Quadrilateral Meshing with Directionality Control through the Packing of Square Cells

Kenji Shimada * **Jia-Huei Liao** †
Carnegie Mellon University

Takayuki Itoh ‡
IBM Research, Tokyo Research Laboratory

Abstract

This paper proposes a computational method for fully automated quadrilateral meshing. Unlike previous methods, this new scheme can create a quadrilateral mesh whose directionality is precisely controlled. Given as input: (1) a 2D geometric domain, (2) a desired node spacing distribution as a scalar function defined over the domain, and (3) a desired mesh directionality as a vector field defined over the domain, the proposed method first packs square cells closely in the domain. The centers of the squares are then connected by Delaunay triangulation, yielding a triangular mesh topology. The triangular mesh is further converted into a quad-dominant mesh or an all-quad mesh that satisfies the given mesh directionality. Since the closely packed square cells mimic a pattern of Voroni polygons corresponding to a well-shaped graded quadrilateral mesh, the proposed method generates a high quality mesh whose element sizes and mesh directionality conform well to the given input.

Keywords: quadrilateral meshing, unstructured grid, mesh directionality, Voronoi diagram, Delaunay triangulation

1 Introduction

Some FEM analyses prefer quadrilateral meshes over triangular meshes. Examples of such analyses include automobile crash simulation, sheet metal forming simulation, and fluid dynamics analysis. It is also known that 4-node quadrilateral elements perform better than 3-node triangular elements when used in FEM analyses of plain stress and strain[15].

Quadrilateral meshing is often a bottleneck in FEM, however, due to its severe requirements of element shape regularity, precise node spacing control, mesh directionality control, and adaptive remeshing capability. These requirements are also common to triangular meshing, with the exception of mesh directionality control, which is unique to quadrilateral meshing. Quadrilateral meshing usually has a desired "mesh flow direction" predicted by boundary geometries or the directionality of physical phenomena to be analyzed using FEM. For example, in fluid dynamics simulation a quadrilateral mesh should align along shock/boundary layers and stream lines. Similarly, in automobile crash simulation, a mesh should align along the direction of force transmission.

Assuming that grid size distribution is given as a scalar field and the directionality is given as a vector field defined over a domain to be meshed, we propose a computational method that creates a well-shaped, well-aligned, graded quadrilateral mesh. The proposed approach is an extension of the *bubble mesh* method that we previously proposed for triangular meshing [23, 21, 22, 29]. In bubble meshing, a well-shaped graded

*Kenji Shimada, Mechanical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, U.S.A.
Tel:(412) 268 3614, Fax:(412) 268 3348, shimada@cmu.edu, http://ciel.me.cmu.edu/shimada/
†Jia-Huei Liao, Mechanical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, U.S.A.
‡Takayuki Itoh, IBM Research, Tokyo Research Laboratory, 1623-14 Shimotsuruma, Yamato, Kanagawa, 242, Japan.

triangular mesh is created by (1) packing an appropriate number of spherical cells, or bubbles, closely in a domain, while the sizes of the spheres are adjusted based on a specified node spacing function, and (2) connecting the bubbles' centers by constrained Delaunay triangulation to generate node connectivity. The novelty of the bubble mesh process is that the close packing of bubbles mimics a pattern of Voronoi polygons that yields well-shaped triangles.

In this paper, we extend the bubble mesh concept for quadrilateral meshing such that we pack square cells, instead of spherical cells, closely in a domain, mimicking ideal Voronoi polygons that yield a well-shaped quadrilateral mesh. Another major extension over the original bubble mesh is to allow the user to specify a desired mesh directionality by a vector field.

The remainder of the paper is organized as follows. After reviewing previous work we outline our basic approach to quadrilateral meshing. We then elaborate on the technical issues of: (1) how to find node locations suitable for quadrilateral meshing, and (2) how to connect the nodes to obtain a mesh topology that aligns along a specified mesh directionality.

2 Previous Work

There are several reviews available of mesh generation methods [27, 5, 9, 19]. Ho-Le, in his comprehensive survey paper [9], gives a classification based on the temporal order in which nodes and elements are created. The resultant classification is well-accepted and referred to by many other researchers. One problem, as Ho-Le acknowledged in the paper, is that some methods do not seem to fit into any class, while others could be put into two or more classes. In fact, as research in mesh generation has matured, most modern algorithms utilize and combine several sub-processes to improve the quality and efficiency of meshing.

In this section, therefore, we summarize and review some of the key sub-processes commonly used in existing quadrilateral meshing methods. These sub-processes include: (1) node placement and connection, (2) mesh template mapping, (3) element-level domain decomposition, (4) grid-based spatial subdivision, and (5) triangular to quadrilateral mesh conversion. One complete meshing scheme can be characterized by a combination of these sub-processes, performed sequentially or merged into a single process.

Common limitations among previously proposed approaches to quadrilateral meshing include: (1) little or no control over mesh directionality; (2) poor control over node spacing, and/or (3) no efficient adaptive remeshing capability.

2.1 Node placement and connection

In this process, a mesh is constructed in two stages: (1) node placement, and (2) node connection. Node placement and connection can serve as a complete meshing process. The process has become popular due to its conceptual simplicity and the availability of a robust mathematical algorithm for node connection, called *Delaunay triangulation*. When Delaunay triangulation is used for node connection the triangular mesh generated must be converted to a quadrilateral mesh by using a mesh conversion process described later under Triangular to Quadrilateral Mesh Conversion.

During node placement, an appropriate number of nodes needs to be inserted in a well distributed configuration. Several early methods use random node placement followed by validity checks[7, 3, 4, 16]. Lee proposed a CSG-based node placement method[12, 13] in which regular node distribution patterns prescribed for all CSG primitives are combined by Boolean set operations into a single set of nodes.

Although most approaches place all the nodes at one time and then connect them at once in another step, in Frey's and Ruppert's methods[6, 18] two stages of node placement and connection are applied in an iterative manner.

Shimada et al.'s *bubble mesh*[25] and Bossen and Heckbert's *pliant method*[2] use proximity-based forces to find node locations suitable for anisotropic meshing.

2.2 Mesh template mapping

When used for 2D meshing or surface meshing, the template mapping technique maps a prescribed simple mesh template such as a square grid into a given four-sided patch using a blending function. This mapping technique has been one of the most popular approaches in commercial software packages. One drawback

of this method, however, is that it is applicable only to topologically simple domains, and thus it is often necessary for users to subdivide the domain manually into a set of simple subdomains. If this manual subdivision is carefully done the mesh directionality can be controlled to some extent. The process, however, is highly labor intensive, and the mesh directionality cannot be controlled in a precise manner.

