2D Mesh Generation Scheme for Adaptive Remeshing Process in Finite Element Method*

Akira TEZUKA**

The discretization of a whole domain is a prerequisite process for Finite Element Method (FEM). Since manual mesh generation is tedious, mesh generation schemes have been studied in recent years. In iterative analyses, the geometry of the domain is drastically changed by each iteration. It is known that mesh regeneration eliminates mesh distortion caused by the geometry change, which greatly affects the convergence of the whole analysis. If error control in the analysis is additionally considered in the mesh regeneration, such a process should be adaptive; adaptive remeshing is required for a reliable solution. There are many mesh generators available, however, most of them require interaction with the user, and thus cannot be used for adaptive remeshing. In this paper, a sophisticated mesh generation scheme for the adaptive remeshing in the 2D FEM is discussed in detail. After comparison with other schemes, it is concluded that our scheme is superior to the others in terms of flexibility and time complexity.

Key Words: Computational Mechanics, Numerical Analysis, Finite Element Method, Automatic Mesh Generation, Adaptive Remeshing Process, Time Complexity

1. Introduction

The Finite Element Method (FEM) requires a lot of discretized data as a prerequisite for the analysis. Since preparation of such data by hand is tedious, automatic uniform mesh generation schemes have been developed. Recently FEM has been frequently applied to nonlinear problems, in which a single analysis is applied iteratively until the solution converges. Since the shape of the domain is drastically changed during this process, if the mesh is fixed and not regenerated, it is distorted. The performance of FE analysis is very sensitive to the geometry of the finite element. If the mesh is distorted, the FE solution has considerable error. If even one pixel of the mesh is crushed, the FEM analysis cannot be continued. Therefore it is necessary to regenerate the mesh (remeshing) as the domain changes during the itera-

tion process, to obtain acceptable results. Since the user is not involved in the iterative process, the mesh generator must follow changes in the domain automatically. To do so, the following conditions are required for a uniform mesh generator.

- (1) The only input data required for the mesh generator are geometrical data for the domain.
- (2) The mesh generator is applicable to any domain with arbitrary shape.
- (3) The reliability and time complexity of the algorithm are good.
- (4) The shape of the generated mesh is distortion-free and traceable.

However, a uniform mesh is not sufficient to obtain reliable solutions in FEM. The quality of the FE analysis strongly depends on the quality of the mesh. Depending on the characteristics of the problem, some regions of the domain should be refined to catch stress concentration, for example. To generate such an improved mesh manually is tedious and requires much experience and engineering intuition. Therefore, automatic mesh refinement is an important feature of a mesh generator. Since the exact solution and the error in the FE approximated solution are

* Received 31st March, 1995. Japanese original : Trans. Jpn. Soc. Mech Eng., Vol. 60, No. 576, A(1994), pp. 1863-1870. (Received 20th December, 1993)

** Mechanical Engineering Laboratory, Ministry of International Trade and Industry, Namiki 1-2, Tsukuba, Ibaraki 305, Japan

Table 1 Advantages and disadvantages of each mesh generation method

	(1)	(2)	(3)	(4)	(5)
a	×	×	○	×	×
b	×	×	○	○	×
c	○	○	○	○	×
d	○	○	×	○	○
e	○	○	×	○	○

unknown, we can estimate some value related with error in the initial analysis and improve the mesh based on this estimated value for the next analysis. This is the basic principle of mesh adaptation. Mesh regeneration with adaptation is called adaptive remeshing. A mesh generator for adaptive remeshing should meet one of the following requirements.

(5) The mesh generator should be applicable to arbitrarily shaped multidomains, or

(5') The mesh generator should be applicable to an arbitrary distribution of nodes.

Methods of automatic mesh generation can be classified as follows. Table 1 shows which of the requirements mentioned above is satisfied by each method.

- Mesh generator with mapping
- Boundary-fitted method
- QUAD tree method
- Advancing front method
- Computational geometry based mesh generator

Typical mesh generators which use mapping are the serendipity mapping method and the conformal mapping method. These methods provide good quality meshes with good time complexity, but they are not applicable to arbitrary domains. The serendipity mapping method was introduced by Zienkiewicz and Phillips⁽¹⁾ as an application of the concept of isoparametric shape functions to finite element approximations. This method is most widely applied in structural mechanics and dynamics, because the basic principle is exactly the same as in the shape function in FEM. Since it is possible to define serendipity shape functions of any order, this method can theoretically be used to generate finite elements for any arbitrary domain, only if it is decomposed into several simple-shaped subdomains before mesh generation. However a computer cannot handle this tedious process and it must be performed by an analyst. Therefore, without an automatic pre-decomposition scheme, the serendipity method cannot be used for remeshing, although it is simple and widely used. The conformal mapping method is based on the flow net in the potential flow field, and is only applicable to two-dimensional

domains. Although the meshes generated are of very good quality, its application is limited, and it becomes a stepping stone for the elliptic mesh generation method.

