International Journal of Signal Processing, Image Processing and Pattern Recognition
Vol.7, No.3 (2014), pp.135-146
http://dx.doi.org/10.14257/ijsip.2014.7.3.12

A Robust Mesh Growing Surface Reconstruction Algorithm based on Octree

Liu Xumin, Yang Lixin and Li Cailing

(College of Information Engineering, Capital Normal University, Beijing 100048)

Abstract

In order to overcome the disadvantages of mesh-growing method for large-scaleor normal information needed or pre-processing needed data, a robust mesh-growing surface reconstruction algorithm based on Octree will be proposed. Starting from an octree for dividing points cloud into space. Then select the flattest point and its nearest two points to be the seed triangle, and put the three sides into the queue. And make principles to ensure the growing of points be close to the optimal. Finally, we judge the type of points to determine topology reconstruction. Compared with TightCocone algorithm, this algorithm can deal with point cloud data without the normal information and pre-progressing and the reconstruction results are good.

Key words: *surface reconstruction, mesh growing, octree, flat factor, point cloud search*

1. Introduction

Surface reconstruction from point cloud is the hot topic in Computer Graphics researches. Surface reconstruction can be widely used in Reverse Engineering, Virtual Reality, Mold reconstruction, Conservators, Medicine and other fields. Surface reconstruction algorithms are generally divided into two categories: implicit and explicit.

The basic idea of implicit reconstruction is to find an implicit function$f(p)$, and makes $f(p) = 0$, where $p$is either the whole point cloud or only a part of it. The most common implicit functions included: Oh take, *et al.*, in [1] introduced radial basis functions (RBF) algorithm that given a point cloud distributed over a surface, it sampled to construct a coarse-to-fine hierarchy of point sets and interpolated the sets of points. The speed of running time was fast, but it required the normal of points and complex computation. Kazhdan, *et al.*, in [2] converted surface reconstruction to a Poisson problem, the solution was anti-noise, but need the normal of points as input data. After that, a new approach called Polar Field based implicit surface reconstruction is presented by Lin Yuxu, *et al.*, in [3]. In this approach, a 3D surface was expressed as an equipotential surface of scalar polar field that can be solved with a greedy algorithm. This algorithm can maintain detail, better robustness, but it has a higher computational complexity and memory consumption.

Explicit methods can be classified into two main groups: Voronoi/Delaunay-based and mesh-growing approaches. The main idea of the Voronoi diagram is to create a zone that affected only by the core point.The Voronoi polygon problem is actually converted to a Delaunaytriangulationproblem for it is difficult to solve with computer. A Crust method based on Voronoi is proposed by Amenta, *et al.*, in [4], it was a good approximation of axis by the intensive sampling point clouds and the poles. The worst time complexity of the Crust algorithm is $O(n^2)$, where$n$is the sum of points and poles. To reduce the complexity, Amenta, *et al.*, in [5] then submit the Cocone algorithm which worst time complexity is $O(m^2)$, where $m$is the number of points. The final result was good but difficult to handle the cloud data with holes or non-uniform sampling rate. To overcome the above limitations, Dey, *et al*., in [6] proposed Tight Cocone algorithm which can handle points with holes and without extra input points. Other versions of Cocone were

ISSN: 2005-4254 IJSIP

presented after that, one suited for noisy data called Robust Cocone [7], and one suited for handling large amounts of data called Super Cocone [8]. A Cocone algorithm based on Octree proposed by Dey, *et al.*, in [9], can deal with large mount point cloud data.

Mesh-growing methods generate the tessellated surface from a seed triangle and grow the meshed area by using some criteria. The existed algorithms are: the Ball Pivoting algorithm introduced by Bernardini, *et al.*, in [10] that the advancing fronts rolled to growth etriangle mesh with radius r based on the BPA criteria, yet it was less robust in dealing with point cloud with noisy. Li, *et al.*, in [11] proposed an efficient priority driven algorithm from a different view. The key point of this method was to create a priority queue for the candidate points when grow the advancing front of the meshed area and then select the highest priority point. But it was appeared to be inadequate to deal with complex shape models. Angelo, *et al.*, in [12] then presents a new mesh-growing algorithm for fast surface reconstruction, a G2S criterion was introduced which was a good approximation of two-dimensional Delaunay. But the algorithm didn't do well in non-uniform point cloud data. In the implementation, the above mesh-growing algorithms use the hash-table to store the cloud point data and pre-process the data to get *K*-nearest neighbor points, the disadvantage is how to solve the hashing conflicts problem.

In order to reduce the time complexity, and process the data cloud without the normal of point cloud and pro-processing, a mesh growing surface reconstruction algorithm based on Octree is proposed. First an octree is built for dividing points cloud into space and then find the flattest point as initial point, its nearest two points $q$and r are chosen to compose the seed trianglepqr. Finally, we determine topology reconstruction by the type of the candidate points.

