see Figure 8). The exterior boundary of S and surface S has been processed in the previous

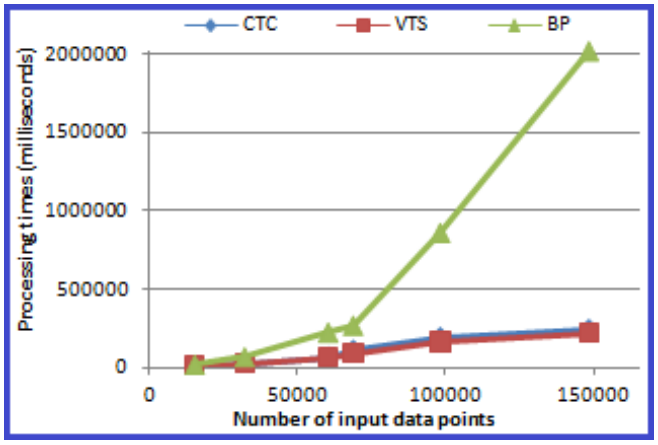

Figure 6. Comparison of the processing times between the methods: CTC, VTS and BP. On this graph, we do not plot the last example with 886639 points (in Table I) because it is too far from other examples and therefore spoils the graph.

works [9, 10]. Therefore, the number of boundary points and their density depend on the previous processing and the initial shape of the surface. For these reasons, the number of triangular faces (in Table I) is a bit different from the classical triangulation algorithms (in 2D Delaunay triangulation: the number of triangular faces is at most $2n - 2 - b$ triangles; where n is the number of vertices and b is the number of vertices on the convex hull). After processing the ourward triangles on the boundary, we fill some small holes of the triangulated surface. The result is shown in Figure 9 with the approximation error approaching the input surface.

5 Discussion and Evaluation

In this section, let us give some discussion and evaluation about our method. As presented in [11], the main novelty of our method for triangulating the surface of 3D point clouds is that the neighboring point p_3^i is searched in a rectangle supported by the edge e_i under consideration. So, the most important point of the proposed method is our particular search in order to speed up the Delaunay triangulation. In fact, the input points are in 3D. We first projected them onto a natural grid in the (x, y) plane for computing the Delaunay triangulation. The surface is then triangulated on 3D by restoring the z coordinates for each point. The way to search a neighboring point p_i for creating a triangle (p_1, p_i, p_2) in [15] (in 2D) is nearly the same way of computing the compactness triangle (CTC in [11]) (i.e., the largest angle at the found point). This method mostly needs to compute all neighboring points in one side of an edge p_1, p_2 to find the best one (see Figure 7(a)). Therefore, the complexity of this computation is always at most $N \times i$ (where N is the number of points of the surface) for iteration. In our method (VTS in [11]), we aim to find the point that is close to the position of the third point of an equilateral triangle created by e_i (this point is always located on the non-empty voxel crossed on the ray L, if it exists). If there

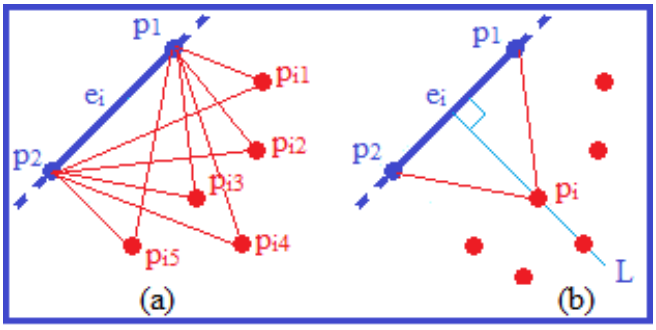

Figure 7. Searching a neighboring point: a) computing the compactness; b) voxel traversal search.

is no point on L, we dilate from L until finding an appropriate point (see Figure 7(b) and Figure 1). For this reason, we do not need to check all neighboring points as computed in the method [15]. Another method is known as triangulation of a planar in 2D [16] that the complexity is expensive for checking the Delaunay criterion of edge flipping. Therefore, comparing the existing methods in 2D [15, 16], the processing time of our method is faster (see Table I). The BP method [14] computes a triangle mesh interpolating a given point cloud entirely in 3D. In order to find a neighboring point, it has to test all points by pivoting around an edge until finding the best point; so, the processing time is higher. In order to evaluate the quality (smoothness or accuracy) of our triangulated surface (generated by our method) with respect to the initial shape of that surface, we use the Metro tool [21] to compute the distance between two sampled-points surfaces (S1: the input surface of 3D point clouds; and S2: the output surface after simplifying and triangulating S1, as described in [10]). The initial shape of the BP method is well preserved after triangulating while consuming the running time for the surface triangulation. Our method, VTS obtained the good results on both processing time and approximation errors. After triangulating the surfaces, we reconstruct these surfaces by removing the outward triangles on the boundary and filling the holes, the initial shape (e.g. curvature, bend, ridge, and valley) of these surfaces is well preserved (see Figures 9, 10).

