Engineering with Computers (2009) 25:207–219
DOI 10.1007/s00366-008-0119-9

ORIGINAL ARTICLE

Surface mesh regeneration considering curvatures

A. C. O. Miranda · L. F. Martha · P. A. Wawrzynek · A. R. Ingraffea

Received: 12 February 2008 / Accepted: 7 November 2008 / Published online: 5 December 2008

Abstract This work describes an automatic algorithm for unstructured mesh regeneration on arbitrarily shaped three-dimensional surfaces. The arbitrary surface may be: a triangulated mesh, a set of points, or an analytical surface (such as a collection of NURBS patches). To be generic, the algorithm works directly in Cartesian coordinates, as opposed to generating the mesh in parametric space, which might not be available in all the cases. In addition, the algorithm requires the implementation of three generic functions that abstractly represent the supporting surface. The first, given a point location, returns the desired characteristic size of a triangular element at this position. The second method, given the current edge in the boundary-contraction algorithm, locates the ideal apex point that forms a triangle with this edge. And the third method, given a point in space and a projection direction, returns the closest point on the geometrical supporting surface. This work also describes the implementation of these three methods to re-mesh an existing triangulated mesh that might present regions of high curvature. In this implementation, the only information about the surface geometry is a set of triangles. In order to test the efficiency of the proposed algorithm of surface mesh generation and implementation of the three abstract methods, results of performance and quality of generated triangular element examples are presented.

Keywords Mesh generation · Curvature surfaces · Background octree · Advancing-front technique

A. C. O. Miranda (✉) · L. F. Martha
Department of Civil Engineering and Computer Graphics Technology Group (Tecgraf),
Pontifical Catholic University of Rio de Janeiro,
Rua Marquês de São Vicente 225-Gávea,
Rio de Janeiro, RJ 22453-900, Brazil
e-mail: amiranda@tecgraf.puc-rio.br

P. A. Wawrzynek · A. R. Ingraffea
Cornell Fracture Group, Rhodes Hall, Ithaca, NY 14850, USA

1 Introduction

This paper describes an algorithm for generating triangle finite-element meshes on surfaces of arbitrary shape and curvatures. It is an extension of a previously proposed algorithm for generating unstructured meshes in three-dimensional (3D) [1, 2] and two-dimensional (2D) [3, 4] domains. Algorithms have been described in the literature previously that generate 3D surface meshes by re-meshing existing triangulations [5–7], or by using analytical surface descriptions for geometrical support [8–10]. The present algorithm generalizes the type of geometrical support that may be used by making use of three generic functions that abstractly represent the supporting surface. For example, the surface to be re-meshed may be a triangulated mesh, a set of points, or an analytical surface (such as a collection of NURBS patches). The present algorithm, as does its ancestors, incorporates well-known meshing procedures [11–17] but also introduces some original steps. It is essentially an advancing-front technique that takes special care to generate elements with the best possible shape. To enhance the quality of the mesh's element shape, an a posteriori local mesh improvement procedure is used. In a particular case, this paper also describes an implementation of the three generic methods for re-meshing existing surface triangulations and demonstrates the quality of the generated meshes.

The main objective of this implementation is shape-quality improvement of existing surface meshes. In these cases, surfaces are composed by a set of triangles topologically arranged. This is the only available geometric information and a parametric space is unknown. Differently than in a previous version that generates meshes in parametric space [2], the current algorithm works directly in Cartesian coordinates, which increases its generic characteristics. Of course, re-triangulating an existing triangulated surface with no knowledge on the original surface analytical description will introduce some geometric approximations. However, this is a very common application and works well if the existing triangulation does not present a course mesh in regions with high curvatures. In addition, special care must be taken in the implementation of the three generic functions that geometrically represent the surface to maintain the good time performance of the parametric space mesh generation.

The meshes generated by the present algorithm have similar characteristics to those produced by its ancestors [1–4], which are summarized as follows.

1. The algorithm produces well-shaped elements, avoiding elements with poor aspect ratio. While it does not guarantee bounds on aspect ratios of elements, care is taken at each step to produce good-quality meshes.
2. The mesh conforms to an existing discretization on the surface's boundary. This is important in the generation of finite-element meshes because it is usually desirable to have a mesh that conforms to the mesh generated on adjacent patches.
3. The algorithm presents a smooth transition between regions with elements of highly varying sizes. This is a desirable feature because a finite-element analysis requires high element density in regions with high gradients of response, while a low density may be used in other regions. It is not uncommon in problems with very high gradient producing features (e.g. cracks or shock waves) to have element sizes that differ by two, or in some cases three, orders of magnitude in element size. The element transition capability is also important for surfaces with regions of high curvatures because, in such locations, the algorithm will locally refine the mesh.

To facilitate a smooth transition between regions and regions of high curvatures, the present algorithm employs an octree data structure. However, unlike some authors, e.g. Rassineux [17], who use a quadtree/octree procedure to generate internal nodes prior to element generation, here it is used to provide local policies used to define the discretization of the surface mesh and to define the sizes of triangular elements to be generated during the advancing-front procedure. The authors feel that this approach tends to provide better control over the quality of the generated mesh and to decrease the number of heuristic, clean-up procedures.

The remainder of the paper is organized as follows. Initially, the paper describes the algorithm for generating unstructured triangulations for arbitrarily shaped 3D surfaces using the generic functions that abstractly represent the geometry of the supporting surface. After that, the paper describes the implementation of the three generic functions to re-mesh existent triangulated meshes. Four examples of triangle surface meshes are presented in order to obtain results of performance and quality of generated triangular elements. The final section comments on the results.

2 Description of the algorithm

Figure 1 shows the process overview of the proposed mesh generator. The input data for the present algorithm is a polygonal description of the boundary of the surface patch to be meshed and a 3D supporting surface that is abstractly represented by three generic methods defined as follow:

First method: Given a point location, the method returns the desired characteristic size of an ideal equilateral triangular element at this position. The length of the triangle's side is considered as its characteristic size.
Second method: Given the current base edge in the boundary-contraction algorithm, the method locates the ideal apex point that forms a new triangle. The method has two input arguments: the height of the candidate equilateral triangle and a unit vector in the edge perpendicular direction. This unit vector is a "surface intersection direction" that defines a plane perpendicular to the base edge at its mid point. These arguments are used to determine an optimal triangle apex point location.
Third method: Given a point in space, the method returns the closest point on the geometrical supporting surface. The method also receives a surface projection vector as additional information. This method is used only in the final stage of the surface mesh generation for local mesh improvement.