2. A Mesh Growing Surface Reconstruction Algorithm based on Octree

2.1. Build an Octree

Octree is a data structure of the three-dimensional space with eight child nodes. Each node of an Octree is regarded as the rectangular volume element. To build an octree, the whole Npoints should be loaded firstly, and then tag the maximum and minimum coordinates in XYZ axes and calculate the center point coordinates width, height and length of the bounding box. An octree can be recursive divided with end conditions of level limit or the limit number of points in one node. We choose the latter one for the uniform distribution of points, and recursive divided it with the limitation of the leaf nodes contain a maximum number of K points.In the experiment, we choose$K = 48$.

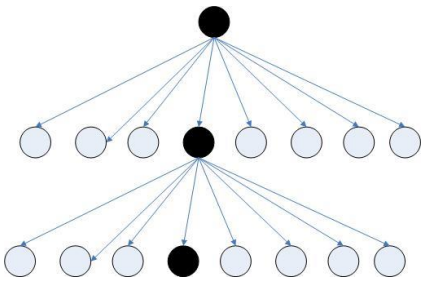


Figure 1. Structure of an Octree, each Node has Eight Child Nodes

Searching point's coordinates in 3D space may influence mesh-growing algorithm's speed, the common spatial search data structure includes: Hash-table, Quad-tree, R tree, KD tree, octree and so on. Hash-table is a linear storage method, and required pre-progressing of finding points' *K*-nearest neighbors. Hence, it is not easy to find a

solution to the hash conflicts for higher density point cloud and the synchronization of *K*-nearest neighbors during the topologicalre construction. Since octree can handle large-scale of point cloud data, as Figure 1 shows, the total number M of bounding box and the level $l$of an Octree should satisfy:$8^l = M$,when$l \geq 8, M \geq 16777216$, that M is enough large to handle large mount data. The algorithm use the coding search method presented by Zhang Yongyu, *et al.*, in [13] as Figure 2 shows, each cube is encoded with Morton encoding. The node marked by "00" continues to be divided into eight sub-nodes, the number 000,001,002,003,004,005,006,007 corresponds to the binary code 000,001,010,011,100,101,110,111.Using this encoding method, the worst time complexity of finding a point is $O(l + K)$, where K is the total number of points in one bounding box. The complexity is far smaller than the average $O(N/2)$which is the ordinary array storage time complexity. One octree can accommodate the number of $K * M$points.

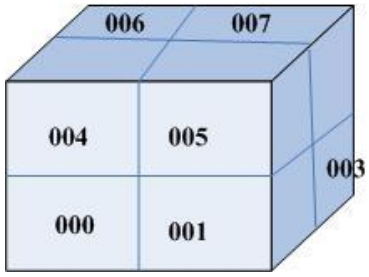


Figure 2. Schematic Diagram of Morton Encoding

2.2. Select the Seed Triangle

There are many ways to select the seed triangle. Lin, *et al.*, in [14] supposed to find the first point $p$of the seed triangle equals to the maximum in Z axes, select its nearest point $q$as the second point, then with pqaxis and both a cylinder diameter and height of l equals to the length of pq. Keep expanding l until all the points were surrounded in the cylinder, and find a point r which makes $|pr| + |pq| + |qr|$minimum, then pqr is the wanted seed triangle. The above algorithm satisfies the Delaunay triangulation principle, but the actual application is complex and $p$probably is a noise point. Chen Jinrui, *et al.*, in [15] supposed to choose a flattest point$p$ and its nearest two points $q$and r are chosen to be the seed trianglepqr. This algorithm needs *K*-nearest neighbors of every point so it costs more time. In order to reduce computation time, point cloud density is considered when finding the flattest area point. In searching flattest point, the density of candidate minimum bounding box should be greater or equal to a threshold value and then calculate its points' flatness. It can effectively reduce the computation time and filter the noise. Finally, calculate the flatness of candidate points according to *Eqs.* (1).

$$k(p_i) = \frac{1}{K}\sum \frac{fabs(dot((p_{ij}-p_i),v))}{\|p_{ij}-p_i\|} \quad (1)$$

Where $M(p_i)$is the covariance matrix of the point p_i, and v is the $M(p_i)$'s smallest eigen value's corresponding eigenvectors.$M(p_i)$ Is calculated by *Eqs.* (2)

$$M(p_i) = \sum_{j=1}^{K}(p_{ij} - p_{ic}) \otimes (p_{ij} - p_{ic}) \quad (2)$$

Where p_{ic} is the arithmetic mean of the nearest Kneighborhood of $p_i$by *Eqs.* (3)

$$p_{ic} = \frac{1}{K}\sum_{j=1}^{K} p_{ij} \quad (3)$$

If there are enough sample points, Ji Zhenfang in [16] presents that v can be estimated

as the normal of this sample point p_i. If the area is flatter,$dot((p_{ij} - p_i), v) \to 0$, and the flatness value is smaller, on the contrary, the flatness value is greater. In the experiment, we choose $K = 20$, the flattest point p_i should meet$\{p_i | k(p_i) = \min\left(k(p_j)\right), j \in S\}$. Where S is the total number of all the candidate points? After find the first point p, its nearest two points q and r are be selected to consist the seed triangle pqr.

