International Journal of CAD/CAM Vol. 1, No. 1, pp. 11~21 (2002)

INTERNATIONAL JOURNAL OF CAD/CAM
www.ijcc.org

Automatic Conversion of Triangular Meshes Into Quadrilateral Meshes with Directionality

Takayuki Itoh[1]* and Kenji Shimada[2]
[1]*IBM Research, Tokyo Research Laboratory*
[2]*Mechanical Engineering, Carnegie Mellon University*

Abstract – This paper presents a triangular-to-quadrilateral mesh conversion method that can control the directionality of the output quadrilateral mesh according to a user-specified vector field. Given a triangular mesh and a vector field, the method first scores all possible quadrilaterals that can be formed by pairs of adjacent triangles, according to their shape and directionality. It then converts the pairs into quadrilateral elements in order of the scores to form a quadrilateral mesh. Engineering analyses with finite element methods occasionally require a quadrilateral mesh well aligned along the boundary geometry or the directionality of some physical phenomena, such as in the directions of a streamline, shock boundary, or force propagation vectors. The mesh conversion method can control the mesh directionality according to any desired vector fields, and the method can be used with any existing triangular mesh generators.

***Keywords*: quadrilateral mesh, triangular mesh, conversion, directionality.**

1. Introduction

In some types of finite element method (FEM) analyses, such as sheet-metal forming simulations and automobile crash simulations, quadrilateral meshes are preferable to triangular meshes because they produce more accurate results more efficiently. Such engineering analyses occasionally require a quadrilateral mesh well aligned along the boundary geometry or the directionality of some physical phenomena, such as along the directions of a streamline, shock boundary, or force propagation vectors.

Although there are many approaches to generating quadrilateral meshes, their capabilities of controlling the mesh directionality are quite limited. The existing quadrilateral meshing approaches include: template matching [1], medial-axis-based decomposition [2], quad-tree decomposition [3-5], advancing front [6-10], and triangular-to-quadrilateral mesh conversion [11-21]. In this paper we focus on the triangle-to-quadrilateral mesh conversion methods, which take advantage of the benefits of triangular mesh generation: (1) a fully-automated meshing process, (2) flexible control of element sizes, and (3) less computation time than the advancing front method. The advancing front methods [6-10] and the triangular-to-quadrilateral mesh conversion methods [18-20] control mesh directionality, but based only on the domain boundary; they cannot create a quadrilateral mesh that aligns well with an arbitrary vector field given by the user.

In this paper we propose a triangular-to-quadrilateral mesh conversion scheme that can control the mesh directionality of an output quadrilateral mesh accurately based on a user-specified vector field. Given a triangular mesh and a vector field, the method generates a quadrilateral mesh. It first scores the geometric irregularity and the directionality error of the quadrilaterals formed by all possible pairs of adjacent triangular elements in the input mesh. It then converts pairs of adjacent triangular elements into quadrilateral elements according to the weighted sum of the shape irregularity and the directionality error. The proposed conversion method can be used with any existing triangular mesh generators.

The remainder of the paper is organized as follows. After reviewing previous mesh conversion methods in Section 2, we describe data structures for triangular meshes and vector fields in Section 3. We then describe the algorithm of our mesh conversion method in Section 4. After discussing our results in Section 5, we offer some conclusions in Section 6.

2. Previous Work

Given a triangular mesh, existing triangular-to-quadrilateral mesh conversion methods [11-21] join pairs of adjacent triangular elements selectively and then convert the pairs into quadrilateral elements. The quality of the output quadrilateral mesh strongly depends on which pairs of triangular elements are joined. The shapes of the quadrilateral elements and the number of triangular elements left in quad-dominant meshes strongly

*Corresponding author:
E-mail : itot@trl.ibm.com

depend on this selection of triangular pairs.

One of the goals of triangular-to-quadrilateral mesh conversion is to maximize the number of triangular pairs. This problem is called maximum matching in graph theory, and there are algorithms available for solving this problem. Suppose the connectivity of input triangular elements is interpreted as an undirected weighted graph, the graph nodes represent triangular mesh elements, and graph edges represent connectivity between mesh elements. Preferable quadrilateral meshes can be obtained by applying a maximum matching algorithm to non-bipartite graphs. This process, however, is computationally expensive, and it does not necessarily create a quad-dominant mesh suitable for engineering analysis. Another approach to solving the mesh conversion problem is to apply integer programming [21], which is also computationally expensive. In most cases a quadrilateral mesh of sufficient quality for engineering analysis can be generated without performing maximum matching or integer programming, as can be seen in many previously proposed mesh conversion methods.

In the rest of this section we survey and categorize previous mesh conversion methods. Note that the common shortfall of these methods is limited control over mesh directionality. Some of the methods can align an output mesh along the domain boundaries, but none can realize a user-defined arbitrary directionality.

2.1. Conversion methods that minimize the number of triangular elements

The methods in this category [11-12] count the number of unprocessed adjacent triangles for each triangle and mark those that have only one unprocessed adjacent triangle as high-priority triangles. These triangles are then extracted and converted into quadrilateral elements with their adjacent triangles. The adjacency of triangles is dynamically updated during the conversion process, and many triangles are therefore marked as high-priority triangles during the process. Finally, many of the marked triangles are converted into quadrilateral elements yielding a quad-dominant mesh.

Since the goal of these methods is to generate all-quadrilateral meshes, they also include post-processing for converting isolated triangles. Heighway [11] proposes a method that swaps the edges of quadrilaterals lying between two isolated triangles until the two triangles become adjacent, as if the two triangles 'walk' toward each other. Johnston *et al.* [12] describe a method that subdivides or swaps edges of isolated triangles until they are locally converted into all-quadrilateral elements.

2.2. Conversion methods that minimize geometric irregularities

The methods in this category [13-17] first calculate the values of a scalar function representing the shapes of the quadrilaterals generated by all possible pairs of adjacent triangular elements. They then convert the triangle pairs into quadrilateral elements in order of the values of this function.

Various functions can be used to evaluate quadrilateral shapes. Lo *et al.* [13] propose an evaluation function defined by the ratio between the shape evaluation values of the four possible triangles generated by dividing the quadrilateral by its two diagonals. Borouchaki *et al.* [17] propose an evaluation function based on the angles of the four vertices of each quadrilateral.

