Journal of Computers Vol. 31 No. 5, 2020, pp. 212-223
doi:10.3966/199115992020103105016

A Variable-Scale Refinement Triangulation Algorithm with Faults Data

Xiao-xia Luo, Zi-tong Wang*, Meng-fan Wang, Xiao-nan Shi

College of Computer Science and Technology, Xi'an University of Science and Technology Xi'an, China
{1536531696, 514861529, 3502337847, 12606137}@qq.com

Received 11 May 2019; Revised 20 September 2019; Accepted 21 November 2019

Abstract. The Delaunay triangulation in an unconstrained region has great research value. However, traditional unconstrained triangulation cannot deal with faults data. This paper summarizes the advantages and disadvantages of the traditional algorithm and proposes a variable-scale refinement triangulation algorithm with faults data. The triangulation algorithm is used to correctly embed faults line segments in an efficient manner. Firstly, based on the traditional incremental insertion method, a boundary contraction algorithm is proposed. New coordinate values are obtained using the magnification factor to construct an initial convex packet of the Delaunay triangulation. Secondly, the refinement of the overall mesh using a variable metric approach sets the scale and gradually refines the faults data. Finally, the refined faults data are embedded in the original triangulation information and a new constrained Delaunay triangulation is generated by the growth method. The triangulation generated by variable-scale refinement is uniform and easier to embed in the faults line segment, which improves the efficiency of network construction.

Keywords: boundary contraction, segment datasets, triangulation, variable-scale refinement

1 Introduction

In recent years, triangulation methods have been widely used in finite element analysis, computer imaging, coal exploration and geological information modeling [1-2]. The triangulation of the original graphics to obtain high-precision graphics processing requirements is increasing. However, the vertices of the initial triangle belong to the original data points or contour feature points. The distribution of data points is different for different application requirements and the spatial distribution of the triangles is uneven. Hence, refining triangulation is an important and challenging task.

Only a few researches were carried out to refine triangulation. Sang et al. [3] investigated the adaptive h-refinement algorithm for electric field integral equation applied to a conducting missile target. The process incorporated advancing front Delaunay algorithm to refine the triangular-cell surface mesh in conjunction with Laplacian smoothing to maintain the mesh quality. Tan et al. [4] proposed an algorithm by combining the vertex snapping method and element refinement method in mesh cutting to calculate barycentric coordinates of each element in cutting area and rendered the cutting trajectory by using duplication vertices. Long [5] combined the contour model and constrained Delaunay criterion to propose the refined CTIN (Constraint triangulated irregular network), which reduced the computational complexity. Bedregal et al. [6] improved the geometric results of edge refinement algorithm and provided precise bounds on the refinement propagation. Therefore, proving that the iterative application of the algorithm gradually reduced the average range of propagation of each target triangle. Gewali et al. [7] introduce the concept of node stability for refining the Delaunay triangulation. They presented an algorithm based on the location of center of gravity of two-dimensional shapes and placed a candidate refinement node to increase its stability.

* Corresponding Author

Although these methods refine triangulation, faults the traditional segmentation algorithm cannot deal with geological structures such as faults and folds faults. The point-by-point interpolation method and the variable-scale refinement algorithm are used to generate an unconstrained DT (Delaunay Triangulation) mesh [8]. The variable-scale subdivision algorithm [9] is used to process the faults edge, avoiding narrow and long appearance of triangles to finally combine with the growth method, and divide and conquer method to construct a CDT (Constrained Delaunay Triangulation) grid with faults constraints. The results proved that the refined triangulation was easier to deal with faults constraints and effectively improved the quality and efficiency of the network when compared with other common variable-scale refinement triangulation methods.

2 A Variable-Scale Refinement Triangulation Algorithm Framework with Faults Data

The overall framework of the algorithm is shown in Fig. 1. The algorithm fuses all the data (including faults constraint data) at first. Based on mesh index, the data are read by the recursive algorithm to form a convex hull. The constraint value of the edge length of the boundary polygon is tested to construct a high precision convex hull [10] and a point position is established using the area method. The optimal refinement scale is determined recursively followed by the complete refinement of the mesh. The refined faults data are embedded in the mesh by using the growth method. The location algorithm uses a high-efficiency cosine algorithm to achieve triangulation with a reasonable embedding of faults segments.

