AUTOMATIC BOUNDARY SIZING FOR 2D AND 3D MESHES

Alexandre Cunha cunha@cmu.edu
Scott Canann scanann@cmu.edu
Sunil Saigal saigal@cmu.edu

Department of Civil and Environmental Engineering
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3890

1 ABSTRACT

The numerical solution of problems in science and engineering via the finite element method requires, as a first step, the discretization of a domain into a set of simply shaped elements. Determining the size of these elements along the domain, including the boundary, to form well-shaped elements is a difficult task. We present in this paper a simple technique, called *smart sizing*, which automatically computes high quality initial element sizing on curves for triangular, quadrilateral and tetrahedral elements. Curve divisions are computed based on curve and surface curvatures as well as feature proximity. In the three dimensional case, refinement of facets is performed as needed to create reasonably sized surface elements. Computing a boundary mesh appropriately is a key step to successfully determine the size and distribution of new elements towards the interior of the domain, especially for the advancing front and constrained Delaunay meshing techniques. The approach presented here is geometry based and does not attempt to account for the physics of the problem.

2 INTRODUCTION

2.1 PREVIOUS WORK

Most unstructured mesh generators apply one or more of the following methods: quadtree/octree (2D/3D domains), Delaunay triangulation (DT), and advancing front. Each has its own way of assigning element sizes along the domain to produce a mesh. In many quadtree/octree and DT approaches, the domain is refined in a top down fashion, breaking the elements to achieve a desired shape quality and/or target size. Many of them are not able to exactly match an initial boundary mesh because the element sizing is an intrinsic part of the generation process. On the other hand, advancing front mesh generators rely on a boundary mesh to construct new elements in the interior of the domain. They usually strictly preserve the boundary which is sometimes needed when meshing multiple domains in different stages. This suggests the definition of a constrained mesh generator (CMG) as one that preserves the input boundary as it is.

In quadtree and octree mesh generators, the entire domain is enclosed by an axes-aligned rectangular bounding box and then recursively subdivided until each leaf cell in the tree intersects the domain in a predefined way. The cells are then warped and cut to conform to the boundary and a triangulation is constructed from the remaining cells to form a triangular mesh. The subdividing phase may be governed by a user supplied spacing function or by a balance condition for the tree. Algorithms basically differ on the subdividing criteria and the method used to ensure conformity between boundary cells (the former being primarily responsible for the element sizes). Quadtree and octree generators include (Yerry and Shephard, 1983, 1984), (Baehmann et al., 1987), (Peruchio et al., 1989), (Shephard and Georges, 1991), (Mitchell and Vavasis, 1992), (Bern et al., 1994), and (Vavasis, 1996).

Some two dimensional mesh generators have made use of constrained Delaunay triangulation (CDT) (Chew, 1989a) to construct a triangular mesh. Most of them cannot be classified as constrained mesh generators because the boundary edges are subdivided to satisfy their shape quality criteria. Chew (1989b) was the first one to use a CDT that guaranteed the construction of a 2D mesh with all angles in the interval [30°, 120°], provided that the length of all given edges are between h and $(\sqrt{3})h$, and no two given vertices are closer than h, where h is the edge length chosen by the user. Since boundary edges not complying with these conditions are divided, the mesher is not a CMG. Extensions to his work have been made by (Ruppert, 1995), (Shewchuk, 1996a), and (Shewchuk, 1996b). In (Shewchuk, 1996b) it is possible to preserve the boundary edges to match adjacent meshes.

P. L. George and colleagues (1990, 1991, 1992) developed a CMG using a three dimensional DT. Given a boundary mesh, the DT of its vertex set is first constructed. New points are then inserted into the domain and a retriangulation forms the tetrahedral mesh. The boundary mesh is recovered applying a set of predefined transformations to adjust those elements crossing it. In a final step, the nodes are smoothed and some faces may be swapped to improve the element quality. Weatherill and Hassan (1992) also developed a similar CMG. A description of improved mesh generators using this technique can be found in (Weatherill and Hassan, 1994a), (Weatherill et al., 1994b) and (Borouchaki et al., 1995).