6 Conclusion

In conclusion, we have proposed a complete method for reconstructing a surface of 3D point clouds based on three steps. The triangulating step is performed by using a fast algorithm, which speeds up the Delaunay triangulation. The obtained results have shown that the processing time is very fast while preserving the mesh quality [11]. We have processed the outward triangles on the boundary (taking the advantage of boundary processing in [9]) to avoid the triangulation of the concave parts of the surface. This step helps us to reconstruct the surface in order to obtain a surface that approaches its initiation [12]. In the last step, we proposed a method to fill the hole of a triangular mesh. We based on the neighborhood information on

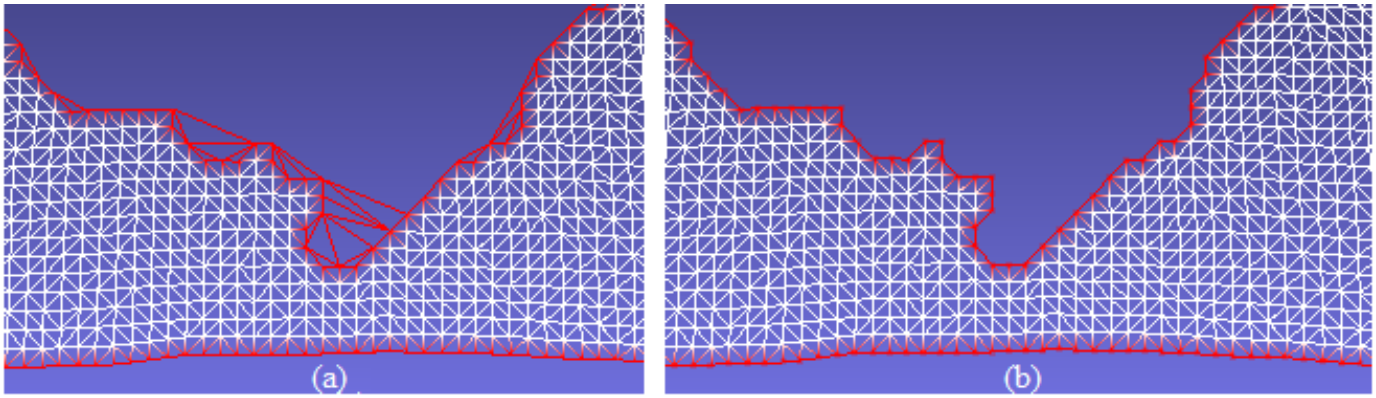

Figure 8. a) Before and (b) after removing the outward triangles on the boundary.

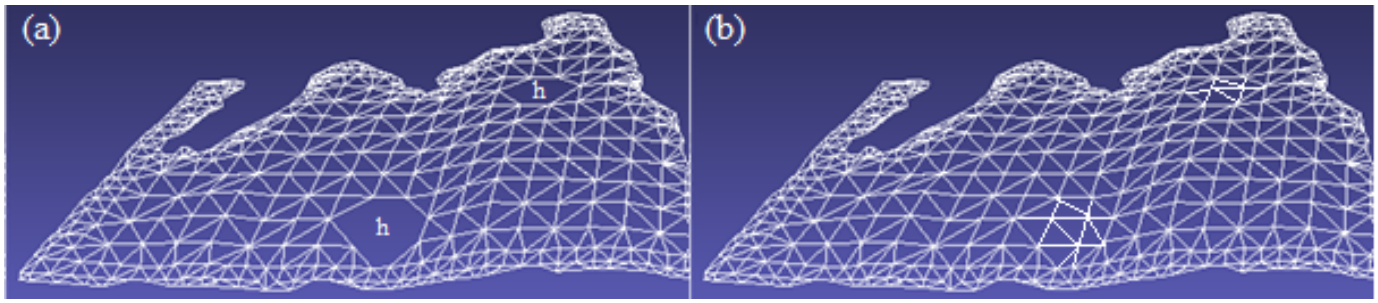

Figure 9. a) A geological surface of 3D point clouds (232 kb). b) After simplifying by using the elaborate method [10] (cell size = 8, $\partial \leq 0.12$), triangulating and filling the holes (h), the size of surface: 18 kb; the approximation error between (a) and (b) is Δ_{max}: 0.020; Δ_{avg}: 0.0006; the triangular faces vary in density from the boundary to the inside of the surface.