The boundary information is given by a list of nodes defined by their 3D coordinates and their normal vectors on the surface and a list of boundary segments (or edges) defined by their node connectivity. The input boundary must be defined a priori. The definition of these points is not part of the proposed algorithm and, for better results, the boundary segment sizes should be consistent with local surface curvatures (i.e., the boundary should be relatively more refined in regions of relatively high curvature). The

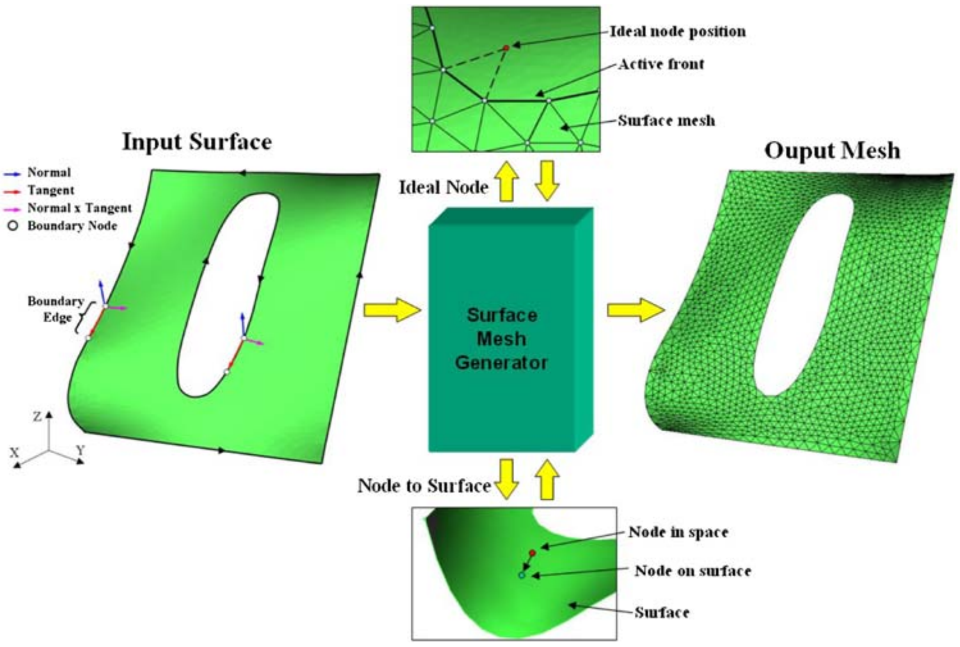


Fig. 1 Process overview of the proposed algorithm

given boundary edges form the initial front that advances as the algorithm progresses. At each step of this meshing procedure, a new triangle is generated for each base edge of the front. The front advances replacing the base edge with new triangle edges. Consequently, the domain region is contracted, possibly into several regions. The process stops when all contracted regions result in single triangles. As the front advances, the generic methods are called to obtain the size of new elements and the position of an ideal node. This is illustrated in Fig. 2 and described in more detail below. After this process, the mesh is improved using smoothing methods.

An R-Tree [18] structure is used in the advancing-front process to improve the efficiency of adjacency queries (elements adjacent to nodes) and for geometric checks (edge intersections and points inside triangle). R-trees are tree data structures that are similar to B-trees [19], but are used for spatial access methods, i.e., for indexing multi-dimensional information such as 3D coordinates. In this work, the 3D bounding box (minimum bounding regular hexahedron) of new triangular elements and edges is used for indexing the auxiliary R-trees in the advancing-front process.

2.1 Advancing-front procedure

The advancing-front technique starts with a boundary that bounds a region to be filled with a triangulation. Triangular elements are “extracted” or “pared” from the region one at a time. As each element is extracted, the boundary is updated and the process is repeated. The procedure terminates when the entire region is meshed. Therefore, the boundary of the region to be meshed is formed by edges of the triangles created in the contraction process. These edges are referred to as boundary edges.

In this algorithm, as in its ancestors [1–4], the advancing-front process is divided into two phases to ensure the generation of valid triangulations. In the first phase, a geometry-based element generation is pursued to generate elements of optimal shapes. After this ideal phase is exhausted and no more optimal elements can be generated, a topology-based element generation takes place, creating

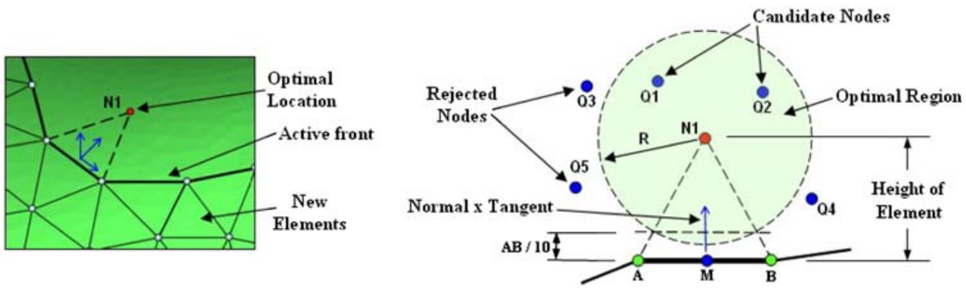


Fig. 2 Determination of a new element

valid, but not necessarily well shaped, elements in the remaining region. The required steps for the advancing-front procedure are as follows.

2.1.1 Front initialization based on given boundary edges

The process starts with the creation of the initial advancing front, which is formed by the given boundary segments. The current boundary edges are stored in two separate doubly linked lists. The first is a list of active edges, which includes all boundary edges that have not been used in an attempt to generate valid triangles. The other is a list of rejected edges, which have failed in the generation of elements for the current phase. Initially, all segments of the given boundary refinement are stored in the first list.

The initial list of active edges on the boundary is sorted by the length of the edges. This has been recommended by other authors [16] to prevent large elements from penetrating regions with small-length edges. This criterion is only used in the initial boundary edge list.

It was also found convenient for some steps in the algorithm to have an additional data structure storing a list of adjacent boundary edges for each node on the current advancing front. This data structure is initialized for all nodes of the given boundary, and is updated as the boundary-contraction procedure progresses.

The data structure also stores normal vectors for all triangular elements, nodes, and edges. These normal vectors are used to improve the efficiency of edge intersection geometric checks, as described in next section. The normal vectors are computed in the following way. The normal vector of a node is computed as the average of the normal vectors of adjacent elements. The normal vector of an edge is the average of normal vector of its end nodes.

2.1.2 Front contraction (geometry-based element generation)

Ideally, the entire mesh will be generated in the geometry-based phase. This depends on the geometry and topology of the given boundary model and, as observed, is strongly related to the segment-size disparity of the given boundary refinement. In this phase, for each base edge on the advancing front, the following is performed (see Fig. 2):