The elliptic mesh generation method is the most widely used in computational fluid dynamics based on finite difference methods with curve fitted boundaries. Meshes are obtained by solving elliptic second-order differential equations, which are generalizations of the Laplace equation for the conformal mapping method. Although this method provides smoothly graded finite elements, which follow the singular behavior of the solution, it cannot deal with complicated geometry. In particular, if the boundaries consist of many segments, then defining the information necessary for the elliptic method requires a considerable amount of work. Furthermore, the quality of the generated mesh strongly depends on the way that the boundary segments are assigned as the boundary condition in the elliptic differential equations. For these reasons, this method is not appropriate for adaptive remeshing^{(2),(3)}.

The QUAD tree mesh generator supports a systematic tree type data handling. The principle was derived from the CAD data structure. Information can be transferred from a CAD stage to the mesh generation via a tree data structure, which is a hierarchical. Thus it is easy to refine the mesh locally with good time complexity, although the shape of the mesh is not good generally. One of the disadvantages of this scheme is that it is difficult to fit the tree structure to the boundary of the arbitrary domain. If the QUAD tree mesh generation method does not have a hierarchical structure, it is called the integer net method^{(4),(5)}. The integer net method is based on a simple idea. A net large enough to cover the whole domain is placed on the domain, the cells both outside the domain and on the boundaries are removed, and a zigzag mesh is fitted to the given boundaries. The time complexity of this algorithm is very good, although it cannot be applicable to subdomain data structures or arbitrarily distributed points. It does not adapt the mesh automatically, although Thacker et al.⁽⁴⁾ introduced the concept of mesh adaptation to this method, by eliminating a part of the grid-oriented mesh and reconstructing the mesh in these regions.

In the advancing front method, the mesh is generated element by element, and interception between the generated elements is checked at each stage. Because of the complicated algorithm, the time complexity is not good in this method. However, the only input data required are geometrical information on the boundaries of a given domain and no additional human decision is required. Therefore, this scheme does not depend on the experience of the user, and it generates

objective meshes, i.e., the same mesh is generated, if the same data for the boundaries of a given domain are used. Since this approach is applicable to the subdomain structures, it can be extended to sophisticated mesh adaptation⁽⁶⁾⁻⁽¹²⁾.

The computational geometry based approach has been researched recently. The principal is to use Delaunay tessellation as part of the mesh generator. The algorithm is simple and systematic, in contrast to the advancing front scheme which is based on a trial and error procedure. Since Delaunay tessellation is defined on given nodes, rather than on a given domain, if the domain is not convex, then some elements of the Delaunay tessellation could cross the given boundaries or be located outside the domain. An extra process is therefore required to adapt Delaunay tessellation for the use in a finite element mesh^{(13),(14)}.

Each scheme has both advantages and disadvantages. Considering the five requirements discussed above for adaptive mesh generator, a combined method is the most suitable. The integer net method satisfies all the requirements, except for the concept of mesh adaptation, while the advancing front method meets all the requirements, except for the good time complexity of the algorithm. It is logical to combine these two methods to construct the perfect method which satisfies all the requirements. A new approach which combines the advantages of the integer net method and the advancing front method should result in a suitable mesh generator for adaptive remeshing.

We compare this new approach with the Delaunay tessellation approach. Delaunay triangulation and finite element generation are not equivalent; the former is defined with respect to nodes, while the latter with respect to domains. To convert a Delaunay tessellation to a finite element mesh requires additional computation, even though the time complexity of an ordinary Delaunay tessellation is reasonable. Particularly in quadrilateral element mesh generation, the Delaunay tessellation approach has disadvantage, because it must always go through "triangulation" even with a grid-node distribution, whereas the new approach does not. Delaunay triangulation is such that the minimum angle is equal to the maximum angle in other triangulation methods. However, this is not the case after conversion to a finite element mesh, which means that the element shape is not guaranteed, especially near the boundary. The algorithm in Delaunay triangulation is simple compared with that of the advancing front method, but round-off errors in computation can lead to the failure of mesh generation. Considering these points, the proposed approach seems to be superior to Delaunay tessellation approach as a 2D adaptive

remeshing process.