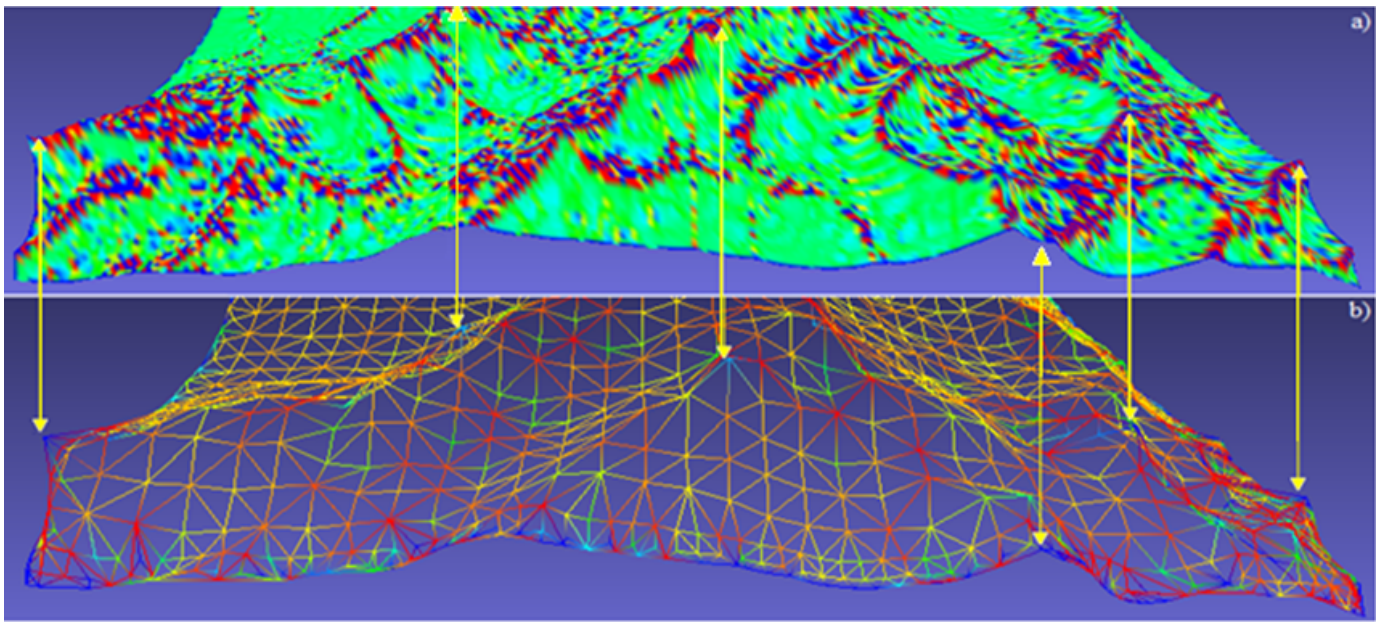

Figure 10. a) The input surface of 3D point clouds with 2629 kb. b) After simplifying by using the elaborate method [10] (cell size = 8, $\partial \leq 0.09$) and triangulating, the size of surface: 68 kb; the approximation error between (a) and (b) is Δ_{max}: 0.018; Δ_{avg}: 0.002; There are some color points which have shown that the characteristics of the surface are well preserved.

boundary points of the holes that structured in a 3D grid to fill and triangulate these holes. Our method does not compute intermediate implicit surfaces (like normal estimation of boundary points) and the quality of the triangular mesh is checked by using a tool Metro in [21]. Therefore, the processing time is very fast, the local curvature of the holes is maintained and the obtained results are "optimal" geological triangulated surfaces adapted to our goal. However, the fault on the surface is not handled because it has been processed before our work.

Acknowledgment

The work in this research is an extending part of a PhD thesis [7] that is funded by Project 322 of Vietnam Ministry of Education and Trainning (MOET). This research is also funded by International University, Vietnam National University, Ho Chi Minh city under grant number T2015-02-IT/HD-DHQT-QLKH. We would like to thank MOET and the International University for the funds, and the reviewers for their valuable comments.

References

[1] M. Pauly, M. Gross, and L. P. Kobbelt, "Efficient simplification of point-sampled surfaces," in *Proceedings of the conference on Visualization'02*. IEEE Computer Society, 2002, pp. 163–170.

[2] Y.-J. Zhang and L.-L. Ge, "A robust and efficient method for direct projection on point-sampled surfaces," *International journal of precision engineering and manufacturing*, vol. 11, no. 1, pp. 145–155, 2010.

[3] H. Hoppe, "Surface reconstruction from unorganized points," Ph.D. dissertation, University of Washington, 1994.