2.3 Element level domain decomposition

Element-level domain decomposition refers to the process of subdividing a domain to the element level either by: (1) iterative element extraction[1, 28, 17, 14]; or (2) recursive domain splitting to the element level. The former is more suitable for quadrilateral meshing, and the advancing front method, adopted in many modern commercial packages, is one example of such an algorithm. In Blacker and Stephenson's *paving*[1], meshing fronts that start from domain boundaries are advanced to the interior of the domain, generating quadrilateral mesh elements one by one. A mesh created by an advancing front type of method aligns well along boundaries, a desirable characteristic in most engineering analysis. Such a method, however, cannot control a mesh directionality inside the domain or generate a mesh with an arbitrary mesh directionality.

2.4 Grid-based spatial subdivision

Grid-based spatial subdivision methods superimpose a hierarchical grid, similar to a quadtree, onto the domain to be meshed. Such methods are typically followed by a two-step procedure: (1) classification of grid elements into three types, inside/outside/on-boundary; and (2) adjustment of on-boundary elements to make them consistent with the domain boundary. Yerry and Shephard's *modified octree* is a representative method in this category[30, 20]. A mesh created by a grid-based method typically has a strong directionality in the coordinate axis directions, and it is not possible to adjust mesh directionality over a domain.

2.5 Triangular to quadrilateral mesh conversion

It is well known that any triangular mesh can be converted into a quadrilateral mesh by adding a node to the center of each triangle and by dividing the triangle into three quadrilaterals. Although the idea is straightforward and the implementation is simple, this process introduces a significant topological irregularity into a mesh, and thus it is usually not practical.

More sophisticated ways to convert triangles into quadrilaterals are proposed by Heighway[8] and Jonston et al.[10].

Heighway presents a technique for combining two adjacent triangles into a quadrilateral. Isolated triangles remaining in the mesh are then combined by moving them toward each other until they become adjacent and can be combined.

Johnston proposes a three step procedure: (1) extract boundary information from mesh data, and apply Laplacian smoothing; (2) identify and prioritize corner- and boundary-elements, and perform element by element conversion by coupling elements, splitting a coupled element, and propagating the split to maintain the conformity; and (3) combine all isolated triangles into adjacent quadrilaterals, and divide the combined five-sided elements into three quadrilaterals by introducing nodes inside.

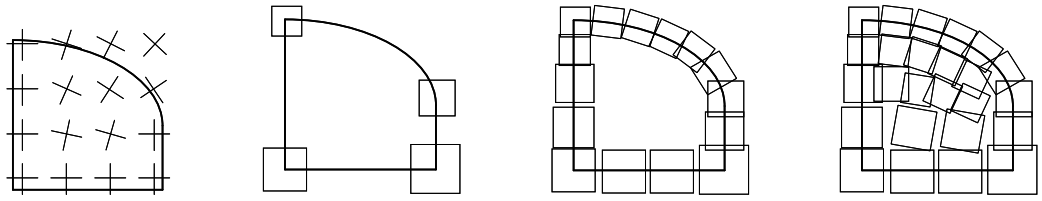
Shimada and Itoh propose a conversion method that uses three conversion templates: (1) from one triangle to three quadrilaterals; (2) from two triangles to four quadrilaterals; and (3) from four triangles to nine quadrilaterals[24]. The method first subdivides a triangular mesh into layers by offsetting boundary, similar to the advancing front method, and then applies conversion templates within each layer.

3 Outline of the Technical Approach

This section describes our basic approach to the following quadrilateral meshing problem.

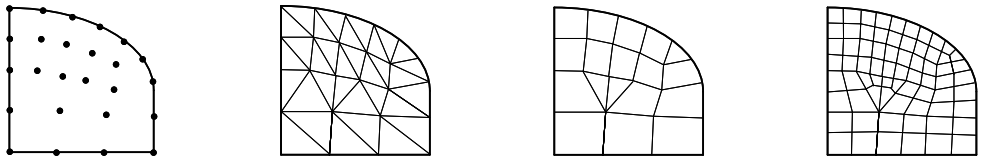
Given:

- a 2D geometric domain
- a desired node spacing distribution $d(\mathbf{x})$, given as a scalar field
- a desired mesh directionality $\mathbf{v}(\mathbf{x})$, given as a vector field



Step 1 Step 2 Step 3

(a) mesh directionality (b) vertex square cells (c) edge square cells (d) face square cells



Step 4 Step 5 Step 6 Step 7

(e) node locations (f) Delaunay triangulation (g) quad-dominant mesh (h) all-quad mesh

Figure 1: Quadrilateral meshing procedure

Generate:

- a well-shaped, graded quadrilateral mesh that is compatible with the given node spacing and mesh directionality

The proposed approach consists of seven steps, as illustrated in Figure 1:

Step 1: Place square cells on all vertices.

Step 2: Pack square cells on all edges.

Step 3: Pack square cells on the face.

Step 4: Place nodes at centers of square cells.

Step 5: Triangulate the domain by Delaunay triangulation.

Step 6: Selectively combine pairs of triangles to generate a quad-dominant mesh.

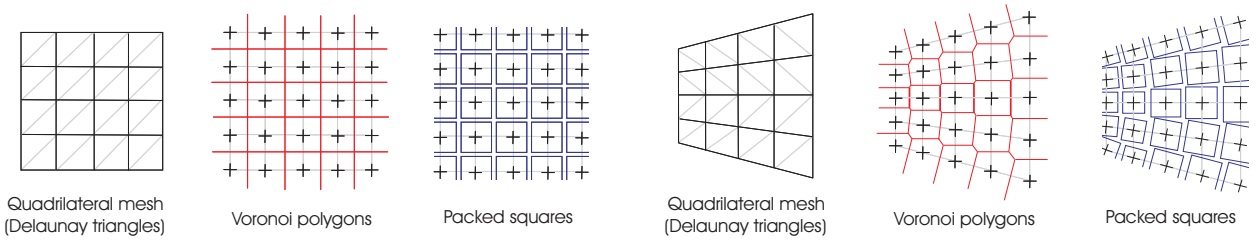
Step 7: Apply mesh conversion templates to obtain an all-quad mesh.

In Steps 1, 2, and 3 we find a node configuration suitable for quadrilateral meshing by closely packing square cells in a domain. The reason we pack squares is that the pattern of packed squares mimics a Voronoi diagram of a well-shaped quadrilateral mesh as shown in Figure 2. Note that the sizes of the cells are adjusted based on a given node spacing distribution $d(\mathbf{x})$ and that the directions of the squares are adjusted based on a given mesh directionality $\mathbf{v}(\mathbf{x})$.

There are two technical issues to be solved in packing square cells tightly in a domain: (1) what are the optimal locations of the squares? (2) how many squares should be packed to fill the domain?

To solve the first issue we use a physically-based model, similar to a particle system in computer graphics. A proximity-based force field is defined between two squares such that the force field exerts an attracting force or a repelling force, moving the cells so that they touch each other along their edges. Also assuming a point mass at the center of each square and the effect of viscous damping, we solve the equation of motion numerically to find a tightly packed configuration of cells.