More recently, a mix of advancing front and DT in three dimensions, as described above, has been proposed to take advantage of the advancing front high quality point placement strategy and DT spatial information (Marcum and Weatherill, 1995, Pascal et al., 1996). Advancing front is known to produce high quality elements around the boundary (point placement) while DT has definite speed advantages.

Since the generators described above preserve the boundary elements and use them to generate new elements inside the domain, they need a well sized boundary mesh to produce high quality final elements. The present work shows a technique to construct such boundary meshes.

2.2 MOTIVATION

The goal of this work is to develop a robust strategy to automatically construct well sized boundary meshes that can be used with either the advancing front or constrained Delaunay mesh generators. The basic idea behind the scheme described here is to perform successive refinements on the boundary mesh of the domain at each iteration until discretization errors are minimized and reasonable quality elements can be expected.

2.3 NOTATION AND BASIC DEFINITIONS

Parametric curves with parametrization t and points are denoted as lowercase boldface letters, like $\mathbf{r}(t)$ and $\mathbf{p}$. Parametric surfaces with parametrizations u and v and sets are denoted by uppercase italic letters, as in $S(u,v)$ and R. For simplicity, parametrizations considered here are in the unit interval [0,1]. Other geometric and mesh entities are represented in lowercase italic letters, as in v for vertices and n for nodes. Greek letters represent scalar quantities. Vertices are referred to as geometric entities, while nodes are mesh components. Both have their position defined by a point. Mesh entities are composed of a set of nodes $N = \{n_i\}$, a set of edges $E = \{e_i\}$ connecting two nodes of the mesh and a set of elements $F = \{f_i\}$ formed by a list of edges. Any curve has two end vertices and a surface is comprised of a set of boundary curves. A volume is formed by a set of surfaces.

To facilitate illustration, some drawings are restricted to two dimensions. Nevertheless, the method described extends to three dimensions, unless otherwise mentioned.

This paper is presented as follows: in section 3, the smart sizing algorithm is presented. Subsections 3.2 and 3.3, respectively, show how curvature is computed for nodes and how proximity refinement is performed based on the distance between nodes and edges. In section 4, surface proximity checking is explained. In section 5 a simple user interface is discussed and section 6 gives a conclusion.

3 SMART SIZING

3.1 ALGORITHM

The smart sizing algorithm is as follows:

```
0.  Let $R = \{\mathbf{r}_i\}$ be the set of curves from given geometry and
    let $V = \{v_i\}$ be its set of vertices
1.  Compute (or allow user to input) a global meshing size $h$
    and a minimum meshing size $h_{min}$
2.  Assign a size for each node $n_i$ at each vertex $v_i$ to be the
    smallest value among the following:
        - global meshing size $h$
        - length of curves incident to $v_i$
        - size due to curvature, i.e., ksize($n_i$)
3.  Process vertices to find small angles
4.  Mesh the curves, using vertex sizes
5.  Begin main loop
6.      Assign sizes for new nodes, size($n$)
7.      Curvature refinement: for each node $n_i$ do
            size($n_i$) = min(size($n_i$), ksize($n_i$))
8.      Proximity refinement: for each edge $e_j$ do
            update size of nearby non-connected nodes
            based on their distance to $e_j$
9.      Remesh the curves
10.     End loop if:
            - the maximum number of iterations is reached,
            - or no modifications were made
```

The global meshing size h is an initial guess for the edge length in all curves. If the user does not provide this value, then it can be computed from the input geometry simply by multiplying the diagonal of the rectangular bounding box of the domain by a scaling factor.

A minimum meshing size, h_{min}, is provided so as to restrict the algorithm from producing too many nodes in extreme situations. It is simply the h value divided by a coefficient much larger than 1, such that $0 < h_{min} << h$.