The logic and algorithm of the new combined approach are explained in this paper. The elements considered are a three-node triangular element (TRI3) and a four-node quadrilateral element (QUAD4) in 2D. This method has already been applied to adaptive remeshing process^{(15),(18)}, the FE analysis of a metal forming process with adaptive remeshing⁽¹⁹⁾ and the analysis of composite materials with adaptive remeshing⁽²⁰⁾, and therefore its reliability is considered to be highly guaranteed. We discuss the time complexity of this mesh generation method with respect to the number of nodes, and compare it with that of other methods such as the advancing front scheme and the Delaunay tessellation scheme. The method of error estimation for adaptive remeshing is not discussed in this paper, but was clarified in our previous papers⁽¹⁵⁾⁻⁽²¹⁾.

2. Algorithm

The algorithm for our two-dimensional mesh generator is represented in Fig. 1.

2.1 Basic logic

Before giving details of our algorithm, the basic logic for two-dimensional mesh generation is explained.

(1) Logic for checking on which side of a plane a point is located

When a triangular element is generated based on a line segment consisting of two nodes, one more node must be selected from many to form a triangle. For lines on boundaries, this node should be inside the domain. To choose a node inside the domain, it is necessary to know on which side of a line segment a node is located.

To explain the logic used to determine the side, we consider the relationship between a line 1 - 2 and

Fig. 1 Algorithm flow for our 2D mesh generator

a point 0. (x_0, y_0) , (x_1, y_1) and (x_2, y_2) are the coordinates of the points 0, 1 and 2, respectively.

As shown in Fig. 2, if the permutation of points 1-2-0 is negative, i.e., clockwise, the determinant of the matrix $\begin{bmatrix} x_1-x_0 & y_1-y_0 \\ x_2-x_0 & y_2-y_0 \end{bmatrix}$ is negative. If the permutation of these points is positive, i.e., counterclockwise, the determinant of the matrix $\begin{bmatrix} x_1-x_0 & y_1-y_0 \\ x_2-x_0 & y_2-y_0 \end{bmatrix}$ is positive. If point 0 is on the line segment, the determinant of the matrix $\begin{bmatrix} x_1-x_0 & y_1-y_0 \\ x_2-x_0 & y_2-y_0 \end{bmatrix}$ is zero.

Therefore, the sign of the determinant D indicates on which side of a plane a point is located.

$$D = \det \begin{bmatrix} x_1-x_0 & y_1-y_0 \\ x_2-x_0 & y_2-y_0 \end{bmatrix} \quad (1)$$

(2) Logic for removing the points outside the domain

In the early stages of mesh generation, grid points are distributed to cover the whole domain, some of which are outside it. The points outside the domain must be eliminated to generate fine elements of a given domain. Therefore logic to remove points outside the arbitrary domain is necessary.

We now consider an arbitrary domain as shown in Fig. 3, and choose a point outside the domain as an observation point. (For example, the observation point $\mathbf{x}_p = \mathbf{x}_{\min} + (\mathbf{x}_{\max} - \mathbf{x}_{\min}) * 1.5$ where $\mathbf{x}_{\min}$ is the minimum location vector in the domain, and $\mathbf{x}_{\max}$ is the maximum location vector.) Now we check if a grid point is inside or outside the domain. We set S to 0 initially, then make a triangle using a two-node portion on the boundary and the observation point. Every two-node portion on the boundary has a direction.

Fig. 2 Three geometrical relationship between a line and a point in 2D

Fig. 3 Arbitrary domain for IN/OUT check

The connectivity of the outer boundary is counterclockwise, while that of the inner boundary is clockwise. Depending on the direction of the two node portion in a triangle, the triangle is considered to be clockwise or counterclockwise (see Fig. 4). We ignore triangles which do not have the point which needs to be checked. If the point is inside a counterclockwise/clockwise triangle, we add $+1/-1$ to S . If the point is on a counterclockwise/clockwise triangle, we add $+0.5/-0.5$ to S . If S is 0 for all the triangles on the outer boundary, then the point is outside the domain. If S is not 0 for all the triangles on the outer boundary, we initialize S and continue to check on the inner boundary. The relationship is opposite for inner boundary. If S is not 0 for all the triangles on the inner boundary, then the point is outside the domain. If S is 0 for all the triangles on the inner boundary, the point is inside the domain.

2.2 Input data required

The following input data are required for the mesh generation method proposed in this work.