2.3. Advancing front-like conversion methods

In many cases, elements along the domain boundary are the most critical in engineering analysis. Therefore, it is often desirable that elements are well aligned along the domain boundary. Quadrilateral meshes with such well-aligned boundary elements can be generated via triangular-to-quadrilateral mesh conversion by coupling triangles of the input mesh along the domain boundary first.

Shimada *et al.* [20] devised a method that first clusters the input triangular mesh into layered sub-domains along the domain boundary, and then couples the triangles in each cluster. The method generates a topologically regular mesh, and the mesh elements' shapes can be improved by a smoothing process.

Owen *et al.* [18-19] propose the 'Q-Morph' method, which visits front of an input triangular mesh in order and forms quadrilaterals along the visited front edges by re-connecting some edges around the visited front edges. This method generates a high quality quadrilateral mesh well aligned along the domain boundary, similar to a mesh generated by the advancing front method.

3. Preliminaries

In this section we define the data structures for the inputs of the proposed mesh conversion method: a triangular mesh and a desired mesh directionality.

3.1. Data structure of a triangular mesh

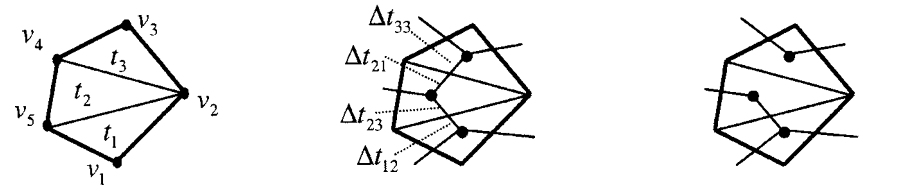
We represent a triangular mesh, M_t, as a planar graph,

$$M_t = (V, T, \partial T, \Delta T), \tag{1}$$

consisting of four ordered lists of:

(1) nodes, $V=(v_1, \ldots, v_l)$,
(2) triangular elements, $T=(t_1, \ldots, t_n)$,
(3) element boundaries, $\partial T=(\partial t_1, \ldots, \partial t_n)$, which defines the three surrounding nodes of each triangle, and
(4) adjacent elements, $\Delta T=(\Delta t_1, \ldots, \Delta t_n)$, which gives at most three adjacent triangles for each triangle.

V and T are topological entities in a triangular mesh, and ∂T and ΔT give topological connections between topological entities. The i th element of ∂T, denoted as ∂t_i, represents the counter-clockwise ordered list of the nodes surrounding the i th triangle t_i. Similarly, the i th



(a) nodes and triangles. (b) triangle adjacency. (c) removal of triangle adjacency.

Fig. 1. Triangular mesh representation.

element of the list ΔT, denoted as Δt_i, represents the counter-clockwise ordered list of the triangles adjacent to the i th triangle t_i. The notation Δt_{ij} represents the j th adjacent triangle of Δt_i. The number of adjacent triangles of t_i is denoted by $|\Delta t_i|$.

For example, the representation of the triangular mesh shown in Fig. 1(a) is:

$$M_t=((v_1, v_2, v_3, v_4, v_5), (t_1, t_2, t_3), ((v_1, v_2, v_3),(v_2, v_4, v_3), (v_2, v_5, v_4)), ((\phi, t_2, \phi), (t_3, \phi, t_1), (\phi, \phi, t_2))), \quad (2)$$

where ϕ in ΔT means that there is no triangle adjacent to a given side. In this example, as implied by the expression $\Delta t_1=(\phi, t_2, \phi)$, the triangle t_1 has only one adjacent triangle t_2, so the number of adjacent triangles is one, or $|\Delta t_1|=1$.

In the mesh conversion algorithms given in this paper, adjacencies between triangles are selectively deleted in order to make pairs of triangles. Fig. 1(a) shows an example of nodes and triangles in a mesh, and Fig. 1(b) shows its adjacencies. To delete the adjacency between t_2 and t_3 in Fig. 1(b), Δt_{21} and Δt_{33} are set to ϕ, yielding a new element adjacency,

$$\Delta T=((\phi, t_2, \phi), (\phi, \phi, t_1), (\phi, \phi, \phi)) \quad (3)$$

as shown in Fig. 1(c).

Our mesh conversion method couples adjacent triangles, t_i and t_j, while deleting the adjacency between t_i (or t_j) and its other adjacent triangular elements. The coupling process is repeated until no triangle has an adjacency with more than one other triangular element. Edges shared by each pair of triangles are then deleted, and finally a quad-dominant mesh is generated.

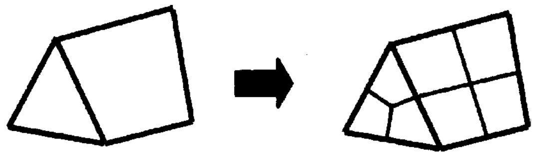


Fig. 2. Quad-dominant to all-quad mesh conversion.

Although the quad-dominant mesh generated by this mesh conversion method contains a small number of triangular elements, it can be converted into an all-quadrilateral mesh by dividing each remaining triangle into three quadrilaterals and dividing each quadrilateral into four quadrilaterals, by adding an inside node for each triangle and by dividing all the edges in two for both triangles and quadrilaterals, as shown in Fig. 2.

3.2. Data structure for desired mesh directionality

One of the inputs of our method is a vector field that represents the user's preferences for mesh directionality. A simple way to represent a vector field is to use a grid so that at each grid-point a vector value is defined. In this paper we assume that the vector field is given as a two-dimensional grid, G, represented as:

$$G=(P_G, D_G) \quad (4)$$

consisting of two ordered lists of:

(1) grid-points, $P_G=((\mathbf{p}_{1,1},\ldots,\mathbf{p}_{1,n}),\ldots\ldots,(\mathbf{p}_{m,1},\ldots,\mathbf{p}_{m,n}))$, and
(2) vector values, $D_G=((\mathbf{d}_{1,1},\ldots,\mathbf{d}_{1,n}),\ldots,(\mathbf{d}_{m,1},\ldots,\mathbf{d}_{m,n}))$.