The second issue of obtaining an appropriate number of squares in the domain is solved by checking the population density and then adaptively adding or removing squares during the numerical integration of the equation of motion, or dynamic simulation.



(a) uniform node spacing (b) non-uniform node spacing

Figure 2: Close packing of square cells for quad meshing

Because square cells are placed in order of dimension (i.e. vertices, then edges, then faces) two fixed squares are already placed at the two endpoints when squares are packed on an edge; these two end squares are stable throughout the packing process, which prevent moving squares from escaping the range of the edge. Similarly, when squares are packed on the face, the boundary edges are already filled with fixed squares, preventing moving squares from escaping the domain. In this way we put higher priority on the cell placement of lower dimensional elements, i.e., vertex square cells over edge square cells, and edge square cells over face square cells. This strategy is sensible because lower order geometric elements are often more critical than higher order elements in FEM analyses.

Once square cells are packed so that they cover the entire domain without significant gaps and overlaps, their centers are connected by Delaunay triangulation (Steps 4 and 5), yielding a triangular mesh. Pairs of triangles are then selectively connected to create a quad-dominant mesh that aligns along the given mesh directionality (Step 6). When an all-quad mesh is required we further apply mesh conversion templates (Step 7). The edge lengths of the mesh elements in Step 7 are reduced by a factor of two compared with the mesh elements in Step 6.

The next two sections, (1) Close Packing of Square Cells and (2) Mesh Topology Generation, describe the essential elements of Steps 1 to 3 and Steps 5 to 7 respectively.

4 Close Packing of Square Cells

In this section we will first discuss how we can generate mesh directionality over the domain. We will then describe how proximity-based forces and potential fields are specified so that square cells repel or attract each other to yield a force-balancing configuration, or a closely packed configuration.

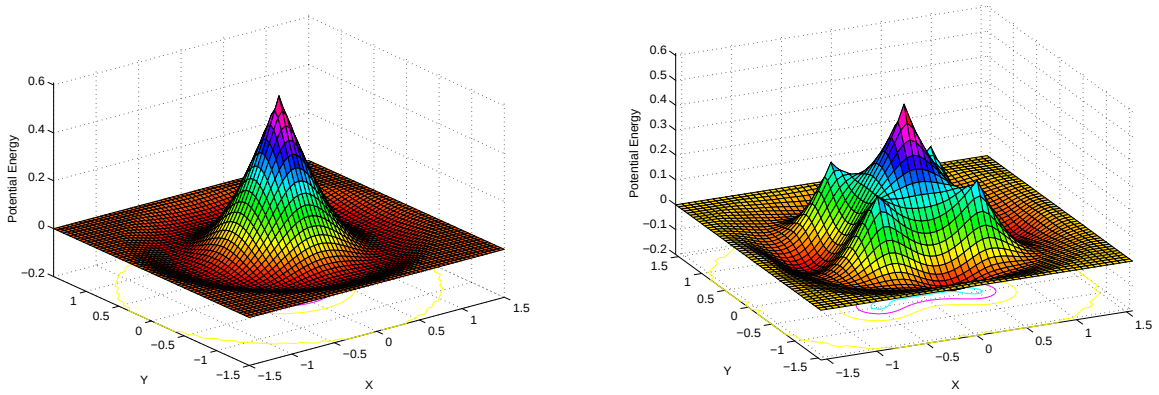
4.1 Mesh directionality

It is important that a desired mesh directionality be specified over the entire domain so that directions of packed square cells are adjusted accordingly. Unless a desired mesh directionality is automatically generated from a previous FEM result, the user typically gives only partial directions or no preference. In such a case it is important that the algorithm generates a complete mesh directionality over the entire domain.

To store a desired mesh directionality we define a background grid that covers the whole domain. The mesh directions are then explicitly stored at the grid nodes, and for an internal point of a grid cell a mesh directionality vector is calculated by linearly interpolating the directions at the four grid nodes.

If mesh directionality vectors are given at only some grid nodes we need to find the mesh directionality vectors at all the others so that the mesh directionality changes smoothly over the domain.

We solve this smooth interpolation problem by using relaxation, similar to Laplacian smoothing, widely used to improve mesh element shapes. As in Laplacian smoothing, which moves a mesh node iteratively to a location which represents the center of gravity of its adjacent node locations, the mesh direction vector at



(a) potential field in the original bubble mesh (b) new potential field for quadrilateral meshing

Figure 3: Potential fields

a grid node is iteratively modified to approach an average of the direction vectors at its four adjacent grid nodes.

4.2 Proximity-based potential fields and forces

In triangular meshing the ideal node configuration is a regular hexagonal arrangement. As proven in the original bubble mesh method [25, 21, 22], such an arrangement can be obtained by defining a force field similar to the van der Waals force, which exerts a repelling force when two molecules are located closer together than the stable distance and exerts an attracting force when two molecules are located farther apart than the stable distance.

Let the positions of adjacent nodes i and j be $\mathbf{x}_i$ and $\mathbf{x}_j$; the current distance between the two nodes $l(\mathbf{x}_i, \mathbf{x}_j)$; the target stable distance $l_0(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{2}\left(d(\mathbf{x}_i)) + d(\mathbf{x}_j)\right)$, which is a desired element size specified by the node spacing function $d(\mathbf{x})$; the ratio of the current distance and the target distance $w(\mathbf{x}_i, \mathbf{x}_j) = \frac{l(\mathbf{x}_i, \mathbf{x}_j)}{l_0(\mathbf{x}_i, \mathbf{x}_j)}$; and the corresponding linear spring constant at the target distance k_0. The force model used in the original bubble mesh is then written as

$$f(w) = \begin{cases} \frac{k_0}{l_0}\left(\frac{5}{4}w^3 - \frac{19}{8}w^2 + \frac{9}{8}\right), & 0 \le w \le 1.5 \\ 0, & 1.5 < w. \end{cases} \tag{1}$$

By integrating the above force field we obtain the following potential field around the center P of the potential field.

$$\Psi_P(w) = \begin{cases} -\frac{k_0}{l_0}\left(\frac{5}{16}w^4 - \frac{19}{24}w^3 + \frac{9}{8}w - \frac{153}{256}\right), & 0 \le w \le 1.5 \\ 0, & 1.5 < w, \end{cases} \tag{2}$$

Figure 3(a) shows this potential field function used in the original bubble mesh for triangular meshing.

This potential field applies either a repelling or attracting force between two nodes based on the following distance comparison. Assuming that two nodes are adjacent to each other, a repelling force is applied if l is smaller than l_0, or if $w < 1.0$. An attracting force is applied if l is larger than l_0, or if $1.0 < w \le 1.5$. No force is applied if two nodes are located exactly at the stable distance or if they are located much farther apart, the cases where $w = 1.0$ or $1.5 < w$. Note that the potential field shown in Figure 3(a) has circular stable positions–anywhere on the circle is equally stable.