- The optimal location **N1** for the vertex of an equilateral triangle to be formed is determined employing the *First* and *Second methods*. Using the base edge middle point (**M**) as input, *First method* returns the target triangle characteristic size (the length of the equilateral triangle side) at this point location. With this size, the height of the candidate triangle is obtained. Then, using the **Normal** $\times$ **Tangent** vector at the middle point **M** as the required unit vector (surface intersection direction) in the edge perpendicular direction and the triangle height, *Second method* returns the desired **N1** location on the support surface.
- The optimal point defines an optimal region in which the vertex of the triangle to be should be located. This region is a sector of a sphere whose center is the optimal point and whose radius is proportional to the height of element. In the current implementation a proportionality constant of 0.85 was adopted. This sphere defines an upper bound for the distance between the target vertex of the triangle and the middle point of the base edge. A lower bound is defined to ensure that the generated triangle will have area greater than the smallest acceptable area. In the current implementation, this lower bound is defined by a triangle with height equal to 1/10 of the base edge. The optimal region is used for two reasons: first, to ensure shape quality of the elements to be generated; and, second, to ensure that new internal nodes will be created only when it is strictly necessary and always in good positions.
- If no existing node is inside the optimal region, a new node is inserted at the optimal location **N1** and an element is generated using this node. If only one node exists in the region, this node is used to generate the element. If more than one node is found in the region, they are ranked according to the included angle with respect to the base edge. The node with the maximum included angle is used to generate the element. A heap list is used to efficiently implement this priority queue.
- Additional geometric checks are performed to ensure that the edges of the new triangular element do not intersect any existing edge of the advancing front and that the new triangle apex does not lie inside any other existing triangle. In both cases, the new element is rejected. These checks are not trivial in 3D space. However, they are performed in a local 2D system on the plane of the new triangular element, avoiding complex geometry checks in 3D space. As mentioned, the algorithm uses an auxiliary R-Tree structures to improve queries for adjacent edges and elements around a candidate triangular element. The 3D bounding box of this triangle is used as an index to the R-Tree. The new edges and adjacent elements edges are transformed to the local 2D system on the plane of the new triangular element and all checks are made in this local system.
- Once a valid triangle is generated for the current base edge, the list of active edges is updated. This is done through the following steps: first, the base edge is removed from the list; then, for the other edges of the element, the edge is either deleted, if it coincides with an edge already in the list, or inserted in the list as a new one.

- Due to restrictions on the allowable range of element shape metrics, there are situations in which the algorithm fails in forming a valid triangle for the current boundary's base edge. In these cases, the current base edge is removed from the list of active edges and is stored in the separate list of rejected edges. It might happen that an edge is subsequently removed from this latter list if it is used as part of a valid triangle for an adjacent base edge.
- When there are no more edges in the list of active edges, the algorithm tries to generate elements using the edges that were previously rejected. It might be the case that base edges that previously failed may now work because the front has changed with the addition of elements. The geometry-based element-generation phase ends when either there are no edges left in the boundary-contraction lists (in which case an optimal mesh was generated) or when a rejected edge fails for a second time.

2.1.3 *Front contraction (topology-based element generation)*

The objective of this phase of the algorithm is to force the generation of valid triangles, even if a measure of the quality of the shape of the element does not fall within the allowable range used in the previous phase. The topology-based element-generation phase starts when a boundary edge fails twice in trying to generate an optimal element. The list of rejected edges of the previous phase is transformed into a list of active edges and, similarly to the geometry-based phase, a list of rejected edges is created for edges that eventually fail in generating valid triangles.

In the topology-based element-generation phase, any node close to the current base edge is selected and stored in a priority queue of candidate nodes. The node that has the maximum included angle with respect to the base edge is chosen for the generation of the new triangle. If the edges of this triangle do not intercept any other edge of the current advancing front, the element is created and the boundary is contracted accordingly. The topology-based phase ends when the lists of active and rejected edges are empty. This phase always generates a valid, even though non-optimal, mesh.

2.2 Local mesh improvement

A smoothing technique is used to improve mesh quality by relocating nodes within a patch. A general formulation for this technique is given by (1), which is a generic form of a weighted Laplacian function [20]:

$$X_0^{n+1} = X_0^n + \phi \frac{\sum_{i=1}^{m} w_{i0}(X_i^n - X_0^n)}{\sum_{i=1}^{m} w_{i0}}. \tag{1}$$

In this equation, m is the number of nodes connected to node O, X_0^{n+1} is the position of node O at smoothing iteration $n + 1$, w_{i0} is the weighted function between nodes i and O, and ϕ is a relaxation parameter which is normally set in the interval (0, 1]. In this work $\phi = w_{i0} = 1.0$, which means that the right side of (1) is reduced to an average of the adjacent points. The smoothing procedure is repeated twice for all internal nodes.

In general, the surface mesh smoothing (1) will move node O to a position off of the surface. The *Third method* is employed after each smoothing procedure as a "pull back" operation, moving the target node back to the geometric supporting surface.

In theory, Laplace smoothing and the "pull back" procedure can cause mesh "folding". A check is made and a node is not moved if doing so will cause an invalid mesh. In practice, there has been no need to enforce this restriction for any of the test cases.

3 Application: re-meshing

To demonstrate the performance and the efficiency of the proposed surface meshing algorithm, this section describes an implementation of the three generic methods for re-meshing existing surface triangulations. This application is very useful when an existing surface mesh needs to be refined, coarsened, improved or changed to conform to a pre-existing boundary discretization. For example, Fig. 3 (left) presents a mesh with triangular elements and four holes. Mainly, in the surface domain, the elements have similar shape and size. However, the boundary mesh presents many poorly-shaped elements (red arrows in right of Fig. 3). In the context of finite-element analysis, these elements are not desirable because they can cause inaccurate results in an important region of the domain.

It is important to mention that the three generic methods described in this section are considering a surface composed solely of a set of triangles. In this implementation, it is considered that there is no information about the original surface on which the existing triangulation was built. Different implementation strategies than the ones described in this section should be adopted for surfaces composed by a set of points or surfaces with an analytical description (with or without a parametric space).

This application requires a minimum set of information about the supporting mesh. The existing triangulation must have one continuous domain, with any number of boundary loops (holes). The nodal incidence of adjacent

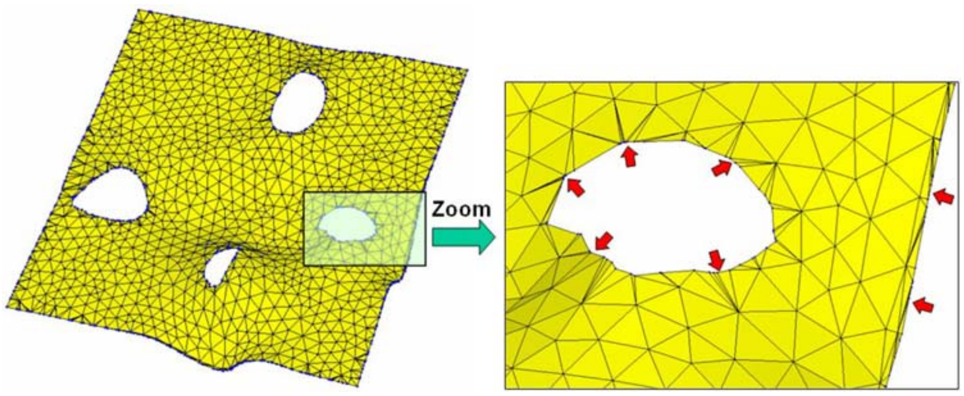


Fig. 3 Example of a mesh with badly-shape elements (*red arrows*)

triangular elements must have a consistent order. This nodal incidence (e.g., in counter-clockwise order) defines a normal direction of triangular elements. The given normal directions of the input boundary nodes, as shown in Fig. 1, must be consistent with the triangles normal directions.

3.1 First generic method implementation

The first generic method of the proposed surface mesh generation obtains the desired characteristic size of a triangular element given the position of a point on the surface. It is important that the algorithm presents a smooth transition between regions with elements of highly varying sizes. Therefore, an auxiliary background data structure is used to store the distribution of characteristic triangle sizes in space.