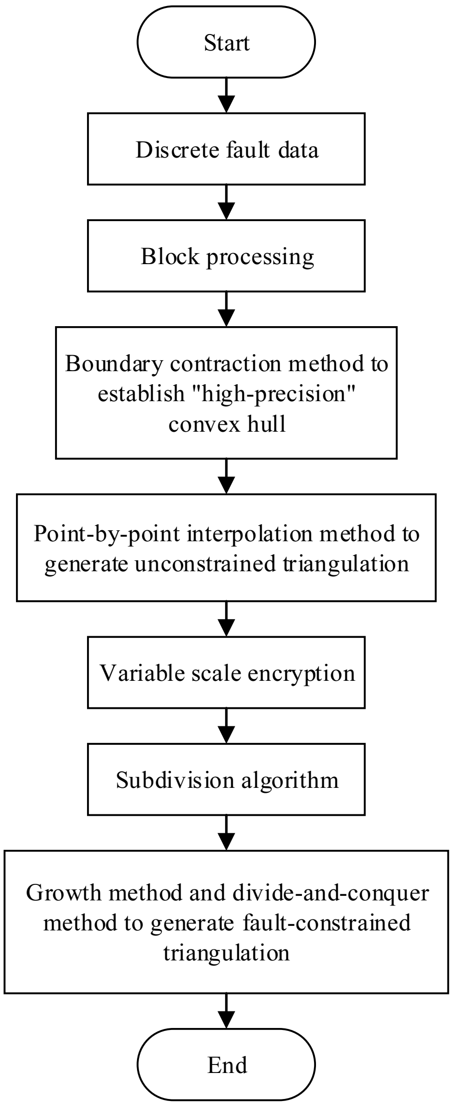

Start
Discrete fault data
Block processing
Boundary contraction method to establish "high-precision" convex hull
Point-by-point interpolation method to generate unconstrained triangulation
Variable scale encryption
Subdivision algorithm
Growth method and divide-and-conquer method to generate fault-constrained triangulation
End

Fig. 1. Algorithm framework

3 Formation of Unconstrained Triangulation

3.1 Building a Mesh Index

Assume that the number of discrete data points is *N*, and all of them are considered in the mesh. Because each mesh is too large or too small, the query efficiency is affected. Given the threshold m, the discrete points *v*, are segmented, $v=\left\{v_1, v_2 \cdots v_{\frac{N}{m}+1}\right\}$. To ensure the number of discrete points in each block is controlled within m, x and y are sorted in ascending order; if y is the same, they are sorted by the size of x size [11] ($\mathrm{m}=\log n$). Consider the four highest value points in the discrete area, i.e., $X_{\max}$, $X_{\min}$, $Y_{\max}$, and $Y_{\min}$.

$$\mathrm{a}_{row}=\frac{(\mathrm{int})(X_{\max}-X_{\min})}{size}+1 \,. \tag{1}$$

$$\mathrm{b}_{\mathrm{col}}=\frac{(\mathrm{int})(Y_{\max}-Y_{\min})}{size}+1 \,. \tag{2}$$

$$\mathrm{size}=\sqrt{\frac{s}{N / m}} \,. \tag{3}$$

$$S=(X_{\max}-X_{\min})*(Y_{\max}-Y_{\min}) \,. \tag{4}$$

where a_{row} and b_{col} represent the number of rows and columns that make up the grid; *N* is the number of discrete points; *size* is the length of each grid; and *S* is the area of the grid.

Project all discrete points on the mesh according to the rules, as shown in Fig. 2.

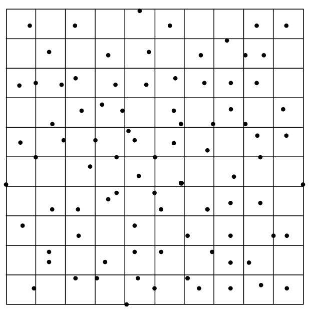


Fig. 2. Virtual grid

3.2 Establish a Regional Convex Hull

The method proposed by Zhang Qunhui et al. [12] is used to generate high finesse convex hulls based on boundary contraction as it is more suitable for faults data. A linked list is created containing the generated convex hull points stored in anticlockwise order. The specific data structure is the coordinates of the points, *x* and *y*. The steps are as follows:

Step 1: Create an initial convex hull. Find the x, y coordinates for this point set as the four extreme coordinate i.e., $X_{\max}$, $X_{\min}$, $Y_{\max}$, and $Y_{\min}$. Set an enlargement $C(C>1)$ to build the four extreme values:

$$\mathrm{x}_{super\_\max}=\left(x_{\max}+\left(x_{\max}-x_{\min}\right)\right)\times C \,. \tag{5}$$

$$\mathrm{y}_{super\_\max}=\left(y_{\max}+\left(y_{\max}-y_{\min}\right)\right)\times C \,. \tag{6}$$

$$\mathrm{x}_{super\_min}=\left(x_{\min}-\left(x_{\max}-x_{\min}\right)\right)\times C \,. \tag{7}$$

$$y_{super\_min} = \left(y_{\min} - \left(y_{\max} - y_{\min}\right)\right) \times C. \quad \textbf{(8)}$$

The maximum and minimum points of x, y, $x+y$, and $x-y$ are connected to form an initial convex hull, as shown in Fig. 3.