In achieving a close packing of squares, however, the potential field shown in Figure 3(a) is not appropriate because it does not take into account mesh directionality, essential to quadrilateral meshing. Considering a mesh directionality there should be only four stable locations around a node, as shown in 3(b), and each of the stable locations corresponds to a situation where two square cells are placed side by side with their edges touching each other. In order to force squares to align this way, we need to add to the original potential

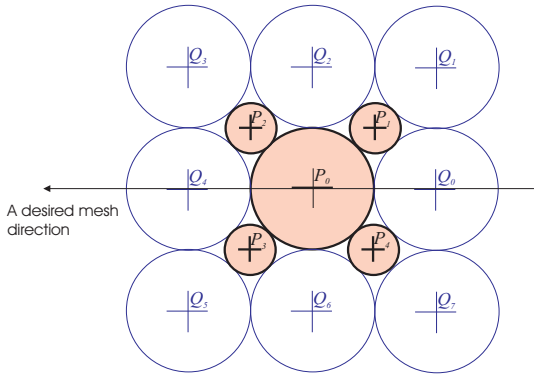


Figure 4: Stable positions in packing square cells

field four sub-potential fields Ψ_{P_1}, Ψ_{P_2}, Ψ_{P_3}, and Ψ_{P_4} at the four corners of a square P_1, P_2, P_3 and P_4 as shown in Figure 4.

If the desired element size is locally uniform the radii of the four sub-potential fields should be $(\sqrt{2}-1)r_0$, where r_0 is the radius of the central potential field Ψ_{P_0}. If graded element sizes are specified, however, the radii of the sub-potentials should be adjusted accordingly.

The potential field shown in Figure 3(b) is thus expressed as a weighted linear combination of the central potential field and the four sub-potential fields, i.e.,

$$\Psi = \Psi_{P_0} + (\sqrt{2}-1)(\Psi_{P_1} + \Psi_{P_2} + \Psi_{P_3} + \Psi_{P_4}). \tag{3}$$

With the above potential field, the primary stable positions of the squares surrounding square P_0 are Q_0, Q_2, Q_4 and Q_6 as shown in Figure 4. Once these primary stable positions are occupied by square cells, then Q_1, Q_3, Q_5 and Q_7 also become stable positions.

4.3 Force-balancing configuration of square cells

Given the proximity-based intercell force, we apply physically-based relaxation to find a close packing configuration of square cells. This is also a configuration that yields a static force balance.

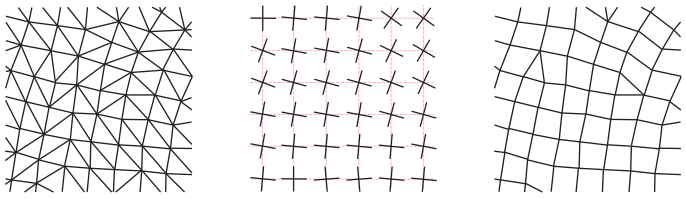
Due to the nonlinearity of the force and complex geometric constraints on square locations, the force balance equation becomes highly nonlinear, and thus it is difficult to solve the equation directly by a multi-dimensional root-finding technique such as the Newton-Raphson method.

Our alternative approach is to assume a point mass m at the center of each cell and the effect of viscous damping c, and to solve the following equation of motion[1] by using a standard numerical integration scheme such as the fourth-order Runge-Kutta method.

$$m\ddot{\mathbf{x}}_i(t) + c\dot{\mathbf{x}}_i(t) = \mathbf{f}_i(t), \qquad i = 1, \ldots, n. \tag{4}$$

In solving Equation (4) numerically, we adaptively adjust the number of square cells packed in the domain. This is important because we do not know beforehand an appropriate number of squares that is necessary and sufficient to fill the region. We generate an initial configuration by using octree subdivision, and although this process gives a reasonably good guess of the number of squares it is still not optimal. We therefore implemented a procedure to check a local population density and to add more squares in sparse areas and delete squares in over-packed areas.

[1] The first order equation can also be used [2]. In either case, the essential point is that after a certain number of iterations the system reaches a virtual equilibrium, where both the velocity term $\dot{x}$ and the acceleration term $\ddot{x}$ approach zero, leaving a static force balance.



(a) triangular mesh (b) mesh directionality (c) quad mesh

Figure 5: Converting a triangular mesh into a quad-dominant mesh

Note that the dynamic simulation and the adaptive node population control described above make efficient adaptive remeshing possible because we do not need to rebuild a mesh from scratch when the domain geometry, node spacing, and/or mesh directionality is slightly modified.

5 Mesh Topology Generation

Once a force-balancing configuration of squares is obtained, the squares' centers must be connected to form a complete quadrilateral mesh. In connecting nodes, *Delaunay triangulation* is first applied to create a triangular mesh, and the triangular mesh is then converted into a quad-dominant mesh by selectively merging two adjacent triangular elements into a quadrilateral element in such a way that the resultant mesh aligns along the specified mesh directionality (see Figure 5). In the final step the quad-dominant mesh is converted into an all-quad mesh by applying two mesh conversion templates: (1) splitting a quad element into four quad elements, and (2) splitting a triangular element into three quad elements.

In converting a triangular mesh to a quad-dominant mesh we use the following three steps so that the resultant mesh aligns along the specified mesh directions. This procedure is based on the practice of removing the shared edge between two adjoining triangles in order to form a quadrilateral element.

1. For the ith non-boundary edge of a triangular element, calculate a score Λ_i that measures how well the resultant quadrilateral element aligns along the specified mesh directions if the edge is removed to form a quadrilateral.

2. Make a priority queue of all the non-boundary edges by sorting the scores assigned to the edges.

3. Delete edges one by one from the top of the priority queue—one edge deletion creates one quadrilateral element.

The quality score Λ_i of a possible quadrilateral element is calculated by comparing the directions of the four side edges of the resultant quadrilateral element with specified mesh direction vectors at the centers of the four edges. For the jth side edge of the quadrilateral element, we take the absolute value of the inner product λ_{ij} of: (1) the unit vector $\mathbf{u}_{ij}$ of the side edge; and (2) the mesh direction vector $\mathbf{v}_{ij}$ at the center of the edge or the unit vector orthogonal to the mesh direction. λ_{ij} is thus expressed

$$\lambda_{ij} = \begin{cases} |\mathbf{u}_{ij} \cdot \mathbf{v}_{ij}|, & |\mathbf{u}_{ij} \cdot \mathbf{v}_{ij}| \geq \frac{1}{\sqrt{2}} \\ \sqrt{1 - (\mathbf{u}_{ij} \cdot \mathbf{v}_{ij})^2}, & |\mathbf{u}_{ij} \cdot \mathbf{v}_{ij}| < \frac{1}{\sqrt{2}} \end{cases} \tag{5}$$

where $j = 1, 2, 3, 4$, the subscript i represents the index of a quadrilateral element, and the subscript j the index of the side edge of the quadrilateral element. Note that the the value of λ_{ij} is bounded between $\frac{1}{\sqrt{2}}$ and 1.