2.3. Make the Criterion of Joining Point for the Advancing Front

In [11], PDA criterion is introduced, first remove the points which don't meet the basic requirement and calculate the weights of remaining points, then sort all the weights and choose the point which has the greatest weight. The disadvantage for PDA is different weight coefficients may correspond to different models. So it has a poor interaction and adaptability which guarantee the quality of the reconstruction result. In [12], a better method is proposed called G2S, it's an approximation of 2D Delaunay triangulation by guarantee that there is no other point exists in a triangular facets circumcircle. And it has a better reconstruction result.

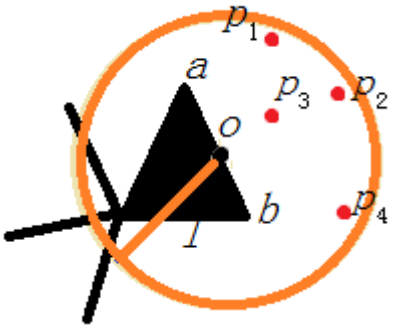


Figure 3. Schematics Join Point

The G2S criterion has more limitations in narrow triangle and growing direction. The algorithm make more restrictive to make 2D Delaunay more suitable for 3D space. The specific steps are as follows:

1) Determine the searching scope:
Shown in Figure 3, first finding the minimum bounding box of O which is the center of the advancing front ab. The searching scope's center point is $O$and itsradius$r = l * k$, l is the length of ab. $k$is a custom positive integer. The smaller k is, the result is meticulous. But if k is too small, it may cause bad reconstruction.

2) Filter the point of falling within the search scope:
The candidate point must be in the growing direction of the advancing front, or it will occur that one edge may be shared by many triangles. The angle between the new triangle's normal and the original triangle's normal should be in the scope of $[-\alpha, \alpha]$, in the experiment, α is in the range of $[45°, 90°]$,all the dissatisfied points should be removed. The new generated triangle also should avoid narrow triangle situation, or it will impede the next topological reconstruction. It is estimated by the angle β between the two new edges and the advancing front, and β should in$[25°, 135°]$.

3) Calculate the distance $d_{tmp}$of every remaining point to the center$O$, if d_{tmp} is shorter than the shortest distance d_{min}, update $d_{min} = d_{tmp}$ and update the corresponding p.

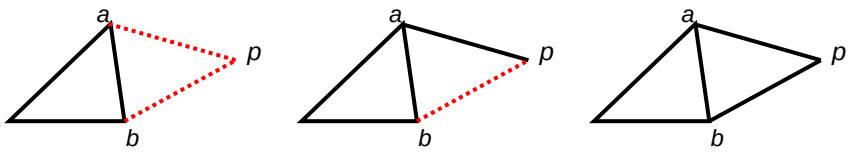
4) Determine whether to continue to search point:

The nearest point of center Omay be not in the minimum bounding box, so all the leaf nodes in the octree should be searched for the accuracy, yet the searching speed is slow. In [17], an algorithm for finding K-nearest neighbors of scattered points in the three-dimensions is introduced. It can be modified to use in searching nearest point. First calculate the shortest distance $dist$between the center points with six rectangular faces. If $dist < d_{min}$, jump the upper level of the current bounding box and continue Step 2). If $dist \geq d_{min}$,the searching is over, select d_{min}'s corresponding point p as the final added point.

5) If there is no suitable point, it means this advancing front is a boundary.

For Figure 3, $p_3$is chosen to be the added point.

2.4. Topological Reconstruction



(a)Join (b)Link (c)Fill

Figure 4. Mesh Topology

The algorithm use ***Join***、***Link***and ***Fill*** in [11] to combine, it can reduce computation and have a good reconstruction result.

To accelerate the running speed, all the useful edge pairs are put in the Map Queue, the searching time complexity is $O(1)$. During the mesh growing, first put the seed triangle's three edges into the advancing front queue. Then pop an advancing front ab. If ab is tagged as a deleted edge, put ab in the deleted edge Map Queue, and continue to pop the next advancing front, otherwise, according to the above criterion, find the added point p:

Execute the topological operation according to the type of point p:

As Figure 4(a) shows, if p is an unused point, add the point p and execute ***Join*** operation. Generate a new triangle apb and put it in the triangle index table, then put the new generated edge ap and pb in the advancing front queue.