The number of subdomains

The number of holes on each subdomain

The last node number on the outer boundary (node numbers are given in a counterclockwise manner)

The last node numbers on the inner boundaries (node numbers are given in a clockwise manner)

The coordinates of nodes on the boundaries of the domain and holes

Parameters for mesh generation (mesh size and ratio of reduced domains)

Based on the numbering rule for the outer and inner boundary, the left side of a line in the direction in which the numbers increase is always inside the domain. Thus by applying the basic logic (1), the inside of the domain can always be recognized. (See Fig. 5(1))

2.3 Generation of grid points

At this stage, there are many possible ways in which to distribute the interior points. For example, grid points defined by an equilateral triangle net or those based on a rectangular net can be used. It is also possible to have grid points with a graded distribution. If an appropriate gradation is chosen for the distribution of grid points, this method can be used to generate very similar finite element discretization to that

Fig. 4 Counting rule

Fig. 5 Process of mesh generation

obtained by applying the elliptic mesh generator. For simplicity, a grid node distribution is chosen in this work. Grid points are generated within a rectangle which covers the whole domain. For the example in Fig. 5 (1), grid points are generated as shown in Fig. 5(2).

If the elements are generated with nodes near the boundaries, their shapes are distorted. We therefore eliminate such points. This idea comes from in Fukuda and Suhara's mesh generator algorithm⁽⁶⁾. To do this, we make a reduced domain and remove the extra nodes outside it. The reduced scale is half of the given mesh size.

Applying the 2nd basic logic procedure to the reduced domain, we can remove grid points outside it.

If there is another subdomain adjacent to this domain, we then locate nodes on the common boundary with the averaged mesh size, to maintain the geometrical connectivity between both subdomains. (See Fig. 5(3))

2.4 Systematic mesh generation at the 1st stage (by the integer net method)

Since the grid points are distributed over the domain, most of the elements can be generated according to a simple rule. We consider a four-point grid surrounded by a dotted line as shown in Fig. 6. Depending on the pattern of the remaining nodes, the element can be generated systematically to some degree.

If there are N_I grid points in the i -direction and

Fig. 6 Numbering relations in a grid in the integer method

Fig. 7 14 Patterns for integer method

Fig. 8 Herringbone triangular mesh pattern

NJ grid points in the j -direction, the coordinates and the node number of the four points in a grid can be easily determined as shown in Fig. 6. Depending on the number of nodes remaining in a grid, the generated pattern is one of 14 patterns, as shown in Fig. 7. If four points remain, the pattern should be #1 or #2 for a TRI 3 element and #3 for a QUAD 4 element. Since an equilateral triangle has angles of 60 degrees, there must be six TRI 3 elements around one node. (With this setting, the angles around a node are almost 60 degrees after applying a smoothing scheme.) The pattern should therefore be a herringbone type as

Fig. 9 Interception check between generated element and reduced boundary

Fig. 10 n th stage of 2D mesh generation by advancing front method

shown in Fig. 8. Depending on the type of herringbone, the #1 or #2 pattern is chosen for the TRI 3 element. If three points remain within a grid, the pattern should be either #4, #5, #6 or #7. If two nodes remain, the pattern is #8, #9, #10, #11, #12, or #13. If one or no points remain within a grid, the pattern is #14 (null).

The generated mesh based only on the patterns is not complete. It is possible that there is some interruption with the boundaries of the reduced domains. Therefore it is necessary to perform an interruption check at the boundaries of reduced domains and to modify the pattern if necessary, as shown in Fig. 9. Once a feasible regular mesh is obtained (see Fig. 5(4) for TRI 3 and Fig. 5(5) for QUAD 4), we regenerate the boundary data with lines used once for the next process, as shown in Fig. 5(6).

2.5 Trial-and-error mesh generation at the 2nd stage (by the advancing front method)

The triangle 1-0-2 is the element generated around node 1 at the n -1th step, and that the line 1-2 is the leading line segment of the n th element, as shown in Fig. 10. Points located on the same side of node 0 with respect to the line 1-2 should be ignored when generating the n th element. For the choice of node 3, the following guidelines are needed.

- (1) Node 3 should be neither node 1 nor node 2.
- (2) If either of the lines 1-3 or 2-3 has already been used twice, this node 3 should be rejected.
- (3) If either of the lines 1-3 or 2-3 crosses any other line used once, then this node 3 should be rejected.
- (4) If there is any line inside the triangle 1-2-3, then this node 3 should be rejected.

(5) If a node is surrounded by elements, then it cannot be node 3.

(6) If a node is far from the line 1-2, compared with the mesh size, it should be neglected.