Using the λ defined above we can calculate the score Λ_i as follows, and it measures how well the ith quadrilateral element aligns along the given mesh direction vector $\mathbf{v}(\mathbf{x})$

$$\Lambda_i = \frac{1}{4}\sum_{j=1}^{4}\lambda_{ij} \tag{6}$$

The value of Λ_i is bounded between $\frac{1}{\sqrt{2}}$ and 1, and as Λ_i approaches 1 the ith quadrilateral element aligns more accurately along the mesh direction vector field.

6 Results and Discussions

The proposed quadrilateral meshing algorithm has been implemented in C and C++ on Unix workstations (IBM RS6000 and SGI O2) and Windows PCs.

In this section we measure the quality of generated quadrilateral meshes using two types of mesh irregularity measures, *topological irregularity* and *geometric irregularity*.

For topological irregularity, we use the following measure [24]:

$$\varepsilon_t = \frac{1}{n}\sum_{i=0}^{n}|\delta_i - D|\,, \tag{7}$$

where δ_i represents the *degree*, or the number of neighboring nodes, and n represents the total number of nodes in the mesh. $D = 4$ if the ith node is an internal node; $D = 3$ if the ith node is a boundary node. As the mesh becomes topologically similar to a structured grid this topological irregularity approaches 0, but vanishes only when the mesh is perfectly structured, a rare situation. Otherwise, it has a positive value that measures how much the mesh topologically differs from a perfectly regular structured grid.

For geometric irregularity we define the measure, ε_g, that is the ratio of the radius of the *minimum inscribed circle* [2] to the radius of the *maximum circumcircle* [3]. Geometric irregularity ε_g is thus calculated as

$$\varepsilon_g = \frac{1}{m}\sum_{i=0}^{m} g_i, \tag{8}$$

where $g_i = \left(\frac{1}{\sqrt{2}} - \frac{r_i}{R_i}\right)$, m is the number of quadrilaterals, r_i the minimum inscribed circle radius of the ith quadrilateral, and R_i the maximum circumcircle radius of the ith quadrilateral. Since the ratio r_i/R_i takes its maximum value $\frac{1}{\sqrt{2}}$ for a perfect square element, an ideal element, the smaller the value of ε_g, the more geometrically regular the quadrilateral mesh.

Five meshing results are shown in Figures 6, 7, 8, 9, and 10, and some statistics are shown in Table 1 and Figure 11 for the first four meshes. Table 1 summarizes the mesh statistics including: (1) the numbers of mesh nodes and elements; (2) CPU times for the initial meshing and CPU times for 100 iterations of dynamic simulation; and (3) mesh irregularity measure. All the CPU times are measured on a SGI O2 workstation with a R5000/180MHz CPU.

In generating Mesh 1 and Mesh 2 shown in Figures 6 and 7 respectively the vector fields that represent desired mesh directions are automatically generated from the boundary geometry. The final quadrilateral meshes are thus aligned along the boundary directions. The node spacing functions are uniform so that the domain is packed with squares of a uniform size, yielding uniform quadrilateral meshes.

In Mesh 3 shown in Figure 8 a non-uniform node spacing function is specified to generate a graded quadrilateral mesh. Note that the sizes of the packed square cells in Figure 8(c) are adjusted based on the node spacing function shown in Figure 8(b), yielding the well-shaped, graded quadrilateral mesh shown in Figure 8(f).

Mesh 4 and Mesh 5 shown in Figures 9 and 10 respectively are meshes of the same geometric domain. The two meshes are created, however, using different mesh direction vector fields. In Mesh 4 the mesh directions are specified so that they align along the domain boundary, and in Mesh 5 the mesh directions are uniform. Note that both meshes are well aligned along the specified mesh directions.

[2] The minimum inscribed circle is the smallest circle tangent to at least three edges of a quadrilateral element.

[3] The maximum circumcircle is the largest circle that goes through at least three vertices of a quadrilateral element.

Table 1: Mesh statistics.

Mesh	*Number of elements in all-quad mesh*	*Number of nodes, quad, and tri in quad-dominant mesh*	*CPU time initial mesh*	*CPU time 100 iterations**	*Mesh irregularity after convergence*	
Mesh 1	743	222, 167, 25	0.787 sec.	4.366 sec.	$\varepsilon_t = 0.18468$	$\varepsilon_g = 0.09933$
Mesh 2	1748	488, 401, 48	1.259 sec.	11.631 sec.	$\varepsilon_t = 0.12705$	$\varepsilon_g = 0.08428$
Mesh 3	617	166, 134, 27	0.660 sec.	2.480 sec.	$\varepsilon_t = 0.19880$	$\varepsilon_g = 0.08841$
Mesh 4	804	239, 180, 28	0.794 sec.	4.432 sec.	$\varepsilon_t = 0.13389$	$\varepsilon_g = 0.09867$

* Approximately 50 to 100 iterations are sufficient to generate a reasonably good mesh.

7 Conclusion

We have presented a new physically-based method for well-shaped, graded quadrilateral meshing of a 2D region. Our central idea was to pack squares closely in a domain to mimic a pattern of Voronoi polygons corresponding to a well-shaped, graded quadrilateral mesh. To obtain a close packing of squares, we proposed a physically-based approach using a proximity-based potential field.

The most powerful feature of this new approach is that we can specify arbitrary mesh directionality as a vector field defined over a domain as well as arbitrary node spacing as a scalar field. The mesh directionality can be either: (1) manually specified by the user; (2) automatically generated from domain boundary directions; or (3) automatically generated from a previous analysis result.

One advantage of our physically-based packing of square cells is that the quadrilateral elements generated are so well-shaped that no further smoothing or *topological cleanup* [26, 11] is necessary. Most previous approaches require smoothing or topological cleanup to improve the mesh quality, and these operations often destroy the node spacing or mesh directionality in the original mesh.

Another advantage of using dynamic simulation is that it makes adaptive remeshing efficient. Adaptive remeshing is necessary in some FEM analyses in which the domain boundary, node spacing, and/or mesh directionality change over time. Fluid dynamics simulations with moving boundaries and large deformation structural analyses fall into this category. In these analyses it is possible that a mesh becomes too distorted over time to yield a valid computational result, and the mesh has to be updated. Our method can handle this remeshing efficiently because it updates the mesh easily by running a few iterations of dynamic simulation without constructing the new mesh from scratch.