[4] J. Ma, "Surface reconstruction from unorganized point cloud data via progressive local mesh matching," Ph.D. dissertation, School of Graduate and Postdoctoral Studies, The Univerity of Western Ontario, 2011.

[5] X. J. Wu, M. Y. Wang, and B. Han, "An automatic hole-filling algorithm for polygon meshes," *Computer-Aided Design and Applications*, vol. 5, no. 6, pp. 889–899, 2008.

[6] Z. Li, D. S. Meek, and D. J. Walton, "Polynomial blending in a mesh hole-filling application," *Computer-Aided Design*, vol. 42, no. 4, pp. 340–349, 2010.

[7] V. S. Nguyen, "3d modeling of elevation surfaces from voxel structured point clouds extracted from seismic cubes," Ph.D. dissertation, Aix-Marseille University, 2013.

[8] P. Verney, "Interprétation géologique de données sismiques par une méthode supervisée basée sur la vision cognitive," Ph.D. dissertation, École Nationale Supérieure des Mines de Paris, 2009.

[9] V.-S. Nguyen, A. Bac, and M. Daniel, "Boundary extraction and simplification of a surface defined by a sparse 3d volume," in *Proceedings of the Third Symposium on Information and Communication Technology*. ACM, 2012, pp. 115–124.

[10] ——, "Simplification of 3D point clouds sampled from elevation surfaces," in *21st International Conference on Computer Graphics, Visualization and Computer Vision (WSCG)*, 2013, pp. 60–69.

[11] V. Sinh.N, Alexandra.B, and Marc.D, "Triangulation of an elevation surface structured by a sparse 3D grid," in *IEEE Fifth International Conference on Communications and Electronics (ICCE)*, 2014, pp. 464–469.

[12] V. S. Nguyen and M. H. Tran, "Reconstruction of an elevation triangular mesh from 3D point clouds," *Journal of Science and Technology, Vietnamese Academy of Science and Technology*, vol. 5, no. 4A, pp. 220–29, 2014.

[13] A.Bac, V. Nam, and M.Daniel, "A multistep approach to restoration of locally undersampled mesh," in *Proceedings of the 5th international conference on Advances in geometric modeling and processing (GMP'08)*, 2008, pp. 272–289.

[14] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, "The ball-pivoting algorithm for surface reconstruction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 4, pp. 349–359, 1999.

[15] T.-P. Fang, L. Piegl *et al.*, "Delaunay triangulation using a uniform grid," *Computer Graphics and Applications, IEEE*, vol. 13, no. 3, pp. 36–47, 1993.

[16] V. Domiter and B. Žalik, "Sweep-line algorithm for constrained Delaunay triangulation," *International Journal of Geographical Information Science*, vol. 22, no. 4, pp. 449–462, 2008.

[17] N.V.TRAN, "Traitement de surfaces triangulées pour la construction des modèles geologique structuraux," Ph.D. dissertation, Université de la Méditerranée, 2008.

[18] G. Casciola, D. Lazzaro, L. B. Montefusco, and S. Morigi, "Fast surface reconstruction and hole filling using positive definite radial basis functions," *Numerical Algorithms*, vol. 39, no. 1-3, pp. 289–305, 2005.

[19] S. Salamanca, P. Merchán, E. Pérez, A. Adan, and C. Cerrada, "Filling holes in 3D meshes using image restoration algorithms," in *International Symposium on 3D Data Processing, Visualization, and Transmission*, vol. 2, 2008.

[20] ISTI. Meshlab, 2013. http://meshlab.sourceforge.net/.

[21] R. P.Cignoni, C.Rocchini, "Metro: Measuring error on simplified surfaces," in *The Eurographics Association*, 1998.

Nguyen Van Sinh is a lecturer of the School of Computer Science and Engineering at International University, Vietnam National University, Ho Chi Minh city. He received the doctoral degree of computer science in 2013 from the École Doctorale en Mathématiques et Informatique (ED.184), Aix-Marseille University, France. He received his master degree of computer science in 2008 from Asian Institute of Technology, Thailand. His research interests include computer graphics, images processing, geometric modeling, 3D simulation, and source code security.

Tran Manh Ha is a lecturer of the School of Computer Science and Engineering at International University, Vietnam National University, Ho Chi Minh city. He received his master degree of computer science in 2004 from the University of Birmingham, United Kingdom and his doctoral degree of computer science in 2009 from Jacobs University Bremen, Germany. His research interests include distributed computing, big data analytics, information retrieval and network management.

Nhan Ba Cong is a master student at the School of Computer Science and Engineering at International University, Vietnam National University, Ho Chi Minh city. He received the bachelor degree of Computer Science in 2006 from University of Science, Vietnam. He currently pursues the master degree of Information Technology Management. His research interests include geometric modeling, big data analytics, mobile computing.