Under these constraints, the node whose angle between nodes 1 and 2 is the maximum, should be chosen as node 3. Since grid points too close to the boundaries were eliminated by constructing reduced domains in the previous process, the probability of generating excessively distorted elements using the above method is small. Once the optimum node 3 is identified, the triangle 1-2-3 is registered as the n th element. At the same time, the new nodes, new elements, and new lines around node 1 should be registered for use in a later process. If the lines 2-3 and 3-1 have not been used before, they should be registered based on the direction rule in Fig. 10. Under this rule, the left side of a once-used line in the direction in which the numbers increase is always a locally ungenerated area. After determining node 3, we choose the leading line segment for the $n+1$ th element. If either of the lines 1-3 or 2-3 has been used only once, we choose this once-used line as the leading line for the next element. If both lines 1-3 and 2-3 have already been used twice, then the mesh generator should choose another line around node 1 which has been used only once as the leading line for the next element. If there are no such lines around node 1, then node 1 is surrounded by elements, and so the generator finds the next node which is not yet surrounded, and chooses a once-used line around this node.

The computational process for constraints (3) and (4) requires much CPU time. However, meshes can be generated without applying these constraints in most cases. Therefore the following simple modification works well in practice. First, try to generate meshes without applying constraints (3) and (4). If this fails, try again applying these two constraints. Using this modification, the efficiency of mesh generation is increased on an average.

If all the line segments except for those on the boundaries have been used twice and if all the nodes except for those on the boundaries are surrounded by elements, then the mesh generation is finished.

2.6 Modification of mesh topology

(1) Concept of accordion element

In our mesh generator, a node is connected with six elements over most of the domain. However, if the imposed mesh size is considerably different in subdomains, it is possible that the number of elements around a node on the subdomain boundary exceeds six. If a node is surrounded by more than eight elements, the angles around the node are much less

than 60° and therefore the shape of the elements must be distorted. To compensate for this, the concept of the accordion element is introduced.

In Fig. 11, the dotted element represents the accordion element, which can be expanded to form a triangle or flattened to form a line depending on the number of elements connected to the node. There are eight elements around the marked node in the right-hand figure, which becomes four nodes in the left-hand figure by expanding the four lines to form six accordion elements. The number of elements surrounding the inside node is thus reduced from eight to five. By applying this process to all nodes which are surrounded by too many elements, well-shaped meshes are obtained automatically.

(2) Mesh pattern change at a corner

From the analytical point of view, at least two TRI3 elements must share the node at a corner of the domain, particularly when the boundary condition is applied at that node. Figure 12 shows the process; the mesh pattern on the left should be changed to that on the right.

2.7 Conversion of TRI3-QUAD4 mixed meshes to QUAD4 meshes

If pattern #3 is used as the four-node grid pattern at the first stage of mesh generation, TRI3-QUAD4 mixed meshes are generated. Although Zhu et al. proposed a complicated scheme⁽²³⁾, it is easier to convert these mixed meshes to QUAD4 meshes (refined twice) using the algorithm below.

(1) Of two or three possible combinations, choose the pair of triangles which has the minimum norm S defined in Eq. (2), under the following constraints.

$$S = \left\{ \sum_{i=1}^4 \left(\frac{\pi}{2} - \text{the angle of the } i\text{th corner} \right)^{10} \right\}^{1/10} \quad (2)$$

Fig. 11 Concept of accordion element in 2D

Fig. 12 Triangular mesh pattern change at a corner

Fig. 13 Three possible choices and optimal pair of triangles

Fig. 14 Conversion to QUAD4

a) The shape of the pair is convex.

b) No two lines in the pair are straight.

(2) For triangles or quadrilaterals, generate new nodes (marked in black in Fig. 14) at the center and the mid point of each side.

(3) Refine a triangle three times and a quadrilateral four times, as shown in Fig. 14.

2.8 Smoothing scheme

Since elements are generated very differently at each stage, and since elements are generated “independently” in subdomains, they are not regular. To produce smoothly graded finite elements, post-processing must be carried out. There are several ways of smoothing finite elements: including the Gaussian, Isoparametric, and Winslow smoothing schemes. Since the effect of Winslow smoothing⁽²²⁾ is stable and superior to that of other smoothing schemes, we used Winslow's algorithm. In this smoothing scheme, a node n is relocated to the weighted average point of the centroids of the elements connected to it as expressed in Eq. (3), where $\mathbf{x}_n^*$, $\mathbf{x}_e$, w_e and NI are the position of the node n to be relocated, the position of the centroid of element e , the area of element e as a weight and the number of elements surrounding node n , respectively. If one of the elements around node n is turned over, node n should not be relocated. Figures 5(7) and 5(8) show generated TRI3 and QUAD4 meshes, respectively, after applying the smoothing scheme.