Many background structures are published in the literature [21–29] in the context of mesh generation. In the present work, as well as in its ancestors [1–4], an octree structure is used. An octree is a tree data structure based on a cell with eight children. Each cell of an octree represents a cube in physical space. Each child represents one octant of its parent. Figure 4 shows an example in which the tree has been refined twice. First, the root cell is subdivided into eight cells, each representing an octant of the root's domain. Then, one of root's children is recursively subdivided into eight cells.

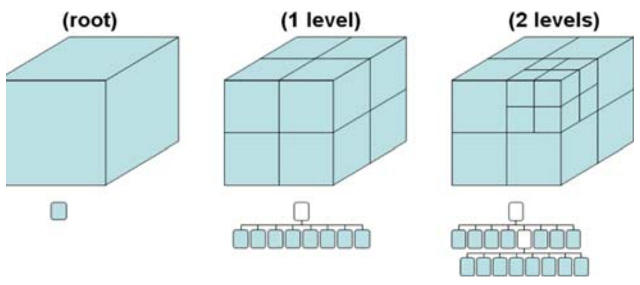


Fig. 4 An octree with two levels of refinement

In the current application the size of the leaf cells in a region of a domain are used as the size of the desired characteristic element size in this region. The creation of the background octree from an input surface follows five steps (see Fig. 5):

1. *Octree initialization based on given boundary edges*: Each segment of the input boundary data is used to determine the local subdivision depth of the octree as shown in Fig. 5a.
2. *Refinement to force maximum cell size*—Fig. 5b: The octree is refined to guarantee that no cell in its interior is larger than the largest cell at the boundary.
3. *Refinement to provide minimum size disparity for adjacent cells*—Fig. 5c: This additional refinement forces only one level of tree depth between neighboring cells and provides a natural transition between regions of different degrees of mesh refinement.
4. *Refinement to account for surface curvatures*—Fig. 5d: The center and the bounding box of each element of the existing supporting mesh are determined and stored, in an R-tree structure. For each element (element A), the algorithm finds the cell in the octree that contains its center. A second element (element B) is selected from the neighboring elements as to maximizes the angle between normal element vectors $\mathbf{N_A}$ and $\mathbf{N_B}$, or similarly, that minimizes the cosine angle between $\mathbf{N_A}$ and $\mathbf{N_B}$:

$$\cos\theta = \frac{N_A \cdot N_B}{\|N_A\|\|N_B\|} \tag{2}$$

If $\cos\theta$ is less than a minimum value, $\cos_{\min}\theta$, then a new cell size, H_{new}, is obtained:

$$H_{\text{new}} = \frac{d \cdot 0.5}{\sin(\theta \cdot 0.5)} \cdot \theta_{\max} \tag{3}$$

in which d is the distance between the centers of elements A and B, and $\theta_{\max}$ is the maximum angle allowed between normal vector of elements. This new size is used to locally refine both cells of elements A and B in the octree. This step is applied to all elements in the supporting mesh.

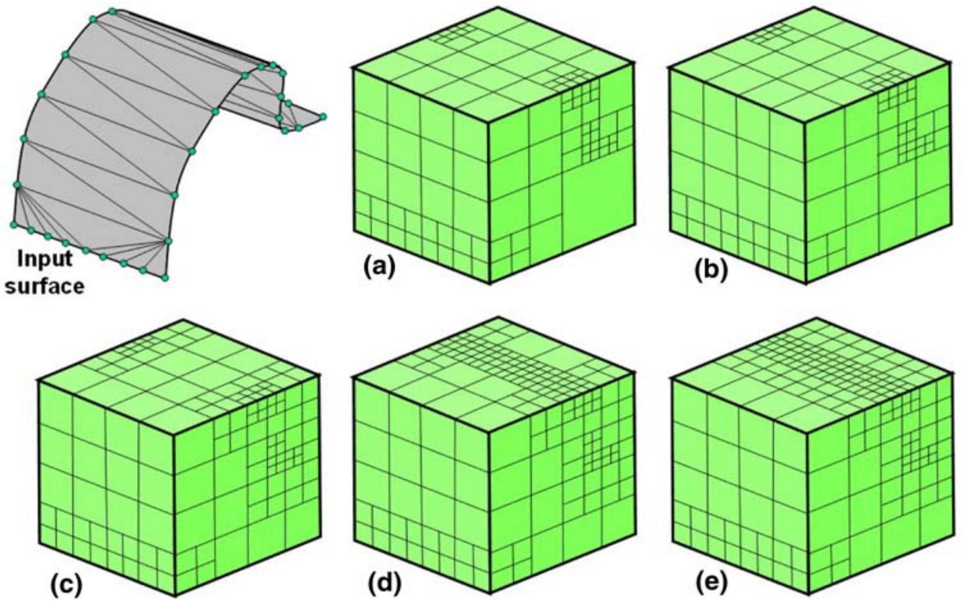


Fig. 5 Refinement of background octree

5. *Refinement to provide minimum size disparity for adjacent cells*—Fig. 5e: Step 3 is repeated for the updated octree.

3.2 Second generic method implementation

The second generic method returns the ideal apex point that forms a candidate equilateral triangle for a given current base edge in the boundary-contraction algorithm (Fig. 6a). In addition to the base edge, the height of the candidate triangle and a "surface intersection" direction are given, as shown in Fig. 6b.

Two different approaches to this task are described in the literature. Lohner [6] tries to find a host triangle, where the ideal node lies, using a neighbor-to-neighbor search. If this fails, octrees are employed. Finally, if this approach fails again, a brute force search over all the surface elements is performed. Carlos et al. [5] use a procedure that guarantees that the new triangle sides have exactly the desired length. This procedure considers the equation of a sphere, and finds the intersections between the sphere and the mesh. The approach used in this work is similar to this one, employing the circle equation in a 2D coordinate system.

Figure 6 illustrates how to obtain the target apex point in the present implementation. A local search based on the auxiliary R-Tree structure creates a list of all neighborhood elements around the middle base edge point. A plane of intersection, containing the given surface intersection direction, is created at this point. The intersection of the neighbor triangles with the plane of intersection forms a poly-line, as shown in Fig. 6b. A 2D local coordinate system is created on the plane of intersection and points of the intercepting poly-line are transformed to this coordinate system. The ideal node position is computed by intersecting a circle, whose center is the base edge middle point and radius is the height of the candidate triangle, with this poly-line. Finally, the point is transformed to the 3D coordinate system. Depending on the irregularity of the support mesh, it might happen that the intersection of a neighbor triangle

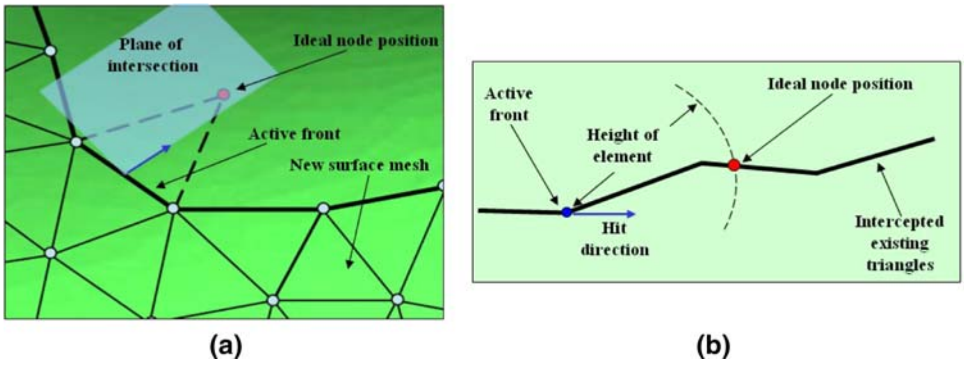


Fig. 6 Obtaining ideal node position in second generic method (top view on the left and lateral view on the right)

causes a self-crossing poly-line configuration. This is solved by considering the interception segment that best approximates the given surface intersection direction.