In step 2, the size of the nodes at the vertices are initialized. The size is computed as the minimum of the sizes due to curvature (ksize(n)), the meshing global size (h) and the length of the curves connected to the node. The curvature size, ksize(n), is computed as follows:

- Evaluate curvature, k, at the node position on curve;
- Locally approximate the curve at that position as a circle with radius R equals to $1/k$;
- Compute the size s of the edge that spans an angle of ϕ, where ϕ, defined as the maximum spanning angle, is the maximum angle spanned along the circle; according to Fig. 1, $s = 2R\sin(\phi/2)$ or $s = 2\sin(\phi/2)/k$;

- ksize(n) = s.

Thus, the edge size is a linear function of the radius. As expected, small curvature will give large edges and large curvature will give short edges. For straight lines or almost straight lines ($k \approx 0$), curvature does not affect sizing.

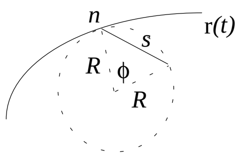

Fig. 1 Node size due to curvature

Curves are now meshed using the sizes at their end nodes. For any two nodes with sizes s_j and s_{j+1} on the curve, the scheme shown in Fig. 2 is used. New nodes are inserted on the curve segment of length L, defining m new edges, each one with a size proportional to s_j (where $s_j < s_{j+1}$). A smooth transition for the edges is required such that the ratio of the lengths of any two adjacent edges is in the interval $[1/\lambda, \lambda]$, where λ is defined as the growth ratio (or grading factor). Note that given the end node sizes, s_j and s_{j+1} ($= \lambda^{m-1}s_j$), and the segment length L, we can determine the growth ratio and the number of new nodes on that segment. The computed growth ratio is restricted to be less than or equal to a user specified value. The growth ratio is similar to Joe's smoothing parameter (Joe, 1986), which controls the variation in size between adjacent elements. The

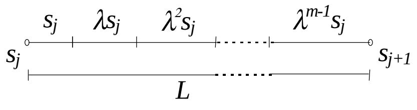

Fig. 2 Meshing scheme for a curve

pseudo-code below returns the set of nodes for a curve segment of **r**(t):

```
mesh_curve ($n_j$, $n_{j+1}$, r, $\lambda$)
    $s_j$ = size of node $n_j$
    $s_{j+1}$ = size of node $n_{j+1}$
    $L$ = length of r between $n_j$ and $n_{j+1}$
    $\lambda$ = min($\lambda$, growth_ratio($s_j$, $s_{j+1}$, $L$))
    $N = \{n_j\}$
    $\Delta = s_j$
    $t = \Delta/L$
    while (t < 1) do
        $n$ = node at r$(t)$
        $N = N \cup \{n\}$
        $\Delta = \lambda\Delta$
        $t = t + \Delta / L$
    enddo
    $N = N \cup \{n_{j+1}\}$
    return $N$
end
```

An adjustment of the last edge size is performed as needed to match s_{j+1} as closely as possible. For the special case when $s_j \approx s_{j+1} \approx h$ nodes are evenly distributed along the curve. The example in Fig. 3 shows a geometric model, its lines and the line divisions after **mesh_curve** is called for all them.

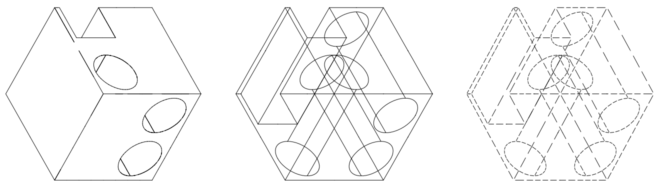

Fig. 3 An object, its lines and their divisions

Once new nodes are created, their sizes are assigned in step 6. The size of a new node is set to be the average length of the edges connected to it. These sizes may be reduced later depending on the curvature at the nodes (step 7) and their proximity to other edges (step 8). After updating the size of the current nodes, the curves are remeshed as described above and steps 6 to 9 are repeated until no modifications are made to the nodal size settings (no new nodes created) or the maximum number of iterations is reached, which is, typically, fairly small (experience has shown four iterations to be adequate).