If p is used, determine p's connection situation:

i. If p is connected with both a and b, as Figure 4(c) shows, execute ***Fill*** operation, generate the new triangle $apb$and put it in the triangle index table. Then put the ap and pb into the delete edge Map Queue.

ii. If p is connected with either$a$or b, as Figure 4(b) shows, execute ***Link*** operation, generate the new triangle $pba$and put it in the triangle index table. Then put the existed ap into the delete Map Queue, and put the new edge bp into the advancing front queue.

iii. If p is connected with neither $a$nor b, as Figure 4(a) shows, execute ***Join*** operation, generate new triangle $apb$and put it in the triangle index table. The new generate edges $ap$and $pb$are put in the advancing front queue.

Until all the advancing fronts are popped from the advancing front queue, the reconstruction is over.

3. Experimental Results

The algorithm was implemented by C++ with OpenGL, and ran in Microsoft Visual Studio 2010 Windows 7 environment. All tests were run on a PC with 2.10GHZ Pentium

Dual-Core processor and 4.00GB RAM (2.96 GB available).

The classic benchmark models are loop, egea, foot, rocker-arm, fnadisk_mc, face-YO, horse and standford_bunny. These models were downloaded from http://shapes.aimatshape.net/ and http://www.lodbook.com/models/. The experiment data and result are shown as Table 1, the reconstruction results are shown as Figure 5-8.

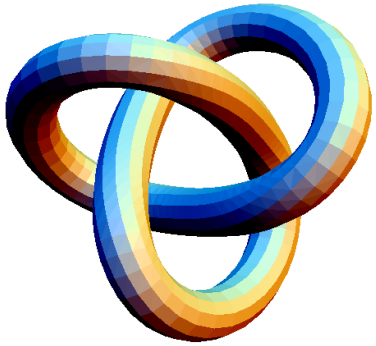
Table 1. Experiment Benchmarks Models and Reconstruction Result

Model name	Vertex num	Face num	Model name	Vertex num	Face num
loop	1440	3000	egea	8268	18486
foot	10010	22516	rocker-arm	10044	22068
fandisk_mc	11984	26518	face-YO	13746	28045
horse	19851	41911	standford_bunny	34834	72413

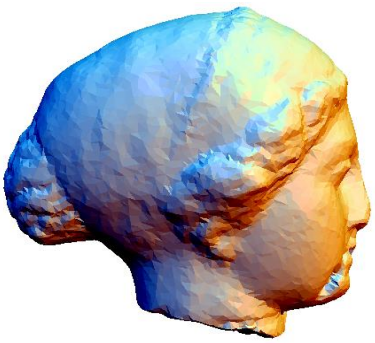
As Table 1 shows, the vertex number of models range from 1000 to 40000 included different levels of vertex number.

1) Testing Results of an Object with Different Geometries:

As Figure 5(a-h) demonstrate the robustness of the proposed algorithm in handling the various geometries. There is a little fragment after the surface reconstruction. The generated triangular mes his neat, well connected. For a non-uniform sampling rate area shown in Figure 6 and Figure 7, the generated triangular mesh is close and uniform.