3.3 Third generic method implementation

The third generic method returns the point on the geometric supporting surface that is closest to a given point near to, but not necessarily on, the surface. During the nodal smoothing phase, points may be moved off the surface. This method is used to move the points back to the surface. As input, the method receives the current node location and its current normal vector. The implementation of this function is quite simple (see Fig. 7): First, a local search based on the auxiliary R-Tree structure creates a list of all neighborhood triangles of the geometric supporting surface around the given node. Second, the node is projected on each neighbor triangle plane, using the given projection direction [30]. Finally, the selected new node location is the one that lies inside one of the neighbor triangles.

Figure 8 presents an example with several phases of the advancing boundary-contraction process. Figure 8a shows the supporting mesh surface, which is a half-torus form. Figure 8f shows the final mesh. The next section presents more examples.

3.4 Example: performance mesh and quality

This section presents additional examples of finite-element meshes generated on 3D surfaces using the proposed algorithm. The main objectives of this section are to: (1) estimate the expected performance of the surface mesh generation algorithm, (2) analyze the visual aspect of generated triangle surface meshes, comparing the results with and without considering surface curvatures, and (3) assess the quality of generated triangular elements.

Figure 9 shows four examples of triangle surface meshes to be re-meshed with the present algorithm. The first one, Fig. 9a, is an open cylinder. The second, Fig. 9b, is a geological salt-dome surface. The third, Fig. 9c, is a

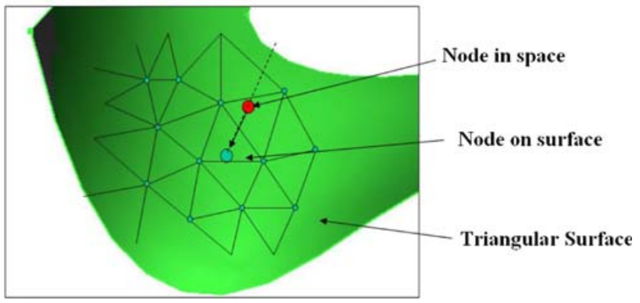


Fig. 7 Projecting a node in space on the supporting surface in the third generic method

geological surface with four internal loops (this is the same surface presented in Fig. 3 with poorly-shaped elements on the boundary). The final example, Fig. 9d, is a surface with many irregularities in its curvatures.

The observed processing times for generating refined meshes for the four example problems are reported in Table 1. These processing times were measured on a Pentium-2.66 GHz PC with 1 GB of RAM, under Windows XP operating system. In addition to the total time, Table 1 also shows partial processing times to build the background octree, to accomplish the advancing-front process, and to improve the quality of the mesh by smoothing. It is clear that most of the computational effort is spent in the advancing-front process.

The data reported in Table 1 contain results for a number of different levels of mesh refinement for each of the example problems. The degree of mesh refinement was controlled by changing the size of the input boundary segments. For these cases, the boundary curves were discretized with uniform segment sizes. No attempt was made to use relatively shorter boundary segments in regions of high curvature. Doing so would improve the quality of some of the generated meshes. However, uniform boundary segments were used to minimize the influence of the boundary discretization on the measured quality of the element shapes in the resulting meshes.

A formal analysis of the computational complexity of the proposed algorithm would be very difficult, especially considering input data-specific steps such as the generic methods. Nevertheless, a realistic estimate of the expected performance of the algorithm is desirable for its practical use. Figure 10 shows a plot of the elapsed processing time as a function of the number of elements generated. A computational complexity of *O(NlogN)* has been proposed in the literature for the best case performance for advancing-front meshing techniques [31]. The equation of a least-squares fit to times reported in Table 1 is $t = 2 \times 10^{-5} N^{1.18}$, where N is the number of generated elements. Using this fitting equation, one may infer that the algorithm's performance is close to *O(NlogN)*.

Figure 11 shows some of the refined meshes for these examples. In Fig. 11c, one may note that the uniform boundary discretization eliminated poorly-shaped elements, when compared to the same region shown in Fig. 3. Figure 11d demonstrates the importance of considering curvatures of the supporting surface in the algorithm. In this figure, the generated mesh was more refined in regions with high curvatures of the supporting mesh. The algorithm is able to handle this effect, providing good mesh transition between regions with distinct element characteristic sizes. For this example, Fig. 12 shows in detail the influence of the surface curvature: Fig. 12a shows a mesh generated without considering curvatures, and Fig. 12b shows a mesh