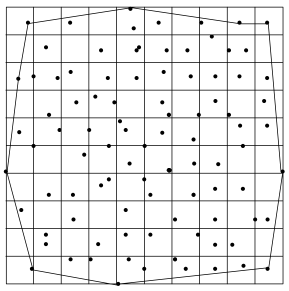


Fig. 3. Initial establishment of convex hull

Step 2: Select two points beside p_i and p_{i+1}, then view the point with the largest distance on the right side of p_i, p_{i+1}. If the point is between p_i, p_{i+1}, insert and repeat the process until the line formed by any two points in the linked list are calculated only once.

Query the point with the largest distance to the right of the line part. This can be judged by combining the mesh and the area of the triangle. When the three-point coordinates are known, the point (x_3, y_3) corresponding to the largest area is given by (9).

$$S = \frac{1}{2} * \left(x_1 y_2 + x_2 y_3 + x_3 y_1 - x_1 y_3 - x_2 y_1 - x_3 y_2\right). \quad \textbf{(9)}$$

Step 3: When the number of vertices in the linked list is equal to the number of times there is no insertion point, the operation is stopped, and the initial convex hull is successfully established.

3.3 Generation of High Boundary Precision Convex Hull

Traditionally, the convex hull has some limitations. Based on the traditional convex hull, this paper sets the maximum boundary value and searches recursively to (1) shrink the boundary of the original convex hull, (2) improve the precision of the convex hull, and (3) to accord with the practical data. The steps are as follows:

Step 1: Set the maximum convex shell boundary constraint value.

Step 2: A line segment that is longer than the largest side is recursively searched in a known convex hull. If it exists, its set *M* is obtained.

Step 3: The line part *AB* in the set *M* is removed sequentially and the closest point *J* near the line segment is searched.

Step 4: If point *J* exists, and if *AJ* and *BJ* meet the boundary constraint value, continue removing the next line segment. Otherwise, repeat step 3. The result with improved precision is shown in Fig. 4.

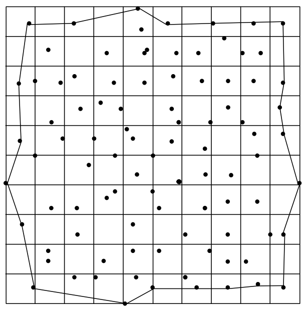


Fig. 4. High-precision convex hull

3.4 Initial Triangulation

Any point inside the convex hull connecting with the convex hull point, forms a base of the triangular mesh. Randomly select any point in the convex hull and connect all the points in the figure to perform the initial triangulation shown in Fig. 5.

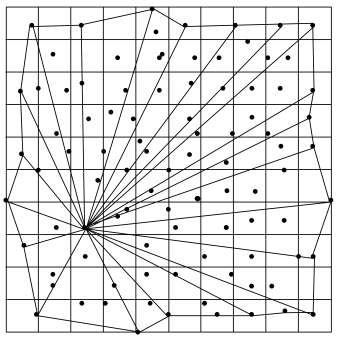


Fig. 5. Initial triangulation

3.5 Incremental Insertion for Complete Delaunay Triangulation

The remaining points are used for incremental insertion based on the initial triangulation. The LOP (Local Optimization Procedure) algorithm is used to determine these points to form the Delaunay triangulation.

Step 1: Randomly select the initial triangle and start searching.

Step 2: Insert the remaining points one by one. Determine the position of the point during the insertion process. Find the triangle *T* containing the point and delete the common side *T*.

The given point $p(x_0, y_0)$, is calculated to determine whether the point *p* is included in the triangle *T*. According to the nature of a triangle, there is a point inside the triangle connecting the three vertices of the point to the triangle. The sum of the three new angles formed by the point as the apex is 360°. There are no angles greater than 180° in the three new corners, so the sum of the two smaller angles must be greater than 180°.

Similarly, if the point is outside the triangle, the sum of the two smaller new angles formed should be less than 180°.