3.2 CURVATURE AT A NODE

In steps 2 and 7 of the smart sizing algorithm, the curvature value at a node on the curve is necessary to compute its size. This value depends not only on the shape of the curve itself, but also on the curvature of each surface connected to the curve. As an illustration, consider the cylinder of Fig. 4 and its geometric representation. If only line curvature is evaluated, the meshing of the straight line lying on the curved surface of the cylinder might produce a wrong number of nodes which could cause problems when meshing the surface along the line.

Normal curvature at a point on a surface curve has two extreme values in orthogonal directions called the principal curvatures, k_1 and k_2 (O'Neill, 1966). The maximum of these two values is used together with the line curvature to evaluate the final curvature at a node (see Fig. 5) and they are most

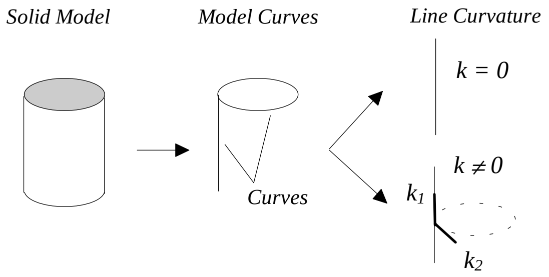

Fig. 4 Principal curvatures avoid wrong evaluation of line curvature

significant for some type of surfaces, as illustrated above, due to their geometric representation or due to their construction itself. Repetitively computing the principal curvatures is quite expensive and some simplifications are made. The curvature at an end node of a curve is the maximum curvature of all its connected curves when evaluated at the node position. No surface curvature is computed because it is anticipated that the line curvature will generally match the surface curvature at such nodes. For example, for node n_j in Fig. 5, $k(n_j) = k_j = max(k(l_1), k(l_2) k(l_3))$, where $k(l_i)$ denotes the curvature of the line l_i at n_j.

For each curve, the largest principal curvature, is computed only once at its middle point, $k_{m/2}$. For any node n on the curve, except the end nodes, $k_{m/2}$ and the curvature of the closest end node are linearly interpolated to give the node's surface curvature value, $k_s(n)$. The final curvature k at a node n is the maximum value of its surface curvature and its line curvature, $k_l(n)$, $k(n) = max(k_s(n), k_l(n))$. The largest principal curvature at the curve middle point is computed as follows: for all surfaces connected to the curve, find k_1 and k_2 at this point and take the maximum.

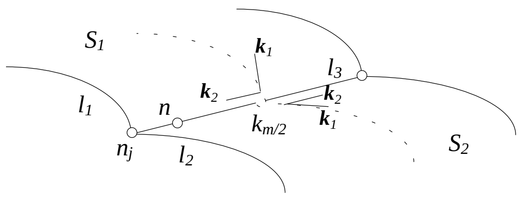

Fig. 5 Curvature computation

An exception to the algorithm for using curvature for sizing can occur when small holes or small curved features are present in the geometry. As was shown, the edge size due to curvature is a linear function of the radius of the circle defined by that curvature. For very small *radii*, the edge sizes will be very small and thus for small holes a large number of tiny edges can be constructed. If the number of elements is to be minimized in such cases, the right approach is to simplify the geometry. Some research has been done in this area to avoid unnecessary elements in the analysis step, either removing such small features (Shephard, 1989) or substituting parts of the geometry by reduced dimension elements, such as beams, plates and shells (Donaghy *et. al.*, 1996).

Small curved features are tracked by comparing the edge size s due to curvature against the global meshing size h and modifying s, if necessary. If the ratio h/s is above a pre-defined threshold, 50:1 for example, a small curved feature has been encountered and s is computed again using an angle Φ larger than the maximum spanning angle ϕ (45° or 90°, for example), $s = 2Rsin(\Phi/2)$, $\Phi > \phi$ (Fig. 6). If the ratio is of the same order of magnitude, up to 5:1 for example, the s value is kept (the hole is not considered small). For values in-between, a linear interpolation of the angles Φ and ϕ is used and a new angle proportional to the current ratio is computed. The edge size is then computed from this new angle.