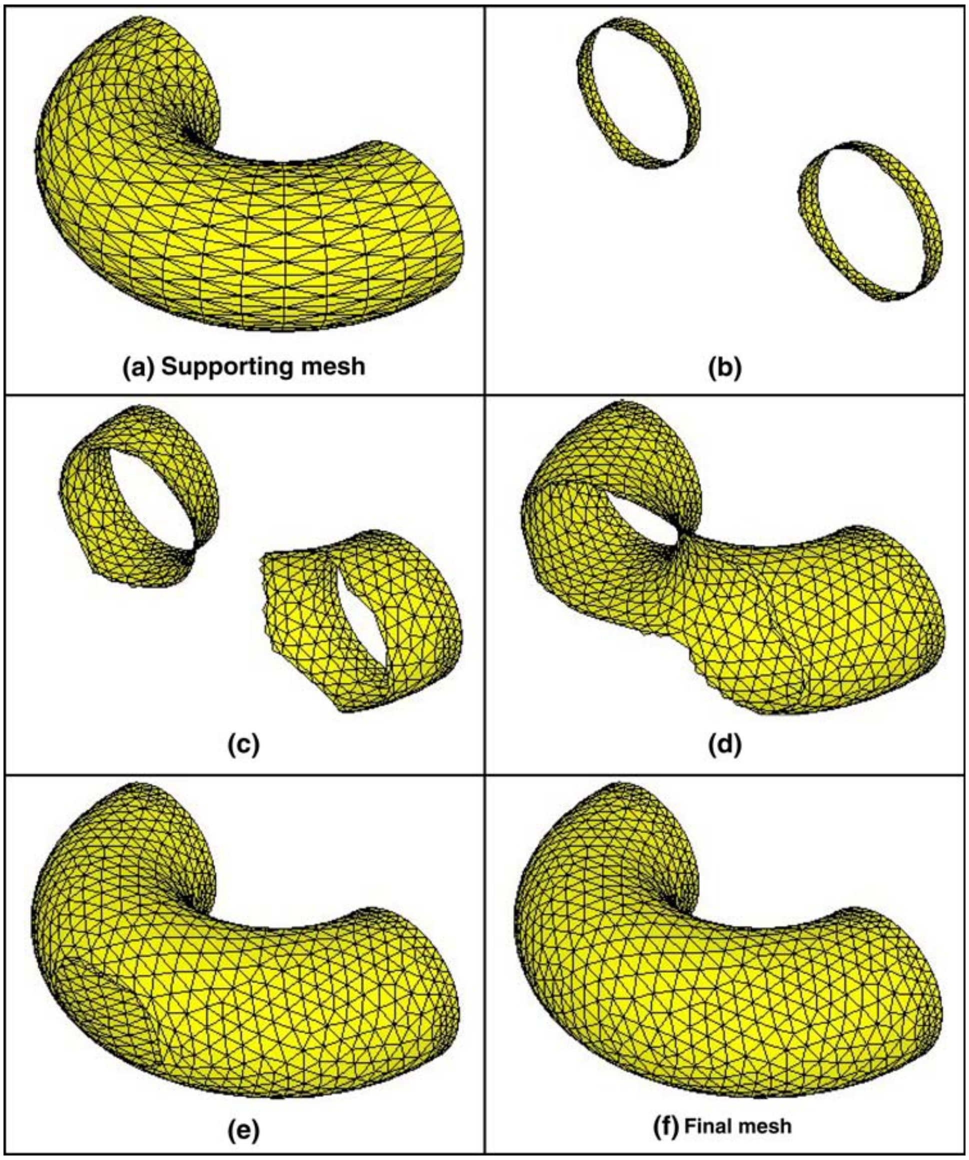


Fig. 8 Example of advancing boundary-contraction process for a half-torus surface

considering theses curvatures. Note that the surface boundary is refined uniformly. In this example, the refinement of boundary curves in locations of high curvature would improve even further the quality of the generated mesh.

In order to study the quality of the shapes of the generated elements a normalized ratio quality measure, γ/γ^*, is adopted [2]. In this measure γ is the ratio between the root mean square of the lengths (S_i) of a triangle's edges and the triangle's area, and γ^* is the corresponding value for an equilateral triangle:

$$\gamma = \frac{\frac{1}{3}\sum_{i=0}^{3} S_i^2}{\text{Area}}. \quad (4)$$

The γ/γ^* quality measure has a valid interval between 1.0 and infinity, with high quality elements, those close to an equilateral triangle, having values close to 1.0.

The quality of generated meshes is presented in the form of a histogram such as the one shown in Fig. 13. In this histogram, the horizontal axis corresponds to the γ/γ^* quality measure in intervals represented by triangular shapes that are shown below the histogram. The vertical axis corresponds to the percentage of elements in each interval of the quality measure. These results demonstrate that the proposed algorithm generates meshes with good quality for the great majority of elements. Note that the quality of meshes in all examples is below 1.5, an indication of very well-shaped elements.

4 Conclusion

This paper has described an algorithm for regenerating triangle finite-element meshes on surfaces of arbitrary

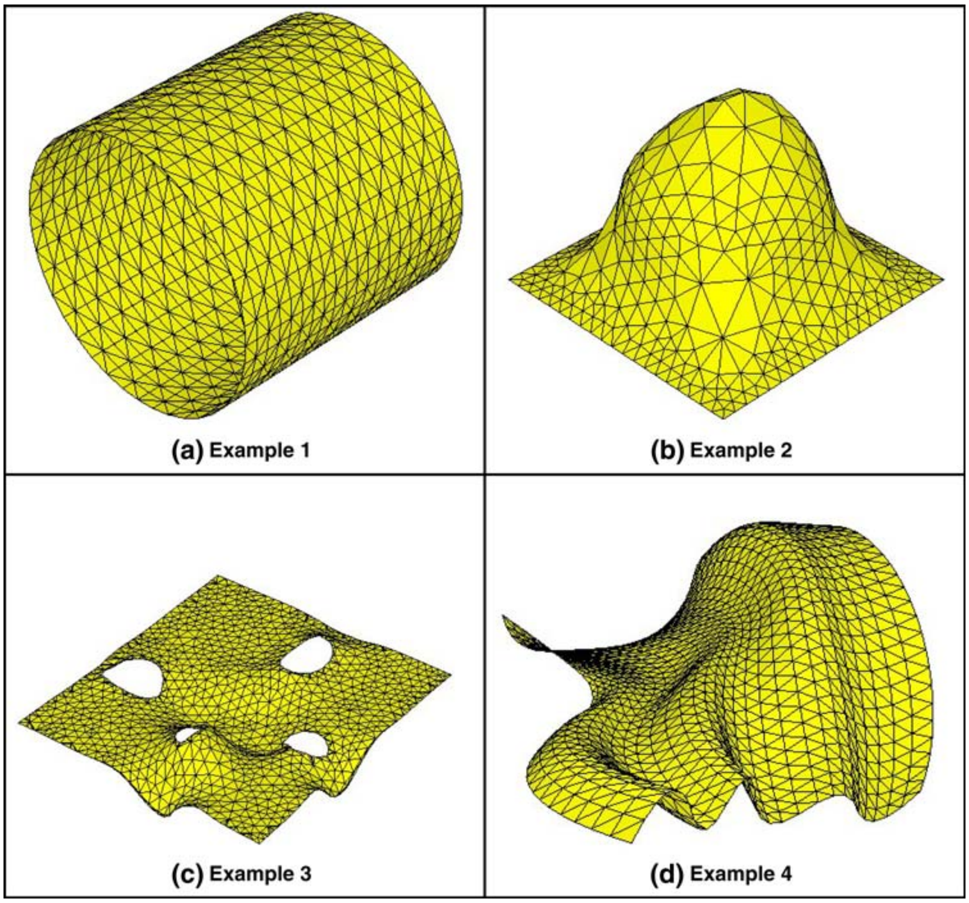


Fig. 9 Examples of surfaces with their initial triangle meshes

Table 1 Processing times for re-meshing example surfaces

Example	Number of elements	Background Octree		Boundary Contraction		Smoothing		Total
		(s)	(%)	(s)	(%)	(s)	(%)	(s)
Example 1-1	3,906	0.031	9	0.313	87	0.016	4	0.36
Example 1-2	15,872	0.094	6	1.391	91	0.047	3	1.532
Example 1-3	63,746	0.563	7	7.610	91	0.203	2	8.376
Example 1-4	79,872	0.579	6	8.406	91	0.235	3	9.22
Example 1-5	84,048	0.578	5	11.140	93	0.250	2	11.968
Example 2-1	2,454	0.015	7	0.188	93	0.000	0	0.203
Example 2-2	9,480	0.078	9	0.781	89	0.015	2	0.874
Example 2-3	42,791	0.578	12	4.172	87	0.031	1	4.781
Example 2-4	60,091	0.578	8	6.328	90	0.109	2	7.015
Example 2-5	72,715	0.609	5	10.687	94	0.125	1	11.421
Example 3-1	6,111	0.109	13	0.672	81	0.047	6	0.828
Example 3-2	24,136	0.578	15	3.188	82	0.125	3	3.891
Example 3-3	34,272	0.610	10	5.562	88	0.140	2	6.312
Example 3-4	97,039	4.500	16	23.093	82	0.547	2	28.14
Example 4-1	7,652	0.094	12	0.656	86	0.031	4	0.781
Example 4-2	29,343	0.578	15	3.078	82	0.110	3	3.766
Example 4-3	33,782	0.578	13	3.859	84	0.110	2	4.547
Example 4-4	42,359	0.609	9	5.734	85	0.157	2	6.500