As shown in Fig. 6(a), the point p is outside the triangle *ABC*. The two smaller angles formed are $\angle APC$ and $\angle BPC$. $\angle APB + \angle BPC < 180^{\circ}$, therefore, the point *p* is outside the triangle *ABC*.

Similarly, as shown in Fig. 6(b), the two new angles formed by the point p and the triangle DEF are $\angle DPE$ and $\angle EPF$. $\angle DPE + \angle EPF > 180^{\circ}$, therefore, the point p is inside the triangle DEF. In summary, the position of the point p is obtained.

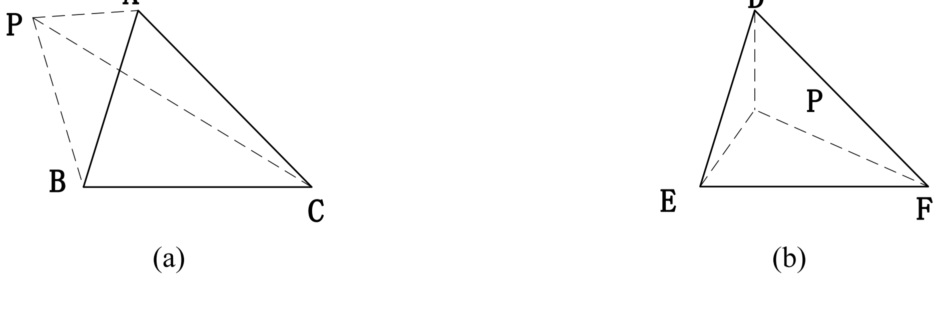


Fig. 6. Judge the position of the point

Step 3: After finding *T*, form a new triangle T_1, T_2, T_3 As shown in Fig. 7.

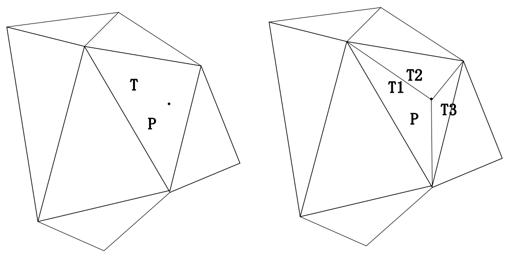


Fig. 7. Formation of three new triangles

Step 4: Use LOP for optimization [13].
Step 5: Repeat step 1. When all the points complete insertion, the triangulation is completed.
The process framework for incremental insertion is shown in Fig. 8.

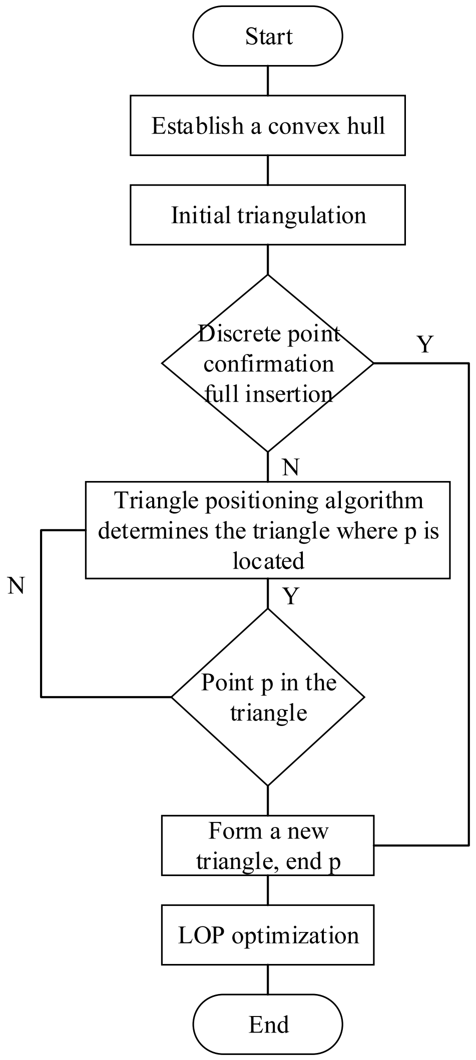

Start
Establish a convex hull
Initial triangulation
Discrete point confirmation full insertion
Y
N
Triangle positioning algorithm determines the triangle where p is located
N
Y
Point p in the triangle
Form a new triangle, end p
LOP optimization
End

Fig. 8. Incremental insertion frame

As the actual data is unorganized, the triangulation during this time is not structured. The generated triangulation needs to be processed, i.e., variable-scale refinement.