$$\mathbf{x}_n^* = \frac{\sum_{e=1}^{NI} \mathbf{x}_e w_e}{\sum_{e=1}^{NI} w_e} \quad (3)$$

Fig. 15 Domain for evaluation of time complexity of mesh generation

Fig. 16 Relationship between CPU time and the number of nodes (TRI3)

3. Discussion

To evaluate the efficiency of our mesh generator, we consider other mesh generation methods. Here, mesh generators using the advancing front method and Delaunay tessellation method are chosen, for comparison. A variety of algorithms are used to generate Delaunay tessellation. Here Watson's algorithm⁽²⁴⁾ is chosen for its simplicity.

We consider the time complexity of our mesh generator. First we evaluate the case of uniform mesh generation, and then extend the discussion to a non-uniform mesh in a multi-domain case. The advancing front method, the mesh generator based on Watson's algorithm (see appendix), and the proposed mesh generator are chosen for evaluation. We discuss the relationship between the computation time (CPU time) for mesh generation (except for renumbering process to minimize the band width of the stiffness matrix) and the number of given nodes n . The calculation was performed by Sun Sparc 2 station. The domain considered is shown in Fig. 15.

Figure 16 shows the relationship between the CPU time required to generate a TRI3 element and the number of nodes on a log-log scale, and Fig. 17 shows the relationship for a QUAD4 element. From these two graphs, the order of the time complexity of our mesh generator, the advancing front method and

the mesh generator using Watson's algorithm are $O(n^{1.04})$, $O(n^{1.57})$ and $O(n^{1.64})$ for the TRI 3 element, and $O(n^{1.19})$, $O(n^{1.80})$ and $O(n^{1.69})$ for the QUAD 4 element, respectively. These results are explained as

Fig. 17 Relationship between CPU time and the number of nodes (QUAD4)

Fig. 18 Mesh-generated regions at two stages for two mesh sizes

follows. Our method consists of two parts. Most of the mesh apart from the boundaries is generated by the integer net method at the 1st stage, which has a time complexity of $O(n^{1.00})$, and the remaining part of the domain near the boundaries is generated by the advancing front method in the 2nd stage, which has a time complexity of $O(n^{1.57})$. Figure 18 shows the ratio between the two areas for mesh sizes of 2.8 and 0.8. Since the number of elements generated by the advancing front method is constant, as shown in Fig. 18, the computational time at the 2nd stage is constant, which means that the time complexity of our mesh generator is almost linear with respect to the number of given nodes n . As described in the introduction, Delaunay tessellation always involves triangulation, which is a time-consuming process in the case of QUAD 4 mesh generation. Since the best algorithm for generating Delaunay tessellation is of the order $O(n \log n)^{(25)}$ (to convert Delaunay tessellation to finite element meshes requires more computational time), our algorithm has good time complexity. It can be concluded that our mesh generator is useful for generating a number of uniform elements.

We now extend the discussion to the multi-subdomain case. If the mesh size is large compared to the subdomain size, mesh generation is performed using only the advancing front method, which means that there is no computational advantage over the integer net method. However the relationship represented by Eq. (4) is valid for the multi-subdomain case using our mesh generator, where m , n_k , a_k , $a_{\min}$ and a_{eq} are the number of subdomains, the number of nodes in the k th subdomain, the time complexity for the k th subdomain, the minimum value of a_k and the equivalent value of the time complexity with respect to the whole domain, respectively. The Delaunay tessellation approach cannot be applied to each subdomain, because of the problems involved in the conversion to a finite element mesh. Therefore even if mesh generation is performed using only the advancing front method in the subdomains, the equivalent time complexity a_{eq} is much smaller than that of Delaunay tessellation mesh generation over the whole domain. It should be noted that generation of subdomain data is necessary in our scheme to generate an adaptive mesh, whereas generation of node distribution is necessary in the Delaunay tessellation approach. Figure 19 shows an example of non-uniform node of distribution. Figure 20 shows meshes generated by our scheme and the Delaunay tessellation scheme, which required computational times of 12.2 seconds and 21.6 seconds, respectively.

$$\left(\sum_{k=1}^m n_k \right)^{a_{\min}} \geq \sum_{k=1}^m n_k^{a_k} \left(= \left(\sum_{k=1}^m n_k \right)^{a_{\text{eq}}} \right) \quad (4)$$