A potential limitation of the proposed method is its relatively expensive computational cost compared to some of the purely geometric approaches. The method, therefore, can best be utilized in applications that benefit from regular element shapes, well-controlled element sizes, and well-controlled mesh directionality. Such applications include FEM analysis of thermal/fluid dynamics simulation, automobile crash simulation, and sheet metal forming simulation.

Finally, like the original bubble mesh method for triangular and tetrahedral meshing, the proposed method can be naturally extended to quadrilateral meshing of a parametric surface and hexahedral meshing of a solid by packing cubical cells instead of square cells.

References

[1] T.D. Blacker and M.B. Stephenson. Paving: A new approach to automated quadrilateral mesh generation. *Intl. J. Numer. Meth. Eng.*, 32:811–847, 1991.

[2] Frank J. Bossen and Paul S. Heckbert. A pliant method for anisotropic mesh generation. In *Proc. of 5th Intl. Meshing Roundtable*, pages 63–74, 1996.

[3] J.C. Cavendish. Automatic triangulation of arbitrary planar domains for the finite element method. *Intl. J. Numer. Meth. Eng.*, 8:679–696, 1974.

[4] J.C. Cavendish, D.A. Field, and W.H. Frey. An approach to automatic three-dimensional finite element mesh generation. *Intl. J. Numer. Meth. Eng.*, 21:329–347, 1985.

[5] M.S. Shephard et al. Trends in automatic three-dimensional mesh generation. *Computers and Structures*, 30(1/2):421–429, 1988.

[6] W.H. Frey. Selective refinement: A new strategy for automatic node placement in graded triangular meshes. *Intl. J. Numer. Meth. Eng.*, 24:2183–2200, 1987.

[7] J. Fukuda and J. Suhara. Automatic mesh generation for finite element analysis. In J.J. Oden, editor, *Advances in Computational Methods in Structural Mechanics and Design*, Huntsville, Alabama, U.S.A., 1972. UAH Press.

[8] E.A. Heighway. A mesh generator for automatically subdividing irregular polygons into quadrilaterals. *IEEE Transactions on Magnetics*, Mag-19, 1983.

[9] K. Ho-Le. Finite element mesh generation method: A review and classification. *Computer-Aided Design*, 20(1):27–38, 1988.

[10] B.P. Johnston, J.M. Sullivan Jr., and A. Kwasnik. Automatic conversion of triangular finite element meshes to quadrilateral elements. *Intl. J. Numer. Meth. Eng.*, 31, 1991.

[11] Paul Kinney. Clean up: Improving quadrilateral finite element meshes. *Proc. of 6th Intl. Meshing Roundtable*, pages 449–461, 1997.

[12] Y.T. Lee. *Automatic Finite Element Mesh Generation Based on Constructive Solid Geometry.* PhD thesis, University of Leeds, Leeds, England, 1983.

[13] Y.T. Lee. Automatic finite-element mesh generation. *ACM Transactions on Graphics*, 3(4):287–311, 1984.

[14] Randy R. Lober, Timothy J. Tautges, and Rich A. Cairncross. The parallelization of an advancing-front, all-quadrilateral meshing algorithm for adaptive analysis. *Proc. of 4th Intl. Meshing Roundtable*, pages 59–70, 1995.

[15] Anish Malanthara and Walter Gerstle. Comparative study of unstructured meshes made of triangles and quadrilaterals. *Proc. of 6th Intl. Meshing Roundtable*, pages 437–447, 1997.

[16] A.O. Mascardini, B.A. Lewis, and M. Cross. Agthom - automatic generation of triangular and higher order meshes. *Intl. J. Numer. Meth. Eng.*, 19:1331–1353, 1983.

[17] Matthew Rees. Combining quadrilateral and triangular meshing using the advancing front approach. *Proc. of 6th Intl. Meshing Roundtable*, pages 337–348, 1997.

[18] J. Ruppert. *Results on Triangulation and High Quality Mesh Generation.* PhD thesis, Univeristy of California at Berkeley, CA, U.S.A., 1992.

[19] N. Sapidis and R. Perucchio. Advanced techniques for automatic finite element meshing from solid models. *Computer-Aided Design*, 8(4):248–253, 1989.

[20] M.S. Shephard and M.K. Georges. Automatic three-dimensional mesh generation by the finite octree technique. *Intl. J. Numer. Meth. Eng.*, 32:709–749, 1991.

[21] Kenji Shimada. *Physically-Based Mesh Generation: Automated Triangulation of Surfaces and Volumes via Bubble Packing.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, U.S.A., 1993.

[22] Kenji Shimada and David C. Gossard. Bubble mesh: Automated triangular meshing of non-manifold geometry by sphere packing. In *Third Symp. on Solid Modeling and Appls.*, pages 409–419, 1995.

[23] Kenji Shimada and David C. Gossard. Automatic triangular mesh generation of trimmed parametric surfaces for finite element analysis. *Computer Aided Geometric Design*, 15/3:199–222, 1998.

[24] Kenji Shimada and Takayuki Itoh. Automated conversion of 2d triangular meshes into quadrilateral meshes. In *Proc. of International Conference on Computational Engineering Science*, 1995.

[25] Kenji Shimada, Atsushi Yamada, and Takayuki Itoh. Anisotropic triangulation of parametric surfaces via close packing of ellipsoids. *Intl. J. on Computational Geometry and Applications*, 1997. submitted.

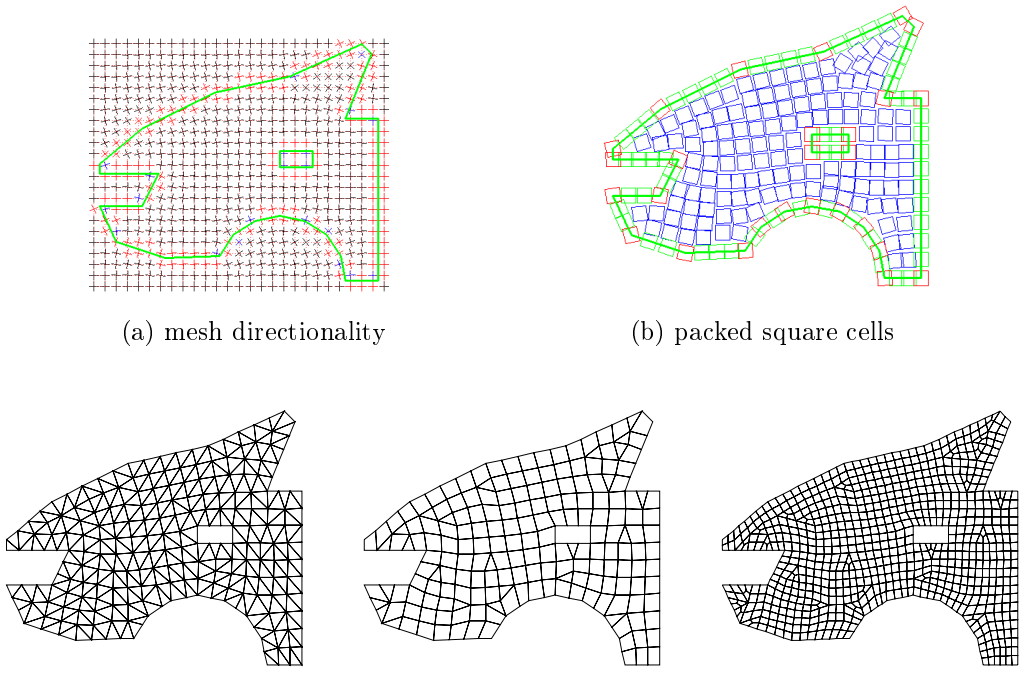
[26] Mathew L. Staten and Scott A. Canann. Post refinement element shape improvement for quadrilateral meshes. *Trends in Unstructured Mesh Generation, ASME*, 220:9–16, 1997.