4 Variable-Scale Refinement Algorithm

The faults data is unorganized and densely distributed. To make the triangulation obtained by the triangulation uniform, convenient, and easy to adopt to the actual terrain, it is necessary to further refine the entire mesh. This paper proposes a variable-scale refinement scheme by setting a certain scale and reducing it gradually to normalize the mesh.

This is achieved by searching the formed triangle mesh, until the height of the triangle exceeds the division *H*. Consider a midpoint p_i off the edge and join them to form two new triangles [7]. The steps are as follows:

Step 1: Set a subdivision size *H*.

Step 2: Use $H * N$ (*N* is a natural number) for global refinement.

Step 3: The final subdivision result is produced when $H * (N-1)$ to *N* is 1, and the triangular mesh formed has a good scale [14] as shown in formula (10).

$$H_{\mathrm{i}} = \int_{N * h \ldots\ldots\ldots\ldots \mathrm{i}=1}^{(N+1-i)^* H_i \quad \mathrm{i} \le N} . \tag{10}$$

Step 4: Stop the subdivision and reiterate steps 1-3 until all triangles coincide at *H*.

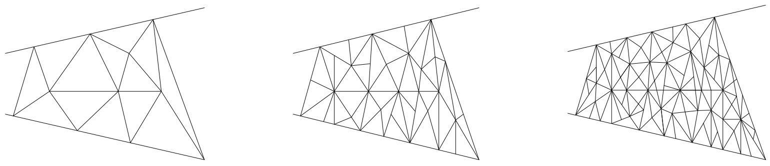


Fig. 9. Variable-scale refinement before and after triangulation

5 Embedded Faults Data

5.1 Faults Data Subdivision Algorithm

Connect all the faults data to form the faults edge. Subdivide all the faults edges, and store the formed points in the array *K* [15] using the following steps:

Step 1: Consider two points beside V_i and V_{i+1} from the discrete point set *V*, where, $i = (1, 2 \cdots n-1)$. Decide whether the line part V_iV_{i+1} is an edge of the triangle in the built CDT, if true, then proceed to the next edge. Conversely, continue to the next step.

Step 2: Calculate the distance *L* between V_i and V_{i+1}, and set a certain step *d* initially. If $d < L$, proceed to step 3, otherwise continue with step 1.

Step 3: Divide $D = \frac{L}{d}$ and V_iV_{i+1} into $D+1$ segments and store the points in the array *K*.

5.2 Insert the Subdivided Faults Data Points

After subdividing, insert the points into the grid using the following steps:

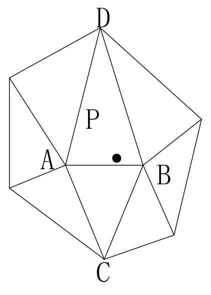
Step 1: Set the search radius *r* according to the maximum height *H* in the variable metric algorithm.

Step 2: Consider a random point k_i from *K*. The shortest distance from k_i to the three vertices of triangle t_i is assigned to d_i. If $d_i < r$, the triangle t_i is stored in the temporary array *E*.

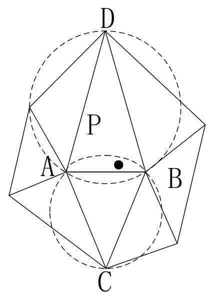
Step 3: Calculate the circumcircle of k_i and *E*. Mark the triangle if there is inconformity.

Step 4: Delete the adjacent edges of the triangle marked in step 3. Connect k_i to each point and store the resulting triangle in *T*. Empty *E* and reiterate step 2-4 until all the faults data is inserted into the grid.

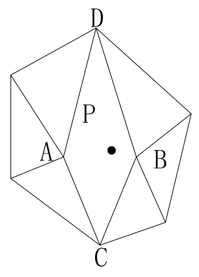
As shown in Fig. 10, the new node *p* is placed first. The relative position of the point *p* is determined by connecting the surrounding points. The common side *AB* is deleted and connected to form a new triangle as per Delaunay triangulation.