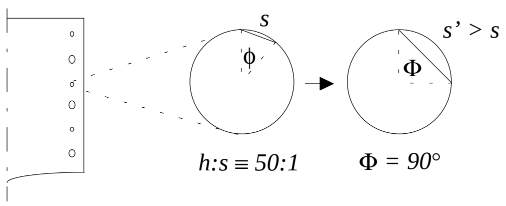

Fig. 6 Small holes

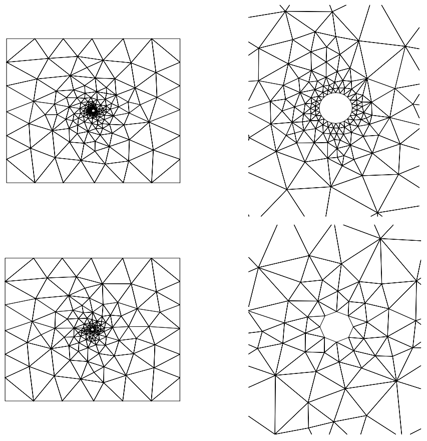

Fig. 7 Meshing with small holes

Fig. 7 shows a small hole centered in a square plate. The mesh on the top (bottom) was obtained before (after) applying the technique described above. An angle of Φ = 45° was used to get eight divisions on the hole line, thus decreasing the number of elements in the mesh on the bottom. Note that when such small holes are desired for an analysis, small hole coarsening can be turned off.

3.3 PROXIMITY

Proximity between edges and nodes is computed to modify the node size, such that reasonably shaped elements can be placed in small areas between nearby lines. This is the most important purpose of smart sizing and most directly affects element quality. For each edge in the current mesh, the closest nodes to it are found and, according to their distance to the edge, their corresponding size may be reduced. The procedure for proximity refinement is as follows (see Fig. 8):

```
curve_proximity ($e$, $\lambda$, bins)
    $N$ = nearby_nodes($e$, bins)
    $L$ = length($e$)
    $\Delta$ = average size of end nodes of edge $e$
    for each $n \in N$ do
        $d$ = euclidean_distance($n$, $e$)
        if ($d > L$) then
            $s = \Delta + (\lambda - 1) d$
        else
            $s = \lambda d$
        endif
        $s$ = max(min($s$, $h$), $h_{min}$)
        size($n$) = min(size($n$), $s$)
    enddo
end
```

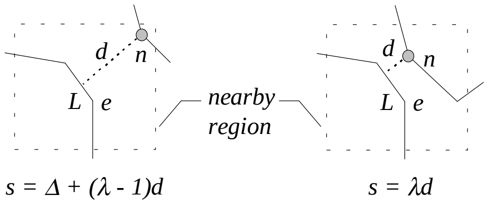

Fig. 8 Node size due to proximity

The λ used in **curve_proximity** is the growth ratio introduced in section 3.1. The nearby region of an edge is determined by its length and the current maximum node size. A bin sort (Samet, 1990) is used to efficiently find the nodes near the edge. It is assumed that the size of any node outside the nearby region is not affected by the edge being analyzed. Note that the new size s is in the interval $[h_{min}, h]$ and that the node size only changes if it is larger than s. This approach tries to keep approximately the same edge length for nearby edges through the adjustment of the sizes of their nodes such that the nearby region can be covered with well-shaped tetrahedrons of that length.

Fig. 9 shows an object meshed before (left picture) and after proximity is considered. Note the element quality improvement around the holes.

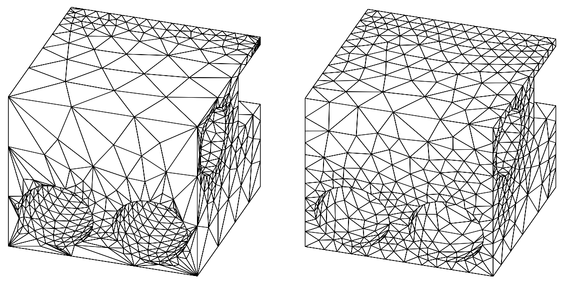

Fig. 9 Meshes without (left) and with proximity considered

An exception to the proximity refinement algorithm occurs when small angles are present in the input geometry which can cause problems when updating the node size due to proximity. Two types of sharp corners are (Fig. 10):

- constant angle: the small angle does not change when approaching the corner;
- decreasing angles: the angle decreases when approaching the corner vertex - there is at least one curved line involved.