Fig. 19 Example of non-uniform node distribution

by proposed scheme

by Delaunay tessellation scheme

Fig. 20 Non-uniform meshes

4. Conclusions

A two dimensional automatic mesh generator for adaptive remeshing is developed. Our mesh generator consists of two parts; the first process uses the integer net scheme to generate a grid mesh over most of the domain according to a simple rule, and the second process uses the advancing front method to generate the remaining part of the domain by trial and error. It is determined by numerical experiments that the time complexity with respect to the number of nodes required to generate a uniform mesh using our mesh generator is $O(n^{1.04})$ for TRI 3 elements and $O(n^{1.19})$ for QUAD 4 elements. Since the proposed mesh generator is applicable to geometrical data with multi-subdomains, it can also be used for non-uniform mesh generation. It is concluded that our mesh generator is suitable for adaptive remeshing, in which the number of elements is assumed to be greatly increased.

Acknowledgment

The author wishes to thank Mr. Osamu Okuda at the National Institute of Materials and Chemical Research at MITI in Japan for discussion, and Professor Noboru Kikuchi at the University of Michigan in the USA for providing useful advice on writing this paper.

Appendix

(1) Definition of Delaunay tessellation

If an element is such that other $n+1$ -node elements have no nodes inside the hypersphere of the considered element in the n -dimensional case, it can be defined as a Delaunay tessellation in computational geometry. The Delaunay tessellation is defined uniquely with respect to given nodes, unless more than $n+2$ nodes are located on the hypersphere of the considered element (degenerate case).

(2) Watson's algorithm⁽²⁴⁾

The algorithm for the 2 D case can be summarized as follows;

0) Generate a Delaunay tessellation with 4 grid-distributed points, to cover the whole domain.

1) Insert a point in the Delaunay tessellation.

2) Identify all elements whose hypercircle encloses the inserted point.

3) Delete these elements in order to make a convex hull.

4) Generate elements, using the inserted node and the convex hull.

5) Repeat steps 1 to 4 until all points are inserted.

Watson's algorithm involves so called IN/OUT check at step 2 above. Since digital computers usually have a round-off error in numerical calculations, the

probability of the IN/OUT check failing is high, even if a very small value ϵ is considered as "zero" in the calculation.

Theoretically, the time complexity is $O(n^{2-1/d})$ for "well distributed" points, where n is the number of nodes and d is the number of dimensions. Sloan's algorithm⁽²⁶⁾ is an improved version of Watson's. The best time complexity for Delaunay tessellations among a variety of algorithms is $O(n \log n)$ ⁽²⁵⁾

(3) Conversion of Delaunay tessellation to finite element mesh

A Delaunay tessellation and a finite element mesh are different. Since a Delaunay tessellation is defined by given nodes, rather than a given domain, there is a great possibility that Delaunay tessellation does not fit the given boundary. To convert a Delaunay tessellation to a finite element mesh requires additional processes, including removal of the meshes outside the given domain, deleting of the meshes interrupted by the boundary and fitting them to the boundary. Extra time may be required to carry out this additional process. Taniguchi and Ohta⁽²⁷⁾ proposed a modified Delaunay tessellation scheme in two dimensional case, in which fitting to the real boundary is processed simultaneously with Delaunay tessellation.