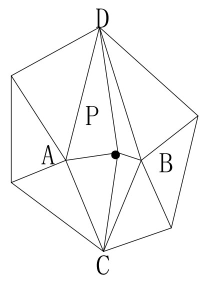
(a) Insert a new node P



(b) Decide how point P is connected to other nodes



(c) Delete the AB side



(d) Form a new triangle

Fig. 10. Inserting the faults data point

In Step 3, the specific steps of the point location algorithm are as follows:

Assuming that the point to be inserted is *p* and the target triangle is triangle *ABC*, and the point is in a straight line, as shown in the formula (11).

$$d = (A_X - B_X)P_Y - (A_Y - B_Y)P_X - B_Y A_X + B_X A_Y \,. \qquad \textbf{(11)}$$

If $d > 0$, the point *p* is on the right side of the line AB. If $d < 0$, the point *p* is to the left of the line *AB*, whereas if $d = 0$, the point p is on the line *AB*. Based on this idea, calculate *d* for the three sides of the triangle *ABC*, assuming that the relationship between *p* and the three sides of the triangle *ABC* is represented by d_1, d_2, d_3. If all three are greater than 0, then the point is inside the triangle. If two of them are greater than 0 and the third is equal to 0, then *p* is above one of the sides of the triangle *ABC*. If only one of them is greater than 0, and all others are 0, then the point p coincides with a certain vertex of the triangle. The rest of the situation is to indicate that point *p* is outside the triangle.

5.3 Faults Segment Embedded in the Mesh

Assign k_i to k_{i+1} and store it in array *K* to decide whether the line part in *K* is a line part in the grid. If not, use the growth method to embed the constraint [12].

Step 1: Consider k_i, k_{i+1}, and $i \le n-1$ from the array *K* in order, and judge whether $k_i k_{i+1}$ is in the mesh. If it exists, continue step 1. Otherwise, proceed to step 2.

Step 2: Consider the midpoint of $k_i k_{i+1}$ as the center of the circle. Use the length of $\frac{k_i k_{i+1}}{2}$ as the radius. Add the triangle with the intersection of the circle to the triangle set $NT = \{T_1, T_2 \cdots T_n\}$. The set *NT* is the influence domain of the constraint line part.

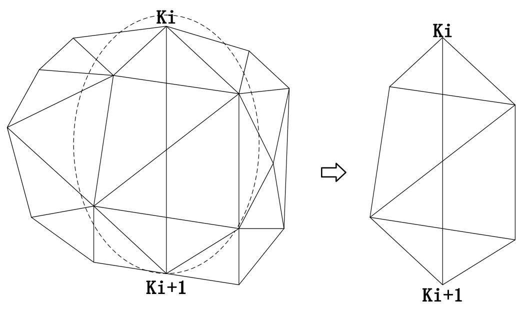
Step 3: Use a queue *Q* to store the points in the array *K*. In the influence domain, use $k_i k_{i+1}$ as the base (for the right example). Expand to the right and search for *p*, so that angle $pk_i k_{i+1}$ is the maximum, which forms $pk_i k_{i+1}$.

Step 4: Repeat step 3 on pk_i and pk_{i+1} to continue producing extensions. If the extended boundary (such as $k_i p$ or $k_{i+1} p$) is the edge of the outer boundary or has been used twice, then stop expanding outward and incorporate the triangle into *T*.

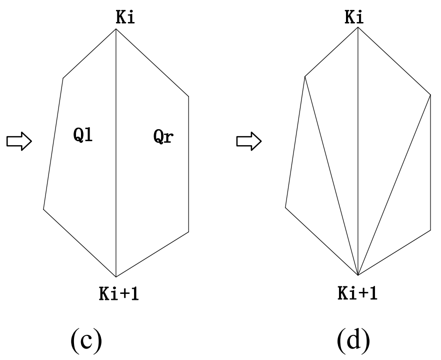
Step 5: Reiterate these steps until the queue *Q* is empty. Continue rebuilding the triangulation of the affected line part.

Step 6: Search the reconstructed triangles for LOP algorithm optimization. If there is a triangle concyclic, swap the diagonal.

As shown in Fig. 11(a), the $k_i k_{i+1}$ line part is in a triangular network and circled. As shown in Fig. 11(b), the line part has an influence domain. Consider the midpoint of $k_i k_{i+1}$ as the center of the circle with the length of $\frac{k_i k_{i+1}}{2}\infty$ as radius. Select the influence domain of the constraint line part and delete the line parts that intersect the constraint line part in the influence domain as shown in Fig. 11(c). Divide it into two parts and follow the steps until the affected line part is completed. Reconstruct the triangulation as shown in Fig. 11(d) to form a triangular mesh.



(a) (b)



(c) (d)

Fig. 11. Constrained triangulation of the affected area

6 Algorithm Analysis

Table 1 presents a comparison of the time complexity for common triangulation algorithms [16].