[27] W.C. Thacker. A brief review of techniques for generating irregular computational grids. *Intl. J. Numer. Meth. Eng.*, 15:1335–1341, 1980.

[28] David R. White and Paul Kinney. Redesign of the paving algorithm: Robustness enhancements through element by element meshing. *Proc. of 6th Intl. Meshing Roundtable*, pages 323–335, 1997.

[29] Atsushi Yamada, Kenji Shimada, and Takayuki Itoh. Energy-minimizing approach to meshing curved wire-frame models. In *Proc. of 5th Intl. Meshing Roundtable*, pages 179–191, 1996.

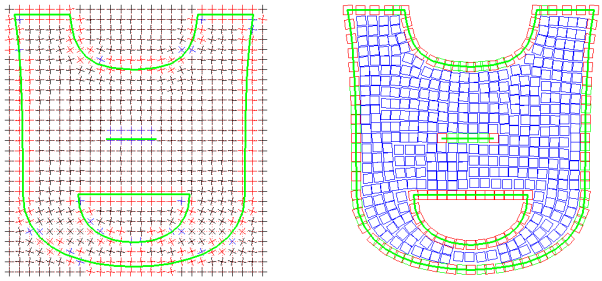
[30] M.A. Yerry and M.S. Shephard. A modified-quadtree approach to finite element mesh generation. *IEEE Computer Graphics and Applications*, 3:39–46, 1983.



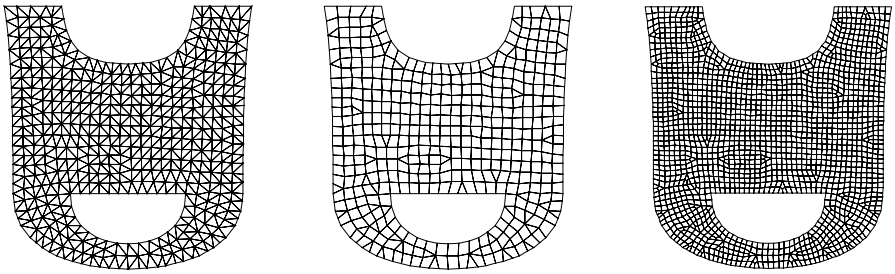
(a) mesh directionality (b) packed square cells

(c) triangular mesh (d) quad-dominant mesh (e) all-quad mesh

Figure 6: Mesh 1: uniform size, mesh directionality aligned along boundary

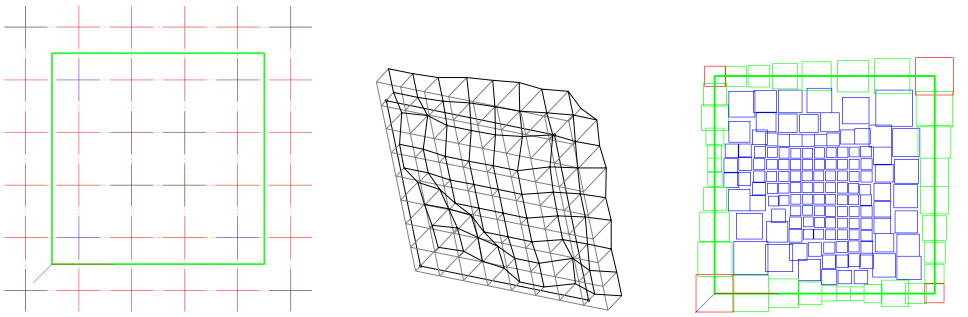


(a) mesh directionality (b) packed square cells

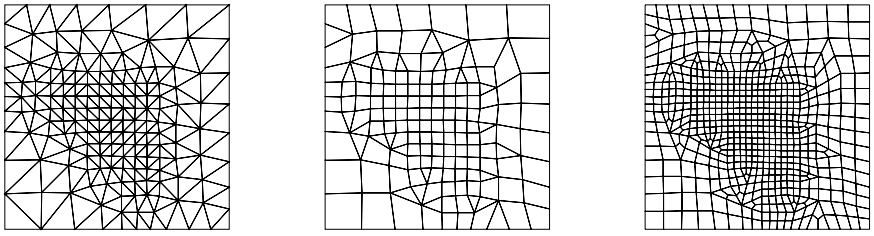


(c) triangular mesh (d) quad-dominant mesh (e) all-quad mesh

Figure 7: Mesh 2: uniform size, mesh directionality aligned along boundary

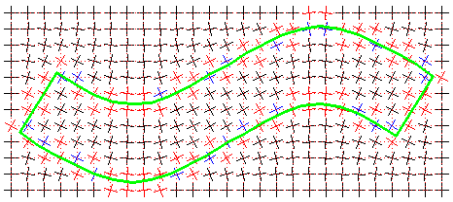


(a) mesh directionality (b) node spacing (c) packed square cells

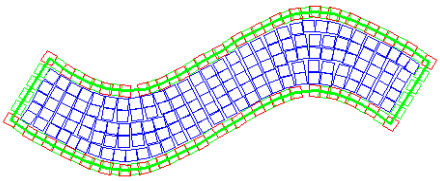


(d) triangular mesh (e) quad-dominant mesh (f) all-quad mesh

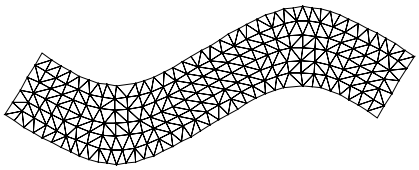
Figure 8: Mesh 3: graded size, uniform mesh directionality