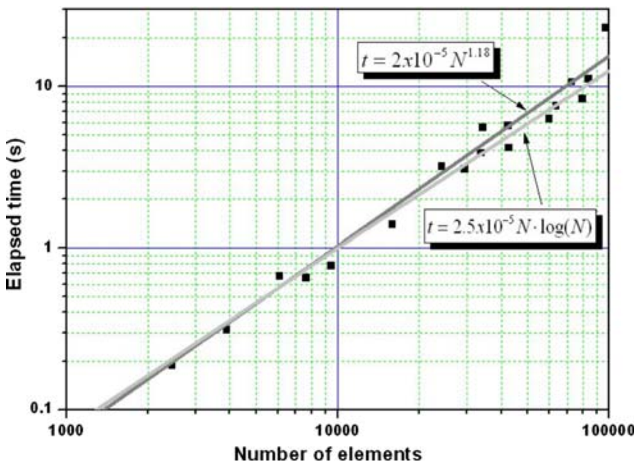


Fig. 10 Generation times for re-meshing the example surfaces

shape and with varying curvature. The algorithm incorporates aspects of well-known meshing procedures and includes some original steps. This algorithm is an extension of a previously proposed algorithm for generating unstructured meshes in 3D and 2D domains.

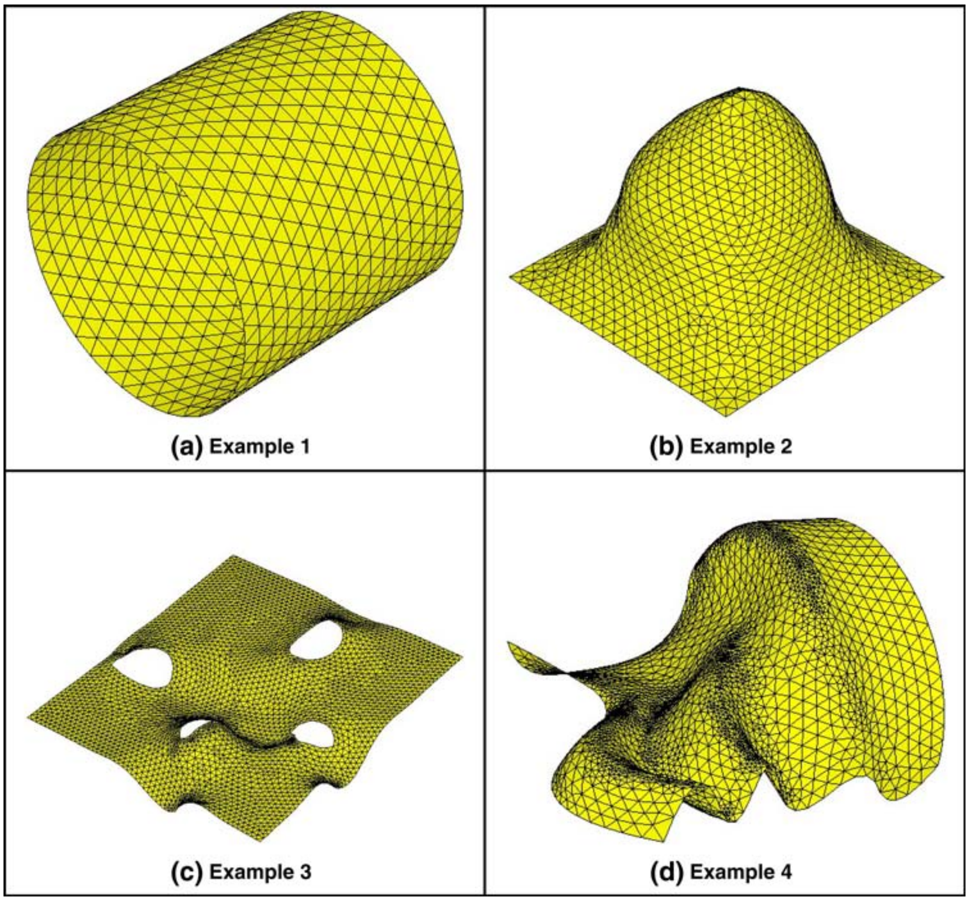
The input data for the present algorithm is a generic supporting surface and a polygonal description of the boundary on the surface patch to be meshed. The supporting surface is represented abstractly by three generic methods:

- The first, given a point location, returns the desired characteristic size of a triangular element at this position.
- The second, given the current edge in the boundary-contraction algorithm, locates the ideal apex point that forms a triangle with this edge.
- The third, given a point in space and a projection direction, returns the closest point on the geometrical supporting surface.

These three generic methods are used in the proposed two-pass advancing-front procedure to generate elements on the supporting surface. In the first pass, elements are generated based on geometrical criteria, which produces well-shaped elements. In the second pass, triangular elements are generated based solely on topology criteria.

The three generic methods were implemented to regenerate triangle mesh surfaces, where the only information about the surface is a set of triangles. This approach is very useful when a mesh needs to be refined, coarsened or improved. Some examples have demonstrated the

Fig. 11 Final triangle meshes of example surfaces



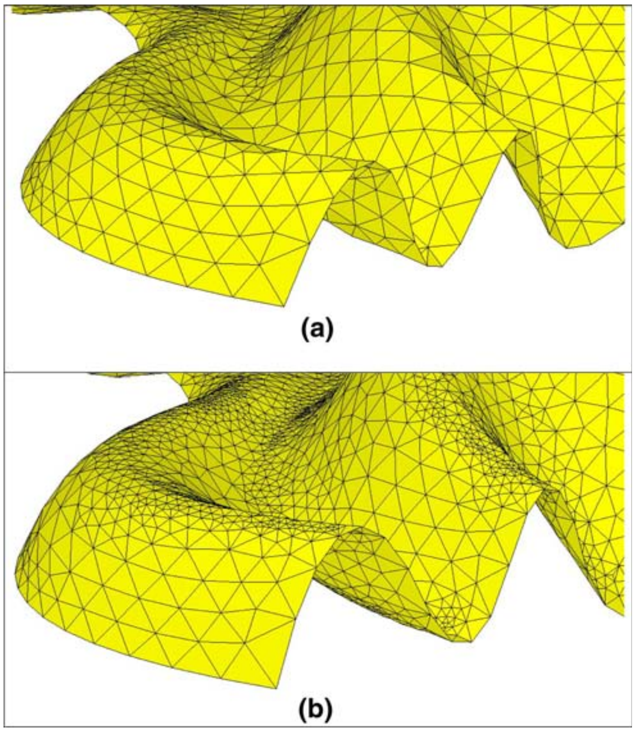


Fig. 12 Differences between meshes for Example 4 with (**a**) no consideration of curvature and (**b**) considering curvatures