Table 1. Complexity of each algorithm in producing triangulation

Algorithm	General	Worst case
Growth method	$O(n^{3/2})$	$O(n^2)$
Traditional incremental insertion	$O(n^{4/3})$	$O(n^2)$
Divide and conquer algorithm	$O(n\log n)$	$O(n^2)$
Insertion method based on optimal convex hull	$O(n\log n)$	$O(n^2)$

When producing variable-scale refinement triangulation with faults data, three activities consume a lot of time: incremental insertion, point location algorithm, and inserting constrained boundary. This paper is based on the traditional incremental insertion method that adds mesh index and convex shell generation. The algorithm complexity of incremental insertion method is $T_1(\mathrm{n})(n\log n)$. During convex hull generation, time complexity is $T_2(\mathrm{n}) = O(n\log n)$. The area method is used in the triangle location algorithm, whose time complexity is $T_3(\mathrm{n}) = O(n)$. The time complexity for the variable-scale refinement process is $T_3(\mathrm{n}) = O(n)$. Time complexity for the subdivision algorithm in the constraint line segment embedding process is $T_4(\mathrm{n}) = O(1)$. The time complexity for embedding faults segments into the mesh using the growth method is $T_5(\mathrm{n}) = O(n^{3/2})$. Therefore, the time complexity for the triangulation of the entire constrained domain is $T_1(\mathrm{n}) + T_2(\mathrm{n}) + T_3(\mathrm{n}) + T_4(\mathrm{n}) + T_5(\mathrm{n}) = O(n^{3/2})$.

In summary, the time complexity of the algorithm is low, whether it is in the incremental insertion method, point location algorithm or constraint domain embedding.

7 Experiment

The algorithm was written in the Microsoft Visual Studio 2015 platform to carry out faults triangulation construction of variable-scale refinement. The time performance test was conducted on the debugged version. The test environment was a Windows 10 64-bit operating system running on a 3.89 GHz Intel i5-8300H CPU with 16 GB memory and a 6 GB GTX1060 graphics card. The execution efficiency of the traditional growth method [17], incremental insertion method [13], and variable-scale refinement triangulation algorithm is shown in Table 2.

Table 2. Algorithm execution efficiency test results

Algorithm	Number of triangles after splitting	Number of triangles after splitting	Whether with a faults
Growth method	625	790	No
Traditional incremental insertion	802	2328	Yes
Variable-scale refinement triangulation with faults data	830	1510	Yes

As shown in Fig. 12(a) represents triangulation without variable-scale refinement and faults data; (b) represents a triangulation result after variable-scale refinement and simple faults details; (c) represents the triangulation of variable-scale refinement based on Fig. 12(b) where the faults is more pronounced. The algorithm used in this paper is superior to the traditional algorithm considering the time efficiency of triangulation as shown in Table 2.

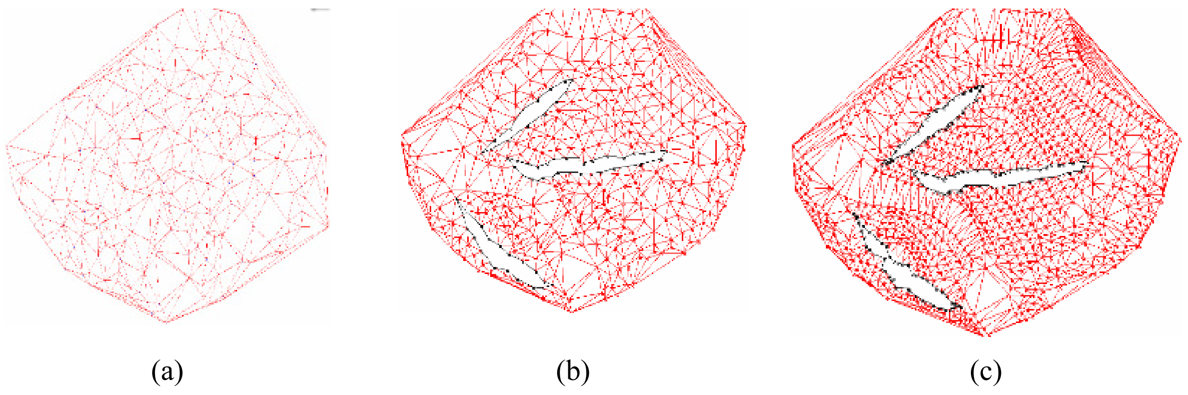


Fig. 12. Variable-scale refinement triangulation splitting effect diagram

8 Conclusion

The proposed variable-scale refinement triangulation algorithm with faults data improved the traditional incremental insertion method and embedded constrained edges. This algorithm improved three traditional triangulation algorithms.