As shown in Fig. 3, the grid G has $(m-1)\times(n-1)$ cells and $m\times n$ grid-points. The vector value, $\mathbf{d}$, at an arbitrary point, $\mathbf{p}$, can be calculated by the following

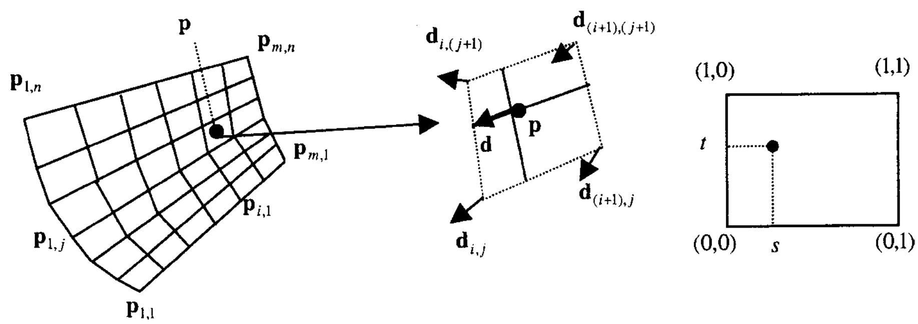


Fig. 3. A 2D grid representing a vector field, and the calculation of a vector value at an arbitrary point.

bi-linear interpolation of vector values assigned to the grid-points:

$$\mathbf{d}(s,t)=(1-s)((1-t)\mathbf{d}_{i,j}+t\mathbf{d}_{i,(j+1)})+s((1-t)\mathbf{d}_{(i+1),j}+t\mathbf{d}_{(i+1),(j+1)}) \quad (5)$$

where (s,t) is the parametric coordinate of point $\mathbf{p}$ calculated by projecting a cell that contains point $\mathbf{p}$.

4. Mesh Conversion with Directionality

This section describes the algorithm of our mesh conversion method. Given a triangular mesh, M_t, and desired mesh directionality, G, the method first scores the shapes and directionality of all the possible quadrilaterals that can be generated by combining pairs of adjacent triangles. The method then converts the pairs of triangles to quadrilateral elements in order of their scores.

Sections 4.1 and 4.2 describe the following two scalar functions used to score a quadrilateral,

(1) ε_{gi} for evaluating the ***geometric irregularity*** of the i th quadrilateral, q_i, formed by coupling two adjacent triangles, and
(2) ε_{di} for evaluating the ***directionality error*** of the i th quadrilateral, q_i.

We then describe the algorithm to pair triangles in an input mesh in Section 4.3. We also describe the algorithm to generate a vector field from a set of input vector values in Section 4.4. In the rest of this paper we represent all possible quadrilaterals formed by joining two adjacent triangular elements and the directions of the edges of the quadrilaterals as the following ordered lists of:

(1) quadrilaterals, $Q=(q_1,\ldots,q_n)$, and
(2) directions of the quadrilaterals' edges, $E=((\mathbf{e}_{1,1}, \mathbf{e}_{1,2}, \mathbf{e}_{1,3}, \mathbf{e}_{1,4}),\ldots,(\mathbf{e}_{n,1}, \mathbf{e}_{n,2}, \mathbf{e}_{n,3}, \mathbf{e}_{n,4}))$.

4.1. Scalar function ε_g for measuring the geometric irregularity of quadrilaterals

In order to measure the geometric irregularity of the i th quadrilateral, q_i, we define the following scalar function:

$$\varepsilon_{gi}=1-\sqrt{2}\frac{r_i}{R_i} \quad (6)$$

Here, as shown in Fig. 4, r_i is the radius of the *minimum inscribed circle*, the smallest circle tangent to at least three edges of an element, and R_i is the radius of the *maximum circumcircle*, the largest circle that goes through at least three vertices of q_i. The radius ratio of the two circles, r_i/R_i, takes its maximum value $1/\sqrt{2}$ for a square, and minimum value 0 for a highly irregular quadrilateral. Therefore, the value of ε_{gi} is 0 in the best case, and 1 in the worst case.

4.2. Scalar function ε_d for measuring the directionality error of quadrilaterals

In order to measure the directionality error of the i th quadrilateral, q_i, we define the following scalar function:

$$\varepsilon_{di}=(2+\sqrt{2})\cdot\left(1-\frac{1}{4}\sum_{k=1}^{4}\frac{\max\{|\mathbf{e}_{i,k}\cdot\mathbf{d}_i|,|\mathbf{e}_{i,k}\cdot(\mathbf{N}\times\mathbf{d}_i)|\}}{|\mathbf{e}_{i,k}|}\right) \quad (7)$$

As also shown in Fig. 5, $\mathbf{d}_i$ denotes the unit vector obtained from the input vector field at the center of the quadrilateral element, and $\mathbf{N}$ denotes the unit normal vector of the quadrilateral element. The value $\max\{|\mathbf{e}_{i,k}\cdot\mathbf{d}_i|, |\mathbf{e}_{i,k}\cdot(\mathbf{N}\times\mathbf{d}_i)|\}$ takes its maximum value 1 for an edge perfectly aligned along the given vector, and minimum value $1/\sqrt{2}$ when the edge and the desired direction form an angle of 45 degrees. Therefore, the value of ε_{di} is 0 in the best case, and 1 in the worst case.

4.3. Coupling of triangle pairs to form quadrilaterals

Two previous sections defined two scalar functions, ε_{gi} and ε_{di}, that measure the geometric irregularity and directionality error, respectively. By taking a weighted sum of these two functions, we define the following metric, ε_i, that decides the order of coupling triangles:

$$\varepsilon_i=(1-a)\varepsilon_{gi}+a\varepsilon_{di} \quad (8)$$
$$0\leq a\leq 1$$

where a is a user-defined weighting factor representing the relative importance of the two measurements. Lower values of a give greater importance to shape regularity than directionality. Values of ε_i for all possible quadrilaterals are first calculated in our algorithm, since they do not change during the entire coupling process. All possible quadrilaterals are then inserted into a list, L, and sorted

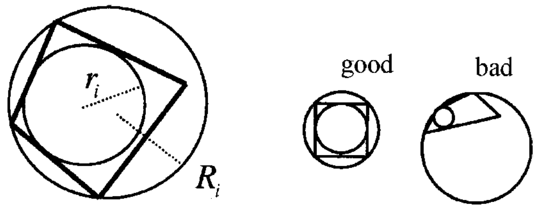


Fig. 4. Function for evaluating the geometric irregularity of a quadrilateral.