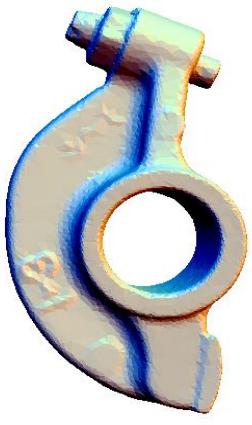


(a)loop

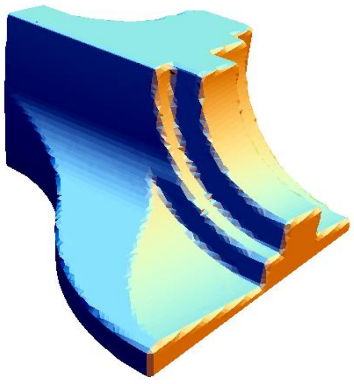


(b) egea

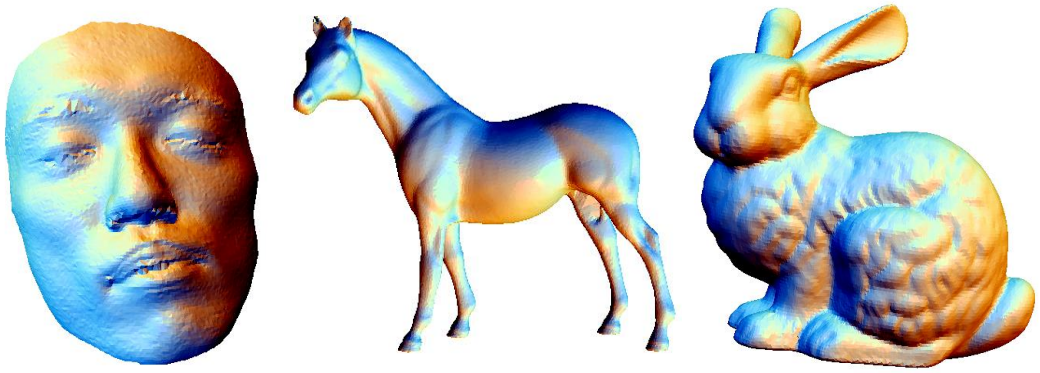
(c) foot



(d) rocker-arm

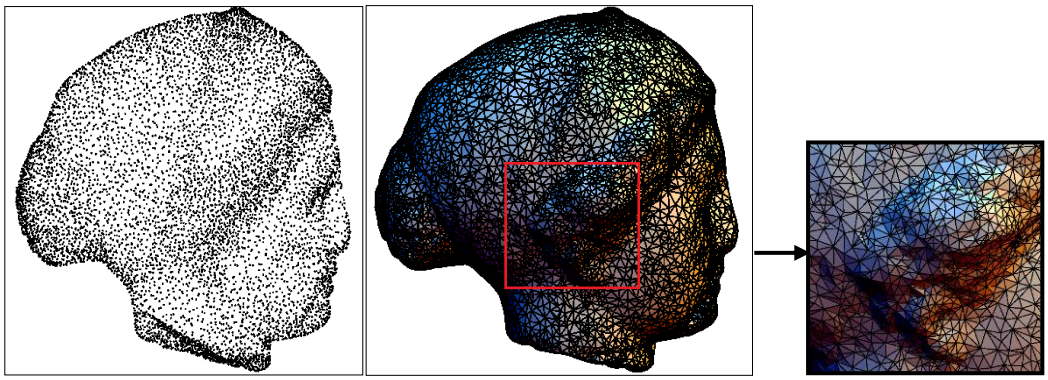


(e) fandisk_mc



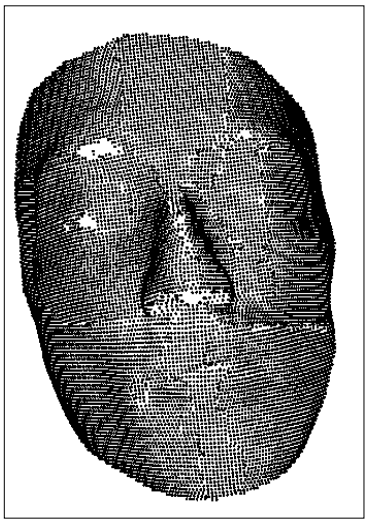
(f) face-YO (g) horse (h)standford-bunny