(1) It improved the traditional convex shell generation scheme. The proposed boundary contraction produces a convex hull with high finesse and sensible formation optimizing the point location algorithm.

(2) The algorithm improved the traditional refinement algorithm by setting a longest side. The refinement triangulation is uniform, which is convenient to process the terrain.

(3) The algorithm optimized the processing of the influence field of the constraint line part. The triangulation with constrained line parts is superior to traditional algorithm in time and efficiency. The algorithm can be extended in three-dimensions to provide a good direction for practical modeling and has good practical value.

The research progresses further on 3D triangulation method with faults data. This method will be extended from 2D to refine the 3D tomographic data triangulation to produce a high precision model with better visualization effect.

Acknowledgements

This work was supported by the National Natural Science Foundation No. 41472234.

References

[1] M.-L. Yin, J.-X. Chen, Z.-R. He, Algorithm of drawing isoline based on improved Delaunay triangle net, in: Proc. 12th IEEE Conference on Industrial Electronics and Applications (ICIEA), 2017.

[2] W.-Z. Shao, F.-Y. Zhao, Z.-Y. Liang, The research of improved progressive triangulated irregular network densification filtering, Beijing Surveying and Mapping 6(2016) 17-21.

[3] K.K Sang, F. Andrew, Peterson, Adaptive h-refinement for the RWG-Based EFIE, IEEE Journal on Multiscale and Multiphysics Computational Techniques (2018) 58-65.

[4] S.-J. Tan, J.-J. Zheng, H.-J. Zhou, A cutting remesh method based on barycentric coordinates for 2D triangulation mesh, in: Proc. 2nd IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), 2018.

[5] T. Long, General type-2 fuzzy logic systems based on refinement constraint triangulated irregular network, Journal of Intelligent & Fuzzy Systems 25(3)(2013) 771-784.

[6] C. Bedregal, M.C. Rivara, Longest-edge algorithms for size-optimal refinement of triangulations, Elsevier Ltd 46(2014) 246-251.

[7] L.-p. Gewali, B. Acharya, Stability aware Delaunay refinement, in: Proc. 11th International Conference on Information Technology: New Generations, 2014.

[8] J.-P. Hu, X. Li, Q. Xie, L. Li, D.-C. Zhang, An unconstrained optimization EMD approach in 2D based on Delaunay triangulation, Journal of Shandong University (Engineering Science) 48(5)(2018) 9-15+37.

[9] P.-L. Wang, Design and implementation of finite element mesh generation and refinement algorithm, Northeastern University (2013).

[10] M.A.N.I. Fahim; S. Mostafa; J. Tasnim; A.B.M. Aowlad Hossain, Alignment of 3-D scanning data for polygonal mesh based on modified triangulation, in: Proc. 6th International Conference on Informatics, Electronics and Vision & 7th International Symposium in Computational Medical and Health Technology (ICIEV-ISCMHT), 2017) 1-5.

[11] Q.-Q Liu, Overview of point location algorithm of Delaunay triangulation on plane domain, Electronic Design Engineering 25(1)(2017) 47-51.

[12] Q.-H Zhang, Z.-Y. Xie, Mixed algorithm of Delaunay triangular subdivision with fault constraint, Journal of Xi'an University of Science and Technology 34(1)(2014) 52-56.

[13] M.-J. Chen, Y.-M. Fang, G.-Z. Li, J. Chen, An improved generation algorithm of Delaunay triangulation, Journal of Kunming University of Science and Technology (Natural Science) 41(5)(2016) 33-38.

[14] Q.-F. Sun, W. Zhao, Y.-X. Duan, P.-G. Liu, Research of contour generation algorithm for complex heterogeneous reservoir computer engineering and applications, Application Research of Computers 36(5)(2019) 1581-1584.

[15] J.-P. Li, M. Xu, Using point angle to improve growth algorithm for Delaunay triangulation, Geospatial Information 16(2)(2018) 82-84+12.

[16] Q.-Q Liu, Research and implementation of Delaunay triangulation algorithm in planar domain, Computer Science and Technology (2016) 20-24

[17] H.-L Jin, L.-L Li, S.-H Yuan, W.-X. Geng, An algorithm of Delaunay triangulation network growth based on rectangular ring partitioning, Henan Polytechnic University (Natural Science) 36(6)(2017) 63-68.

[18] J.R. Shewchuk, B.C. Brown, Fast segment insertion and incremental construction of constrained Delaunay triangulations, Computational Geometry: Theory and Applications 48(8)(2015) 554-574.