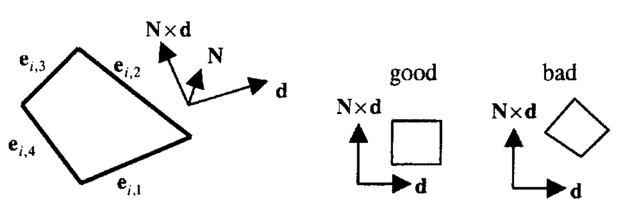


Fig. 5. Function for evaluating the directionality error of a quadrilateral.

```
MeshConversion( M_t , G ) {

  /* Score all possible quadrilaterals */
  for( all t_i ∈ T ) {
     for( all t_j ∈ Δt_i ) {
        form q from t_i and t_j ;
        if( q ∉ L ) {
           calculate the value ε of q ;
           insert q into L ;
        }
     } /* end for( all t_j ∈ Δt_i ) */
  } /* end for( all t_i ∈ T ) */
  sort Q in L by ε values;

  /* Make pairs of triangles */
  while( L is not empty ) {
     extract an quadrilateral q that has
        the smallest ε value from L ;
     suppose two triangles forming q
        as t_i and t_j ;

     if( t_j ∉ Δt_i ) continue;
     for( all t_k ∈ Δt_i ) {
        if( t_k = t_j ) continue;
        delete adjacency between t_i and t_k ;
     }
     for( all t_k ∈ Δt_j ) {
        if( t_k = t_i ) continue;
        delete adjacency between t_j and t_k ;
     }
  } /* end while( L is not empty ) */

  /* Form quadrilateral elements */
  for( all t_i ∈ T ) {
     for( one t_j ∈ Δt_i ) {
        delete the edge shared
           by t_i and t_j ;
     }
  }

} /* end MeshConversion() */
```

Fig. 6. Pseudo code for the mesh conversion method.

by their ε_i values.

The quadrilaterals are then extracted from list L in the order of their ε_i values. Suppose two triangles, t_a and t_b, form an extracted quadrilateral, t_a and t_b's other adjacencies need to be deleted. This process is repeated until the list L becomes empty, and finally no triangle has an adjacency with more than one other triangular element. Edges shared by each pair of triangles are then deleted to form a quad-dominant mesh. The complete procedure for the above algorithm is given in Fig. 6.

Although an output quad-dominant mesh generated by the above algorithm still contains a small number of triangular elements, the mesh can be converted into an all-quadrilateral mesh by applying the templates shown in Fig. 2.

4.4. Automated vector field generation

Although the mesh conversion algorithm described in the previous section requires a desired mesh directionality as a vector field, this vector field need not be provided by the user at all, or it may be provided at only a set of selected locations in the mesh domain. This section describes a method for generating a vector field automatically in these situations.

Suppose that desired mesh directionality is provided by the user as vector values at a set of points in the mesh domain. We denote these points and vector values as:

(1) points, $P_P=(\mathbf{p}_1,...,\mathbf{p}_l)$, and
(2) Vector values, $D_P=(\mathbf{d}_1,...,\mathbf{d}_l)$,

where l is the number of the given points at which the desired mesh directionality is specified.

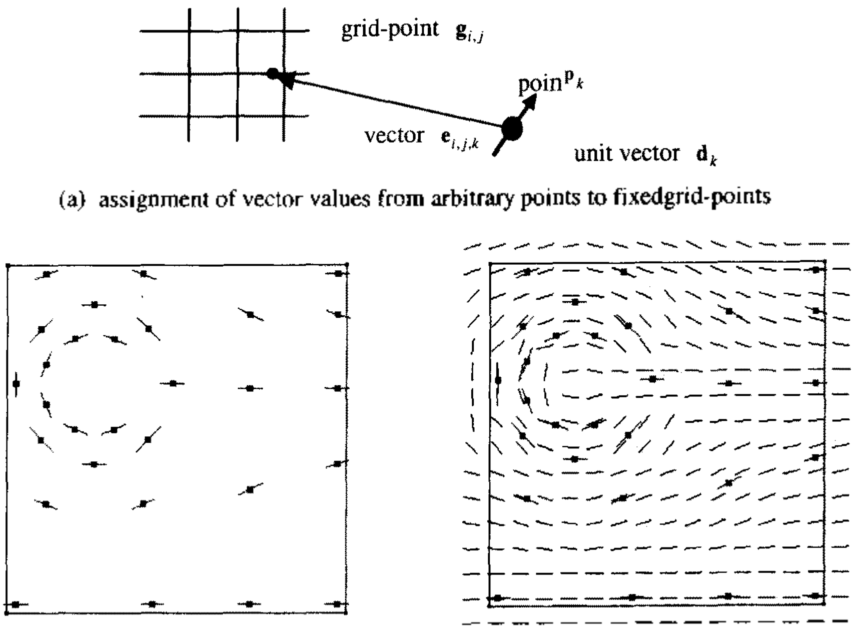
Our implementation assigns vector values to the grid-points of grid G to represent a vector field defined over the entire mesh domain and well aligned along the vector values D_p. We calculate a vector value, $\mathbf{d}_{i,j}$, that is the vector value at a grid-point, $\mathbf{g}_{i,j}$, of a two-dimensional grid using the following formula:

$$\mathbf{d}_{i,j}=\sum_{k=1}^{l}\frac{\mathbf{d}_k}{\left\|\mathbf{e}_{i,j,k}\right\|^2} \tag{9}$$

where $\mathbf{d}_k$ is the given unit vector at point $\mathbf{p}_k$, and $\mathbf{e}_{i,j,k}$ is the vector from point $\mathbf{p}_k$ to grid-point $\mathbf{g}_{i,j}$ as shown in Fig. 7(a). Fig. 7(b) shows an example of a set of input vector values, and Fig. 7(c) shows a complete vector field calculated from the set of input vector values.

This vector averaging technique works best when the input vectors are evenly spaced. When a region has many input vectors clustered together, they tend to outweigh other input vectors. This problem can be avoided by limiting the maximum number of vectors used in a local region.