Figure 5. Surface Reconstruction Results of Different Geometries

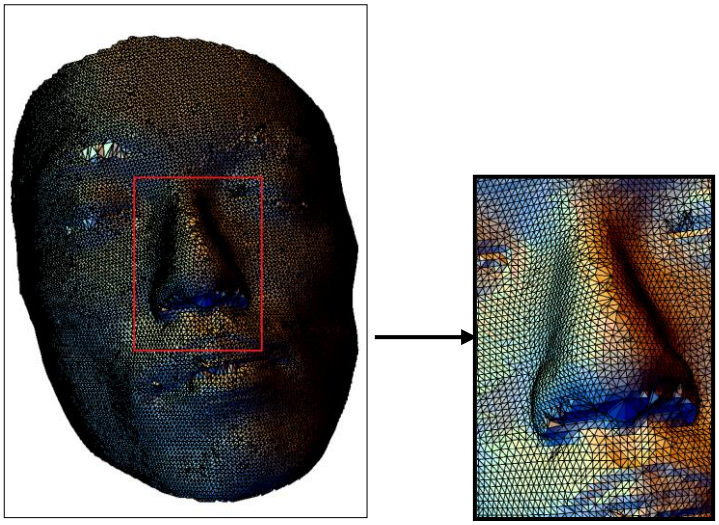


(a)egea point cloud (b)egea mesh (c)part of egea

Figure 6. Egea Reconstruction Triangular Mesh



(a) Face-YO Cloud Point

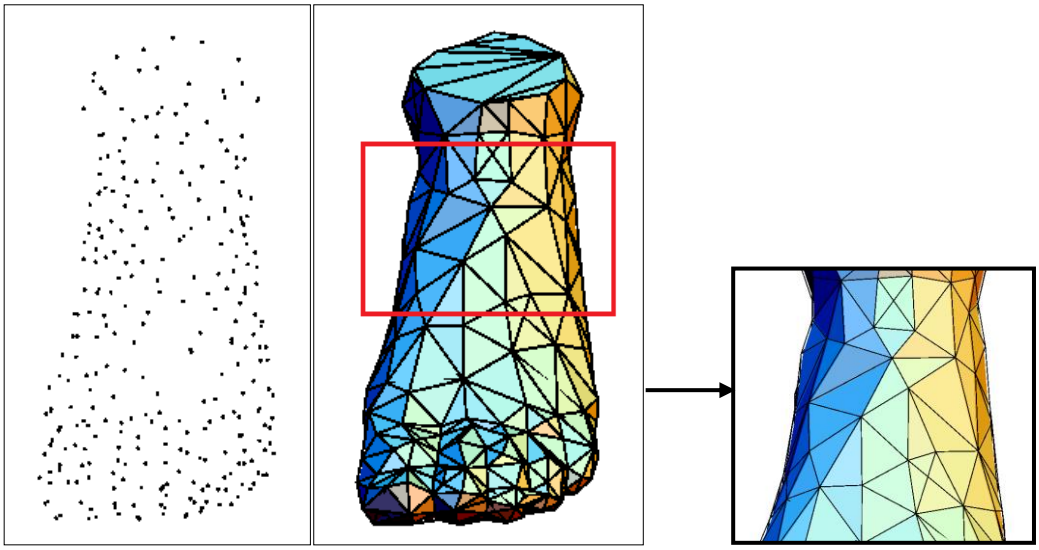


(b)Face-YO Mesh (c)Part of Face-YO

Figure 7. Face-YO Reconstruction Triangular Result

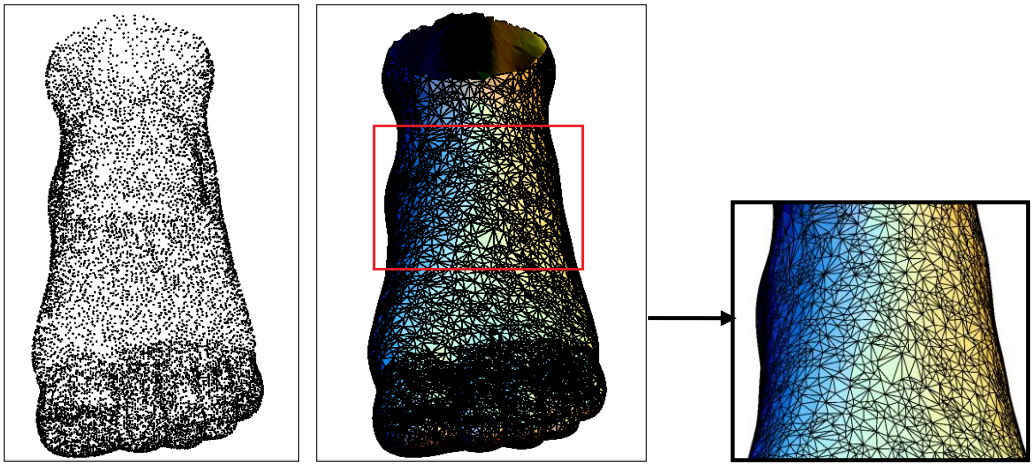
2) Testing Results of an Object with Different Sampling Density:

Figure 8(a) and Figure 9(a) show the different sampling density of foot models. Figure 8(b) and 9(b) show triangularmesh after reconstruction. Figure 8(c) and 9(c) are enlarged the red part it exhibits the right shape and characteristics of the model, and there is no incomplete or deformation during the reconstruction.



(a) Foot Cloud Point **(b) Foot Mesh** **(c) Part of Foot**

Figure 8. Foot_1 Reconstruction Result



(a) Foot Data Point (b)Foot Mesh (C)Part of Foot

Figure 9. Foot_2 Reconstruction Result

3) Compare with Tight Cocone Algorithm:

The algorithm is compared with Tight Cocone which has been kindly provided by Tamal Dey. In the experiment, a windows version is chosen. The option parameter is tcocone –r 1.0. Figure 10 shows the original experimental model, Figure 11 and Figure 10 show the compared parts which are marked by red rectangle. Figure 11(a), 11(c), 11(e) are processed by Tight Cocone, 11(b), 11(d), 11(f) are processed by the proposed algorithm.

It is obvious that the triangular mesh generated by the proposed algorithm is denser and more uniform than Tight Cocone's. The quality of reconstruction is better than Tight Cocone's. And the algorithm needs no the normal information and pre-processing at all.