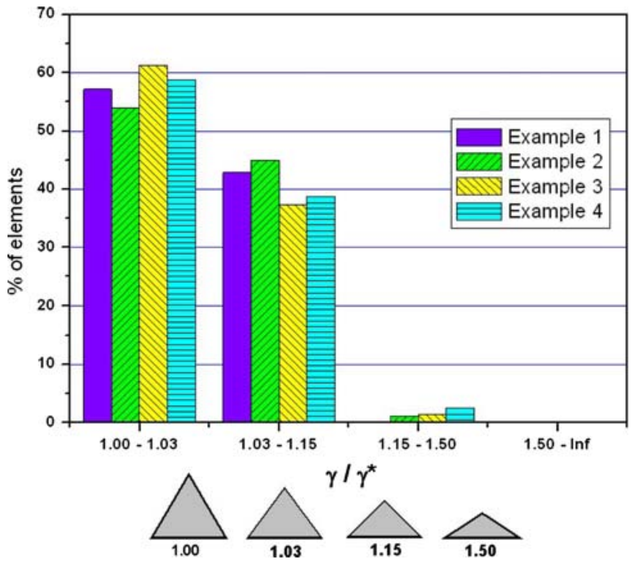


Fig. 13 Histogram of element quality for re-meshing the example surfaces

quality of the generated meshes and the importance of considering surface curvatures in local mesh refinement.

Acknowledgments The first author acknowledges a post-doctoral fellowship provided by the Brazilian agency CAPES, process 2183-06, and the Cornell Fracture Group. The second author has a PQ grant from CNPq. This work has been developed in the Cornell Fracture Group and in Tecgraf/PUC-Rio, Computer Graphics Technology Group. Tecgraf/PUC-Rio is partially supported by PETROBRAS.

References

1. Cavalcante Neto JB, Wawrzynek PA, Carvalho MTM, Martha LF, Ingraffea AR (2001) An algorithm for three-dimensional mesh generation for arbitrary regions with cracks. Eng Comput 17(1):75–91
2. Miranda ACO, Martha LF (2002) Mesh generation on high curvature surfaces based on background Quadtree structure. In: Proceedings of 11th International Meshing Roundtable 1, pp 333–341
3. Miranda ACO, Cavalcante Neto JB, Martha LF (1999) An algorithm for two-dimensional mesh generation for arbitrary regions with cracks, SIBGRAPI'99. In: Stolfi J, Tozzi C (eds) XII Brazilian Symposium on Computer Graphics, Image Processing and Vision, IEEE Computer Society Order Number PRO0481, ISBN 0-7695-0481-7, pp 29–38
4. Miranda ACO, Meggiolaro MA, Castro JTP, Martha LF, Bittencourt TN (2003) Fatigue life and crack path predictions in generic 2D structural components. Eng Fract Mech 70(10):1259–1279
5. Carlos J, Scheidegger E, Fleishman S, Silva CT (1996) Direct (re)meshing for efficient surface processing. Comput Graph Forum 25(3):527–536
6. Lohner R (1996) Regridding surface triangulations. J Comput Phys 126(1):1–10
7. Shostko AA, Lohner R, Sandberg WC (1999) Surface triangulation over intersecting geometries. Int J Numer Meth Eng 44:1359–1376
8. Nakahashi K, Sharov D (1995) Direct surface triangulation using the advancing front method. AIAA, pp 442–451
9. Lau TS, Lo SH (1996) Finite element mesh generation over analytical curved surfaces. Comput Struct 59(2):301–309
10. Lo SH, Lau TS (1998) Mesh generation over curved surfaces with explicit control on discretization error. Eng Comput Int J Comput Eng 15(3):357–373
11. Chan CT, Anastasiou K (1997) An automatic tetrahedral mesh generation scheme by the advancing front method. Commun Numer Methods Eng 13:33–46
12. Jin H, Tanner RI (1993) Generation of unstructured tetrahedral meshes by advancing front technique. Int J Numer Methods Eng 36:1805–1823
13. Lo SH (1985) A new mesh generation scheme for arbitrary planar domains. Int J Numer Methods Eng 21:1403–1426
14. Lohner R, Parikh P (1988) Generation of three-dimensional unstructured grids by the advancing-front method. Int J Numer Methods Fluids 8:1135–1149
15. Moller P, Hansbo P (1995) On advancing front mesh generation in three dimensions. Int J Numer Methods Fluids 38:3551–3569
16. Peraire J, Peiro J, Formaggia L, Morgan K, Zienkiewicz OC (1988) Finite Euler computation in three-dimensions. Int J Numer Methods Fluids 26:2135–2159
17. Rassineux A (1998) Generation and optimization of tetrahedral meshes by advancing front technique. Int J Numer Methods Fluids 41:651–674
18. Guttman A (1984) Rtrees: a dynamic index structure for spatial searching. In: Proceedings of ACM SIGMOD International Conference on Management of Data, pp 47–57
19. Rudolf B (1971) Binary B-Trees for virtual memory. ACM-SIGFIDET Workshop, San Diego, California, Session 5B, pp 219–235
20. Foley TA, Nielson GM (1989) Knot selection for parametric spline interpolation. In: Schumaker L (ed) Mathematical methods in CAGD. Academic Press, New York, pp 445–467

21. Borouchaki H, Hecht F, Frey PJ (1997) Mesh gradation control. In: Proceedings of 6th International Meshing Roundtable, Sandia National Laboratories, pp 131–141
22. Owen SJ, Saigal S (1997) Neighborhood-based element sizing control for finite element surface meshing. In: Proceedings of 6th International Meshing Roundtable, Sandia National Laboratories, pp 143–154
23. George PL, Seveno E (1994) The advancing-front mesh generation method revisited. Int J Numer Methods Fluids 37:3605–3619
24. Borouchaki H, Hecht F, Frey PJ (1997) H-Correction. INRIA Report No. 3199, INRIA, pp 29
25. Lohner R, Parikh P, Gumbert C (1988) Interactive generation of unstructured grid for three dimensional problems. Numerical grid generation in computational fluid mechanics '88. Pineridge Press, Swansea, pp 687–697
26. Owen SJ, Saigal S (2000) Surface mesh sizing control. Int J Numer Methods Fluids 47(1):289–312
27. Mello UT, Cavalcanti PR (2000) A point creation strategy for mesh generation using crystal lattices as templates. In: Proceedings of 9th International Meshing Roundtable, Sandia National Laboratories, pp 253–261
28. Zhu J (2003) A new type of size function respecting premeshed entities. In: Proceedings of 12th International Meshing Roundtable, Sandia National Laboratories, pp 403–413
29. Persson P (2004) PDE-based gradient limiting for mesh size functions. In: Proceedings of 13th International Meshing Roundtable, Sandia National Laboratories, pp 377–388
30. Moller T, Trumbore B (1997) Fast, minimum storage ray-triangle intersection. J Graphics Tools 2(1):21–28
31. Krysl P (2005) Computational complexity of the advancing front triangulation. Eng Comput 12:16–22