If it is desirable that the elements be well aligned along the domain boundary, like meshes generated by the advancing front method, our mesh conversion method can generate such meshes by automatically generating a vector field along the domain boundary using the same method described above. To generate such a vector field we take a set of points on the domain boundary and assign vector values at these points according to the boundary direction.



(a) assignment of vector values from arbitrary points to fixedgrid-points

(b) input: a set of points and vector values

(c) output: generated vector field

Fig. 7. Calculation of a vector field from a set of vector values.

5. Results

The new mesh conversion method was implemented in C++ on Unix Workstations (IBM AIX 4.3.2 and SGI IRIX 6.2) and on Windows NT/95/98 PCs.

In order to evaluate the quality of the meshes generated by our conversion algorithm, we define *topological irregularity*, ε_t, in addition to the geometric irregularity, ε_g, and directionality error, ε_d, as defined in Section 4.

We measure the overall geometric irregularity of an output quadrilateral mesh by taking the average of the geometric irregularity of each element, ε_{gi}, as defined in Section 4.1:

$$\overline{\varepsilon_g}=\frac{1}{m}\sum_{i=0}^{m}\varepsilon_{gi} \tag{10}$$

where m is the number of quadrilateral elements. Since the value ε_{gi} takes its minimum value 0 for a square element, a smaller value of $\overline{\varepsilon_g}$ indicates a more geometrically regular mesh.

We measure the overall directionality error of an output quadrilateral mesh by taking the average of the directionality error of each element, ε_{di}, as defined in Section 4.2:

$$\overline{\varepsilon_d}=\frac{1}{m}\sum_{i=0}^{m}\varepsilon_{di} \tag{11}$$

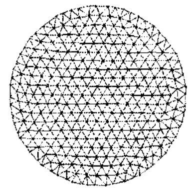
Because the value ε_{di} takes its minimum value 0 for an element perfectly aligned along a given vector field, a smaller value of $\overline{\varepsilon_d}$ indicates a better-aligned mesh.

For topological irregularity, we define the following metric:

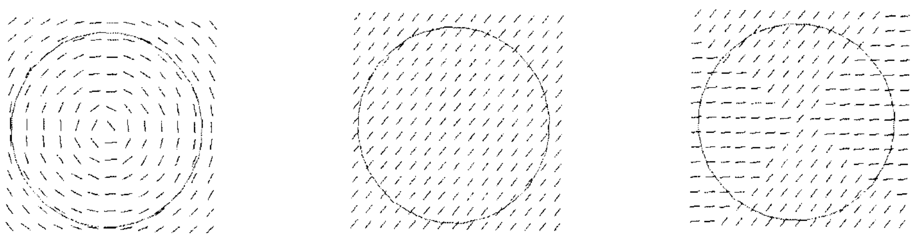
$$\overline{\varepsilon_t}=\frac{1}{n}\sum_{i=0}^{n}|\delta v_i-D| \tag{12}$$

where D=4 for the internal nodes of a quadrilateral mesh, D=2 for the boundary nodes of a quadrilateral mesh, n denotes the number of nodes, and δv_i denotes the number of nodes adjacent to i th node v_i. The topological irregularity $\overline{\varepsilon_t}$ has a positive value that measures how much the mesh differs topologically from a perfectly structured grid mesh. The smaller the value of $\overline{\varepsilon_t}$, the more regular the mesh.

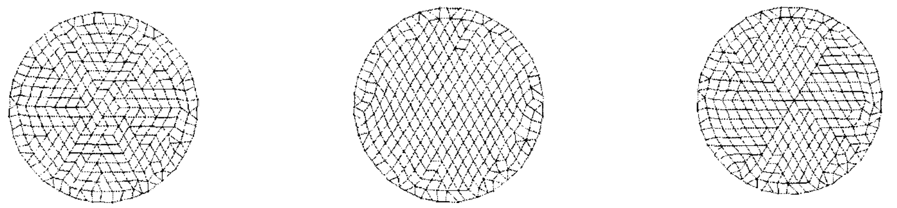
Output quadrilateral meshes differ drastically depending on the input directionality. Fig. 8 shows an example of an input triangular mesh, three different vector fields, the output quad-dominant meshes, the smoothed output quad-dominant meshes, and the smoothed all-quadrilateral meshes. Mesh smoothing is performed by standard Laplacian smoothing, which moves each node to the center of its surrounding nodes. As shown in the left-hand images of Fig. 8, given a directionality along the domain boundary, the method generates a quadrilateral mesh well-aligned along the domain boundary. As shown in the center images of Fig. 8, given a uniform directionality, the method generates a quadrilateral mesh aligned in one direction. As shown in the right-hand images of Fig. 8, given variations in directionality, the



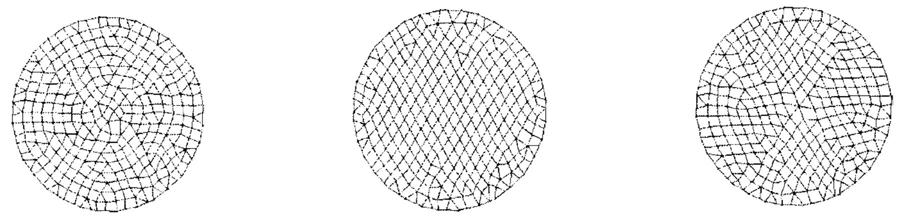
(a) input triangular mesh.



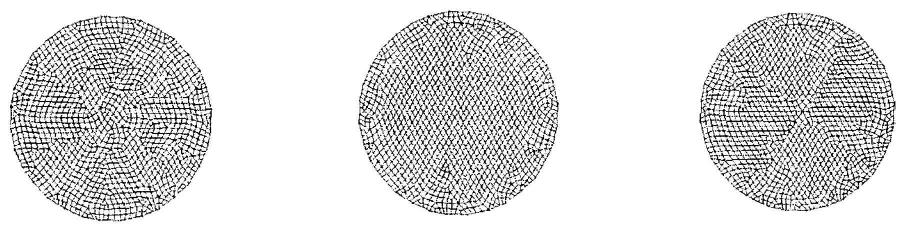
(b) three different input vector fields.



(c) output quad-dominant meshes.