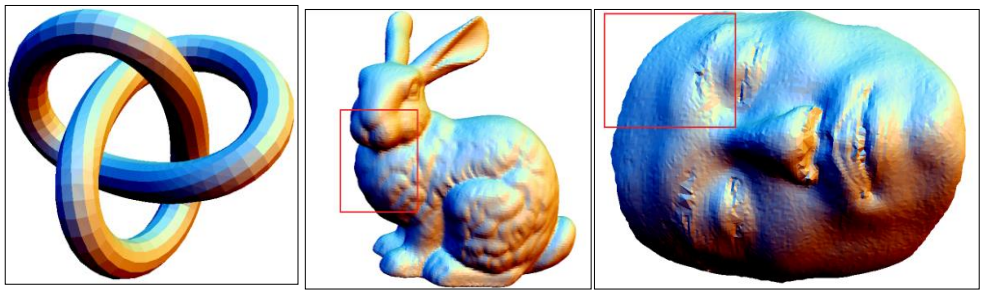
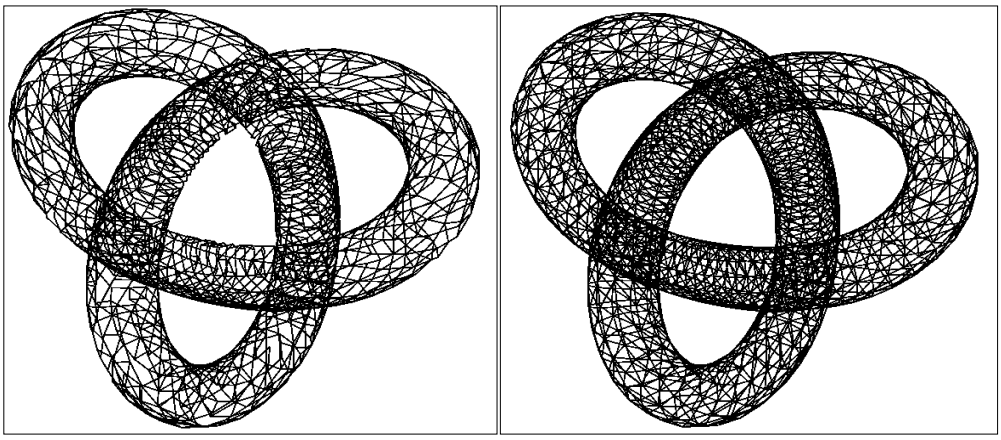
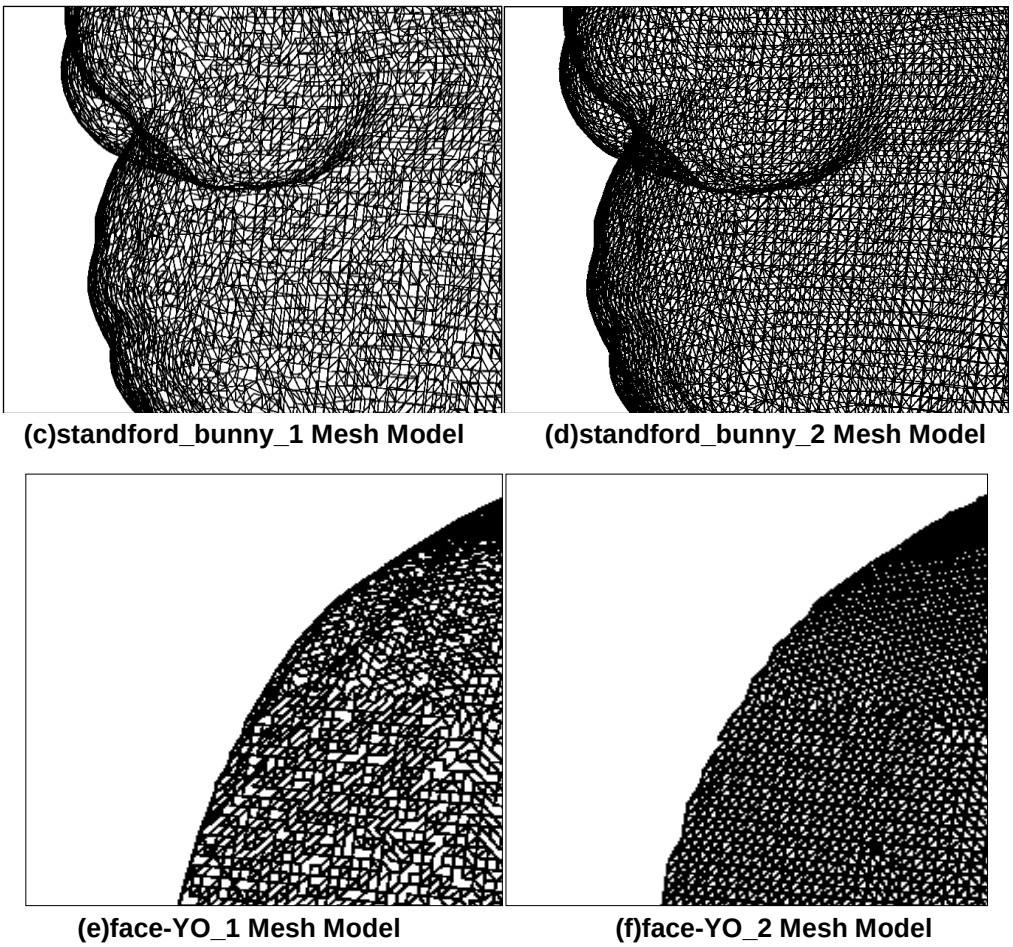