References

- (1) Zienkiewicz, O.C. and Phillips, D.V.; An Automatic Mesh Generation Scheme for Plane and Curved Surfaces by Isoparametric Coordinate, *Int. J. Numer. Methods Eng.*, Vol. 3 (1971), p. 519.
- (2) Thompson, J.F., Warsi, Z.U.A. and Mastin, C. W., *Numerical Grid Generation*, (1985), North-Holland, New York.
- (3) Ghia, K.N. and Ghia, U. ed., *Advances in Grid Generation*, ED-15, (1983), ASME, New York
- (4) Thacker, W.C., Gonzalez, A. and Putland, G.E., A Method for Automating the Construction of Irregular Computational Grids for Storm Surge Forecast Models, *J. Comput. Phys.*, Vol. 37 (1980), p. 371.
- (5) Jo M. and Funada K., Controllable Mesh Generator for Finite Element Method (in Japanese), *Trans. Jpn. Soc. Prec. Mech. Eng.*, Vol. 47 (1981), p. 399.
- (6) Fukuda, J. and Suhara, J., Automatic Mesh Generation for Finite Element Analysis, In *Advance in Computational Methods in Structural Mechanics and Design*, Ed. by Oden, J.T. and Yamamoto, Y., (1972), p. 607, UAH Press.
- (7) Cavendish, J.C., Automatic Triangulation of Arbitrary Planar Domains for the Finite Element Method, *Int. J. Numer. Methods Eng.*, Vol. 8 (1974), p. 679.
- (8) Shaw, R.D. and Pitchen R.G., Modifications to the Suhara-Fukuda Method of Network Generation, *Int. J. Numer. Methods Eng.*, Vol. 12 (1978), p. 93.
- (9) Lo, S.H., A New Mesh Generation Scheme for Arbitrary Planar Domains, *Int. J. Numer. Methods Eng.*, Vol. 21 (1985), p. 1403.
- (10) Jo M. and Akimoto Y., Computer Program of Harmonized Mesh Generation for Finite Element Method (in Japanese), *Trans. Jpn. Soc. Prec. Mech. Eng.*, Vol. 46 (1980), p. 286.
- (11) Okuda, O., Automatic Triangular Mesh Generation of Arbitrary Planar Domains Utilizing the Node-Pattern for Finite Element Analysis (in Japanese), *Trans. Jpn. Soc. Prec. Mech. Eng.*, Vol. 51 (1985), p. 802.
- (12) Tezuka, A. and Okuda O., Automatic 2 D Mesh Generation (in Japanese), *Prepr. of Jpn. Soc. Prec. Mech. Eng.*, (1985), p. 23.
- (13) Schroeder, W.J. and Shephard, M.S., Geometry-Based Fully Automatic Mesh Generation and the Delaunay Triangulation, *Int. J. Numer. Eng.*, Vol. 26 (1988), p. 2503.
- (14) Yagawa, G., Yashimura N. and Nakao K., An Automatic Mesh Generation System based on Fuzzy Knowledge Processing (in Japanese), *Trans. Jpn. Soc. Mech. Eng.*, Vol. 56, No. 532, A (1990), p. 2593.
- (15) Tezuka, A., A Development of Automatic Mesh Generator with Arbitrary Geometry-based Input Description, *Masters of Science Thesis*, The University of Michigan, Ann Arbor, (1989).
- (16) Tezuka, A., An Adaptive Mesh Refinement for the Finite Element Method (2nd report, h-Method by Global Remeshing Scheme) (in Japanese), *Trans. Jpn. Soc. Mech. Eng.*, Vol. 57, No. 534, A (1991), p. 436.
- (17) Tezuka, A. and Kikuchi N., An Adaptive Mesh Refinement for the Finite Element Method (3rd report, Convergence at h-Global Remeshing Scheme) (in Japanese), *Trans. Jpn. Soc. Mech. Eng.*, Vol. 58, No. 555, A (1992), p. 2199.
- (18) Tezuka, A., Adaptive Remeshing Process with Quadrangular Finite Elements, *Advances in Engineering Software*, Vol. 15 (1992), p. 185.
- (19) Yukawa, N., Kikuchi, N. and Tezuka, A., An Adaptive Remeshing Method for Analysis of Metal Forming Processes, *Advanced Technology of Plasticity 1990*, (1990), p. 1719.
- (20) Lin, G. and Kikuchi N., Adaptive Remeshing Methods for Finite Element Simulation of Composite Material Processing, *CED-5/AMD-156*, ASME, (1992), p. 37.
- (21) Tezuka, A. and Okuda, O., An Adaptive Mesh Refinement for the Finite Element Method (trial by r-method), *JSME Int. J.*, Ser I, Vol. 31, No. 1, (1988), p. 50.
- (22) Winslow, A.M., Numerical Solution of the Quasi-linear Poisson Equation in a Nonuniform Triangle Mesh, *J. Comput. Phys.*, Vol. 2 (1967), p. 149.
- (23) Zhu, J.Z., Zienkiewicz, O.C., Hinton, E. and Wu, J., A New Approach to the Development of Automatic Quadrilateral Mesh Generation, *Int. J.*

- Numer. Methods Eng., Vol. 32 (1991), p. 849.
- (24) Watson, D.F., Computing the n-Dimensional Delaunay Tessellation with Application to Voronoi Polytopes, *Comput. J.*, Vol. 24 (1981), p. 167.
- (25) Aurenhammer, F., Voronoi Diagrams-A Survey of a Fundamental Geometric Data Structure, *ACM Computing Surveys*, Vol. 23, No. 3 (1991), p. 345.
- (26) Sloan, S.W., A Fast Algorithm for Computing Delaunay Triangulations in the Plane, *Advances in Engineering Software*, Vol. 9, No. 1 (1987), p. 34.
- (27) Taniguchi, T. and Ohta, C., Application of Delaunay Triangulation to Arbitrary 2 D Domain with Straight Boundaries (in Japanese), *Trans. Jpn. Soc. Civ. Eng.*, Vol. 432/I-16 (1991), p. 69.
-