(d) smoothed output quad-dominant meshes.



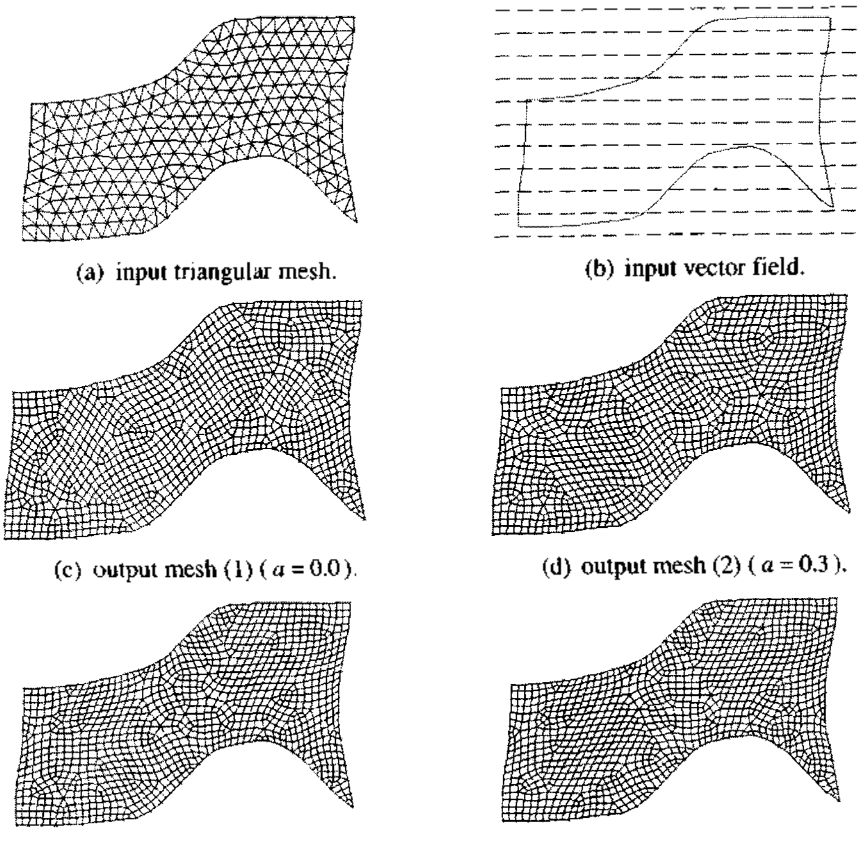
(e) smoothed output all-quadrilateral meshes.

Fig. 8. Output quadrilateral meshes are well-aligned along the input mesh directionality.

method generates a quadrilateral mesh that aligns along the various directions.

The output quadrilateral meshes also vary greatly depending on the value of the weighting coefficient controlling element shape regularity and directionality. Fig. 9 shows an example of an input mesh, an input vector field, and the different smoothed output quadrilateral meshes generated while varying the coefficient value. Table 1 shows the selected coefficient values and the resulting irregularity values. Smaller a values produce the smaller $\overline{\varepsilon_g}$ values, denoting a well-shaped mesh. Larger a values result in the smaller $\overline{\varepsilon_d}$ values, indicating a well-aligned mesh.

The output quadrilateral meshes also diverge depending on the input meshes. Fig. 10(a) and 10(b) show an example of two input triangular meshes that have exactly the same domain boundaries and the same vector field, but the two smoothed output all-quadrilateral meshes are distinct due to the different meshing patterns of the input triangular meshes. Figs. 10(c) and 10(d) show a similar example. Table 2 shows the irregularity values of the output meshes. Note that the domain boundaries, vector fields, and coefficient value are all identical between Fig. 10(a) and Fig. 10(b). Only the input



(a) input triangular mesh.

(b) input vector field.

(c) output mesh (1) (a = 0.0).

(d) output mesh (2) (a = 0.3).

(e) output mesh (3) (a = 0.6).

(f) output mesh (4) (a = 1.0).

Fig. 9. Output quadrilateral meshes vary according to the coefficient value.

Table 1. Coefficient values and irregularity values of meshes in Fig. 9

	Coefficient value	$\overline{\varepsilon_g}$	$\overline{\varepsilon_d}$	$\overline{\varepsilon_t}$
Mesh (1)	a=0.0	0.04932	0.34012	0.28322
Mesh (2)	a=0.3	0.05285	0.27340	0.26224
Mesh (3)	a=0.6	0.07028	0.22332	0.26923
Mesh (4)	a=1.0	0.07682	0.20796	0.28322

Table 2. Coefficient values and irregularity values of meshes in Fig. 10.

	coefficient value	$\overline{\varepsilon_g}$	$\overline{\varepsilon_d}$	$\overline{\varepsilon_t}$
Mesh (1A)	a=0.5	0.07504	0.15359	0.22727
Mesh (1B)	a=0.5	0.02943	0.03439	0.10305
Mesh (2A)	a=0.5	0.13992	0.19212	0.21311
Mesh (2B)	a=0.5	0.03842	0.04097	0.13084

triangular meshes are different. Table 2 shows that all four irregularity values of the output mesh (1B) are much better than those of the output mesh (1A). Similarly, the irregularity values of the output mesh (2B) are much better than those of the output mesh (2A). The input meshes (1B) and (2B) were generated by the square packing method [23], which locates nodes orthogonally and well-aligned along the input vector fields.

It is often desirable that elements are aligned along the domain boundary. The vector fields shown in Fig. 10 were calculated automatically from the domain boundaries of the input meshes by the method described in Section 4.4. Note that the input mesh (1A) in Fig. 10 is exactly the same as the input mesh of Fig. 9, but most of the irregularity values of the output mesh (1A) in Table 2 are superior to those of the output meshes in Table 1. This shows that the vector field calculated automatically by our method results in a high quality quadrilateral mesh.

Fig. 11 shows two more examples of input meshes, vector fields, and output meshes. Input mesh (3) is a graded mesh, and the vector field (3) was automatically calculated from its domain boundary. Input mesh (4) is a uniform mesh, and the vector field (4) has arbitrary directionality. The output meshes (3) and (4) demonstrate that our method works effectively when either graded meshes or arbitrary vector fields are given. Again, the input triangular meshes were generated by the square packing method.