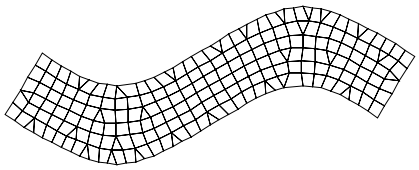
(a) mesh directionality



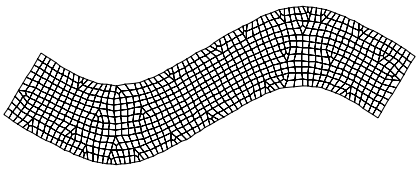
(b) packed square cells



(c) triangular mesh

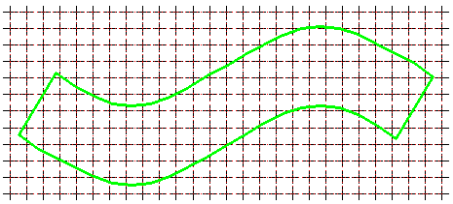


(d) quad-dominant mesh

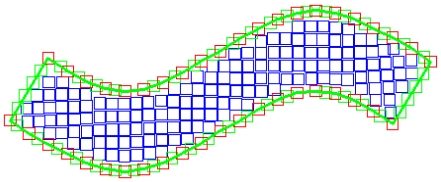


(e) all-quad mesh

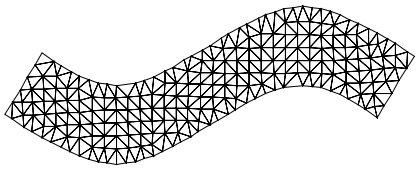
Figure 9: Mesh 4: uniform size, mesh directionality aligned along boundary



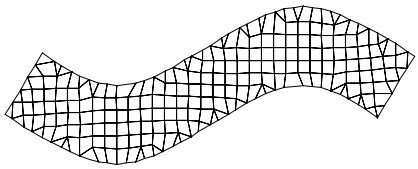
(a) mesh directionality



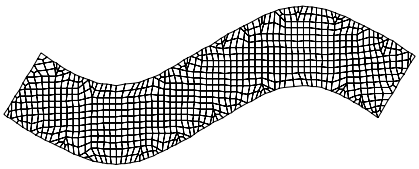
(b) packed square cells



(c) triangular mesh

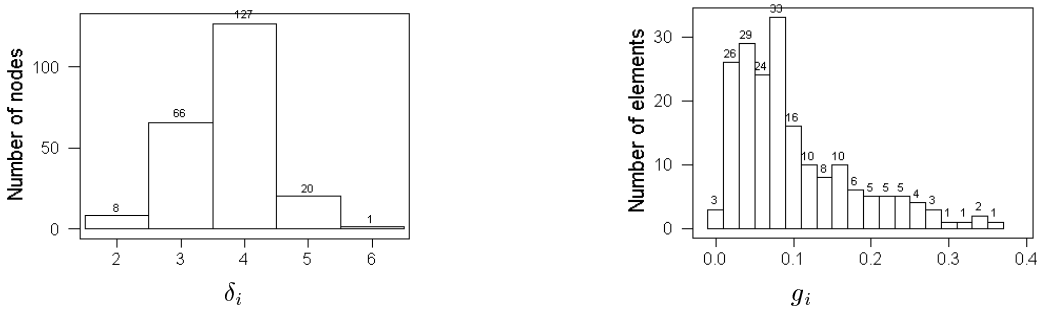


(d) quad-dominant mesh

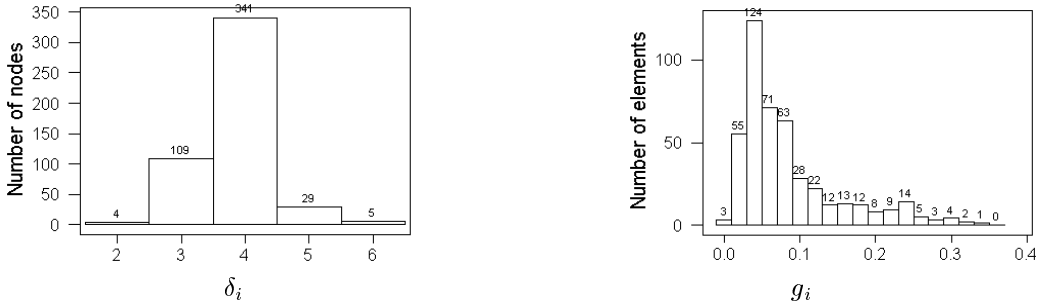


(e) all-quad mesh

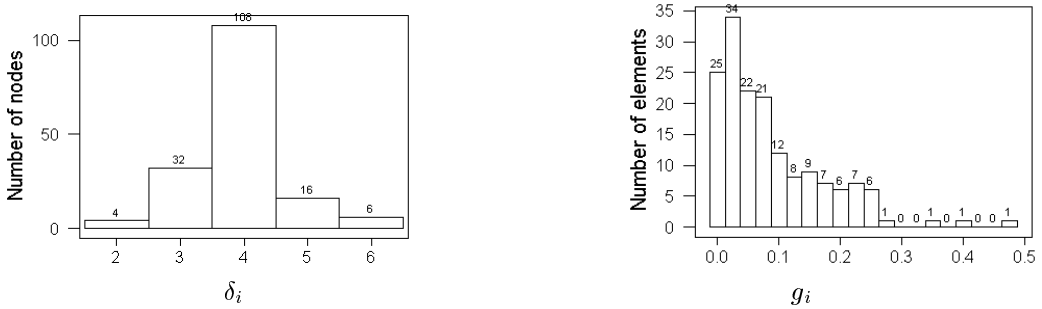
Figure 10: Mesh 5: uniform size, uniform mesh directionality



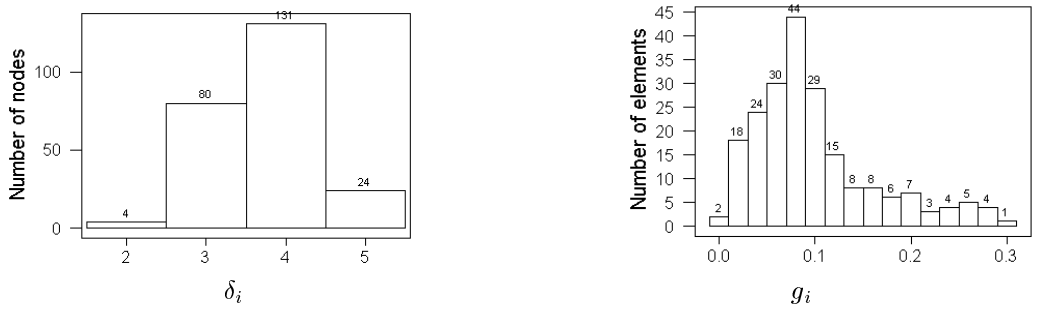
(a) Mesh 1 irregularity



(b) Mesh 2 irregularity



(c) Mesh 3 irregularity



(d) Mesh 4 irregularity

Figure 11: Topological irregularity and geometric irregularity