Figure 10. Loop, standford_bunny and Face-YO



(a)loop_1 Mesh Model (b)loop_2 Mesh Model



(c)standford_bunny_1 Mesh Model **(d)standford_bunny_2 Mesh Model**

(e)face-YO_1 Mesh Model **(f)face-YO_2 Mesh Model**

Figure 11. Partially Enlarge Mesh Model

4. Conclusion

In this paper, a mesh growing surface reconstruction algorithm based on Octree is presented. The algorithm doesn't need the normal of point cloud data, less computation, easy to implement and without pre-processing. Experimental result shows that the algorithm works well. But there may be little fragments in some un-uniform sampling models. Future work includes data parallel processing and improvement of the running speed.

Acknowledgments

This work was supported in part by NSFC under Grant Nos. 61272029.

References

[1] Y. Ohtake, A. Belyaev and H-P. Seidel, "3D scattered data interpolation and approximation with multilevel compactly supported RBFs", Graphical Models, vol. 67, no. 3 **(2005)**, pp. 150-165.
[2] M. Kazhdan, M. Bolitho and H. Hoppe, "Poisson surface reconstruction Proceedings of the fourth Eurographics symposium on Geometry processing", Euro graphics Association Press, Aire-la-Ville, **(2006)**, pp. 61-70.
[3] L. Yuxu, C. Chun, S. Mingli and B. Jiajun, "Polar Field Based Implicit Surface Reconstruction", Journal of Computer-Aided Design & Computer Graphics, vol. 21, no. 8, **(2009)**, pp. 1035-1041.
[4] N. Amenta, M. Bern and M. Kamvysselis, "A new Voronoi-based surface reconstruction algorithm Proceedings of the 25th annual conference on Computer graphics and interactive techniques", ACM Press, New York, **(1998)**, pp. 415-421.

[5] N. Amenta, S. Choi, T. K. Dey and N. Leekha, "A simple algorithm for home omorphic surface reconstruction", Internat. J. Comput. Geom. & Applications, vol. 12, **(2002)**, pp. 125–141.
[6] T.K. Dey and S. Goswami, "Tight Cocone: A Watertight Surface Reconstructor", Journal of Computing and Information Science in Engineering, vol. 3, no. 4, **(2003)**, pp. 302-307.
[7] T.K. Dey and S. Goswami, "Provable surface reconstruction from noisy samples", Computational Geometry, vol. 35, no. 1-2, **(2006)**, pp. 124-141.
[8] T.K. Dey, J. Giesen and J. Hudson, "Delaunay based shape reconstruction from large data Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics", Piscataway: IEEE Press, **(2001)**, pp. 19-27.
[9] T.K. Dey, R. Dyer and L. Wang, "Localized Cocone surface reconstruction", Computers & Graphics, vol. 35, no. 3, **(2011)**, pp. 483-491.
[10] 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, **(1999)**, pp. 349-359.
[11] X. K. Li, C. Y. Han and W. G. Wee, "On surface reconstruction: A priority driven approach", Computer-Aided Design, vol. 41, no. 9, **(2009)**, pp. 626-640.
[12] L. D. Angelo, P. D. Stefano and L. Giaccari, "A new mesh-growing algorithm for fast surface reconstruction", Computer-Aided Design, vol. 43, no. 6, **(2011)**, pp. 639-650.
[13] Z. Yongyu and M. Jinsong, "Research on the Establishment and the Query Algorithm of the Linear Octree Spatial Index for 3D GIS", Computer Engineering & Science, vol. 31, no. 2, **(2009)**, pp. 61-63.
[14] L. Hong-wei,T. Chiew-lan and W. Guo-jin, "A mesh reconstruction algorithm driven by an intrinsic property of point cloud", Computer-Aided Design, vol. 36, no. 1, **(2003)**, pp. 1-9.
[15] C. Jinrui, "Research on 3-D Reconstruction from Point Cloud", Wuhan: Wuhan University of Technology. Department of Computer Science & Technology, **(2011)**.
[16] J. Zhenfang, "A surface reconstruction of point clouds based on spacial status octree", Shandong: Shandong University. Digital Media Technology and the Department of Fine Arts, **(2011)**.
[17] X. Bangshu, H. Mingyi and Y. Huajing, "Algorithm for Finding k-Nearest Neighbors of Scattered Points in the Three Dimensions", Journal of Computer-Aided Design & Computer Graphics, vol. 16, no. 7, **(2004)**, pp. 909-912,917.