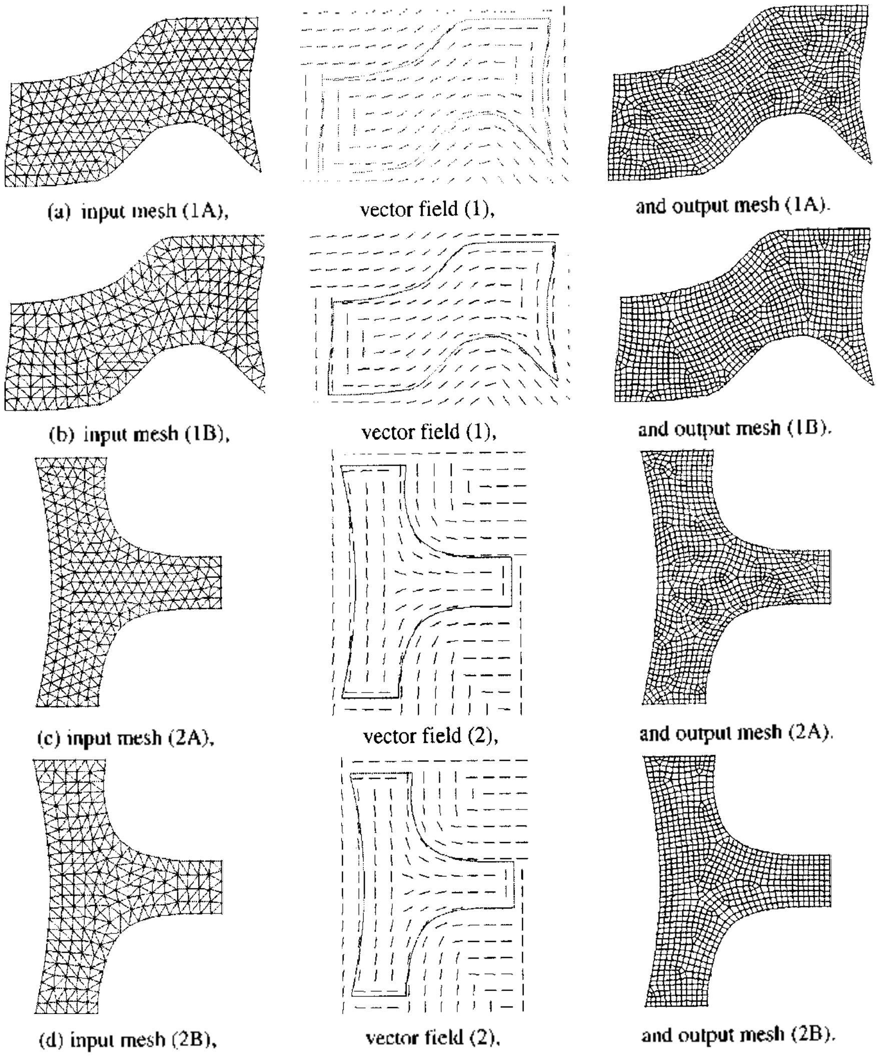


Fig. 10. Output quadrilateral meshes are improved by using the square packing method, in generating input triangular meshes with mesh directionality.

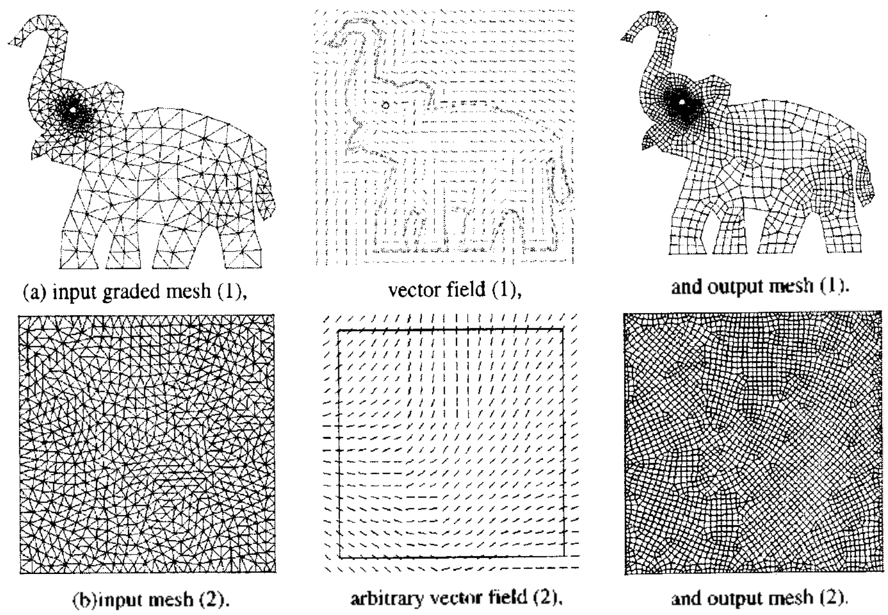


Fig. 11. The mesh conversion method works well even when graded meshes or arbitrary directionalities are given.

6. Conclusion

We have presented a new triangular-to-quadrilateral mesh conversion method that can control the directionality of the output meshes. Our central idea was to use a vector field to represent a user-specified mesh directionality and then to generate quadrilateral elements well-aligned along the vector field. The method first scores, according to their shapes and directionality, all possible quadrilaterals formed by the pairing of adjacent triangles. It then converts the pairs into quadrilateral elements in the order of their scores.

The input mesh directionality can either be: (1) manually specified by the user; (2) automatically generated from the domain boundary; (3) automatically generated from a partial directionality input, or (4) automatically generated from previous analytic results. The method can generate quadrilateral meshes aligned with the input mesh directionality, which is one of the unique features of the proposed mesh conversion method.

Another feature of our approach is the flexible adjustment of the weight between element shape and mesh directionality. Because the importance of these factors depends on the application of the output meshes, it is useful that the method adjusts their respective priorities by changing the coefficient value in the error calculation functions.

References

[1] Ho-Le K. (1988), Finite Element Mesh Generation Method: a Review and Classification, *Computer Aided Design*, **20**(1), 27-38.

[2] Tam T.K.H. and Armstrong C.G. (1991), 2D Finite Element Mesh Generation by Medial Axis Subdivision, *Advances in Engineering Software*, **13**, 313-324.

[3] Yerry M.A. and Shephard M.S. (1983), A Modified-Quadtree Approach to Finite Element Mesh Generation, *IEEE Computer Graphics and Applications*, **3**, 39-46.

[4] Baehmann P.L., Wittchen S.L., Shephard M.S., Grice K.R., and Yerry M.A. (1987), Robust Geometrically-Based, Automatic Two-Dimensional Mesh Generation, *International Journal of Numerical Methods in Engineering*, **24**, 1043-1078.

[5] Shephard M.S. and Georges M.K. (1991), Automatic Three-Dimensional Mesh Generation by the Finite Octree Technique, *International Journal of Numerical Methods in Engineering*, **32**, 709-749.

[6] Blacker T.D. and Stephenson M.B. (1991), Paving: A New Approach to Automated Quadrilateral Mesh Generation, *International Journal for Numerical Methods in Engineering*, **32**, 811-847.

[7] Zhu J.Z., Zienkiewicz O.C., Hinton E., and Wu J. (1991), A New Approach to the Development of Automatic Quadrilateral Mesh Generation, *International Journal for Numerical Methods in Engineering*, **32**, 849-866.

[8] Cass R.J., Benzley S.E., Meyers R.J. and Blacker T.D. (1996), Generating 3-D Paving: An Automated Quadrilateral Surface Mesh Generation Algorithm, *International Journal of Numerical Methods in Engineering*, **39**, 1475-1489.

[9] White D.R. and Kinney P. (1997), Redesign of the Paving Algorithm: Robustness Enhancements through Element by Element Meshing, *Proceedings of 6th International Meshing Roundtable*, 323-335.

[10] Rees M. (1997), Combining Quadrilateral and Triangular Meshing Using the Advancing Front Approach, *Proceedings of 6th International Meshing Roundtable*, 337-348.

[11] Heighway E.A. (1983), A Mesh Generator for Automatically Subdividing Irregular Polygons into Quadrilaterals, *IEEE Transactions on Magnetices*, Mag-19, 2535-2538.

[12] Johnston B.P., Sullivan J.M., and Kwasnik A. (1991), Automatic Conversion of Triangular Finite Element Meshes to Quadrilateral Elements, *International Journal for Numerical Methods in Engineering*, **31**, 67-84.

[13] Lo S.H. (1989), Generating Quadrilateral Elements on Plane and over Curved Surfaces, *Computer and Structures*, **31**(3), 421-426.

[14] Tembulkar J.M. and Hanks B.W. (1992), On Generating Quadrilateral Elements from a Triangular Mesh, *Computers and Structures*, **42**(4), 665-667.

[15] Lee C.K. and Lo S.H. (1994), A New Scheme for the Generation of a Graded Quadrilateral Mesh, *Computers and Structures*, **52**, 847-857.

[16] Borouchaki H., Frey P.J., and George P.L. (1996), Unstructured Triangle-Quadrilateral Mesh Generation, Application to Surface Meshing, *Proceedings of 5th International Meshing Roundtable*, 229-242.

[17] Borouchaki H. and Frey P.J. (1998), Adaptive Triangular-Quadrilateral Mesh Generation, *International Journal for Numerical Methods in Engineering*, **41**(5), 915-934.

[18] Owen S.J., Staten M.L., Canann S.A., and Saigal S. (1998), Advancing Front Quadrilateral Meshing Using Triangle Transformation, *Proceedings of 7th International Meshing Roundtable*, 409-428.

[19] Owen S.J., Staten M.L., Canann S.A., and Saigal S. (1999), Q-morph: An Indirect Approach to Advancing Front Quad Meshing, *International Journal for Numerical Methods in Engineering*, **44**, 1317-1340.

[20] Shimada K., and Itoh T. (1995), Automated Conversion of 2D Triangular Mesh into Quadrilateral Mesh, *Proceedings of International Conference on Computational Engineering Science*, 350-355.

[21] Tajima A., Optimizing Geometric Triangulations by Using Integer Programming, Ph.D. thesis, University of Tokyo, 2000.

[22] Shimada K., Physically-based Mesh Generation: Automated Triangulation of Surfaces and Volumes via Bubble Packing, Ph.D. thesis, Massachusetts Institute of Technology 1993.

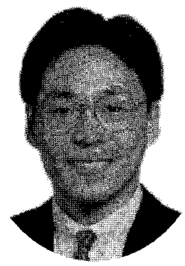
[23] Shimada K., Liao J., and Itoh T. (1998), Quadrilateral Meshing with Directionality Control through the Packing of Square Cells, *Proceedings of 7th International Meshing Roundtable*, 61-75.

[24] Shimada K. and D. Gossard (1998), Automatic Triangular Mesh Generation of Trimmed Parametric Surfaces for Finite Element Analysis, *Computer Aided Geometric Design*, **15**(3), 199-222.

Dr. **Takayuki Itoh** is Research Staff in IBM Research, Tokyo Research Laboratory. Dr. Itoh received his B.S., M.S. and Ph.D. degrees from the Waseda University in 1990, 1992, and 1997. His research interests are in the area of geometric modeling, computer graphics, CAD/CAE, visualization, distributed architecture, mobile computing systems, and information security. Dr. Itoh is a member of ACM and IEEE Computer Society.

Dr. **Kenji Shimada** is Professor at Carnegie Mellon University in the Department of Mechanical Engineering, the Department of Biomedical Engineering, and the Robotics Institute. He received his B.S. and M.S. degrees in Precision Machinery Engineering from the University of Tokyo, and a Ph.D. in Mechanical Engineering from the Massachusetts Institute of Technology. His research interests are in the areas of geometric modeling, FEM mesh generation, computational geometry, computer graphics, CAD/CAM/CAE, and computer assisted orthopedic surgery. Prior to joining Carnegie Mellon in 1996, he was Manager of Graphics Applications at IBM Research, Tokyo Research Laboratory. Dr. Shimada is a recipient of the NSF CAREER Award, IPSJ Best Paper Award, IPSJ Yamashita Award, and Nicograph Best Paper Award. He is a member of ACM, ASME, ASEE, and IEEE.

Takayuki Itoh



Kenji Shimada