A loop over the vertices is done in step 4 to determine if small angles are present and to classify them according to the two types above. In both cases, the size of the edges forming the corner are frozen.

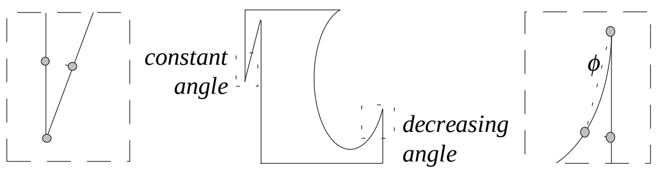

Fig. 10 Sharp corners: constant and decreasing angle

For a constant corner, the size of both corner edges is $\max(h_{min}, \min(h, \lambda l_1, \lambda l_2))$, where l_1 and l_2 are the lengths of the curves forming the corner and λ is the growth ratio. If a tiny curve is present (its length is much less than h) one edge is enough to cover it. Smart sizing cannot take the place of geometric defeaturing – it simply meshes the boundary of the geometry given to it.

For decreasing angle corners, a walk is done on its curves until the edges on these curves form an angle greater than or equal to a minimum corner angle. Then the sizes of these edges are checked for interval $[h_{min}, h]$ and modified, if necessary.

In Fig. 11, an object with these two types of corners is meshed. Note that the shape of the elements on the round corner is improved by increasing the element angles.

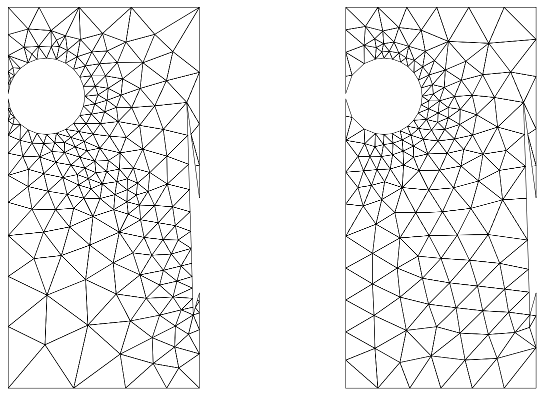

Fig. 11 Meshes before (left) and after applying the technique for sharp corners

4 SURFACE PROXIMITY

It is not always possible to catch surface proximity during the curve meshing phase. That is because determining the proximity of curves does not necessarily determine proximity of the surfaces on which the curves lie on. This is often true for certain representations of geometric primitives and also due to the combinatorial nature of how these primitives could be assembled to form the domain being analyzed. Hence, in some situations the meshing of surfaces can yield large triangular facets on nearby surface areas where there were no curves to detect the proximity of such surfaces. An example of this type of surface proximity would be a large sphere packed inside a cube.

Some facets may be large enough to either compromise the quality of the tetrahedrons that will be inserted between them or even lead to meshing failures. To overcome this problem, the size of the facets are decreased to eliminate the probability of producing poorly shaped tetrahedrons. This is done through evaluation of the proximity of the facets, which is much less expensive, in most cases, than doing it via the continuum description of the surfaces.

The surface proximity checking step is a four fold procedure:

- Preprocess facets and compute their size;
- Determine the size at nodes and the distance between the nodes and nearby facets;
- Mark nodes for refinement if the distance is less than the node's size;
- Perform successive refinements of the facets corresponding to the marked nodes until they reach an acceptable size.

The size at a node is the size of the largest facet connected to the node. The size of a facet f is the distance from the facet to the opposite point in the equilateral tetrahedron whose edge

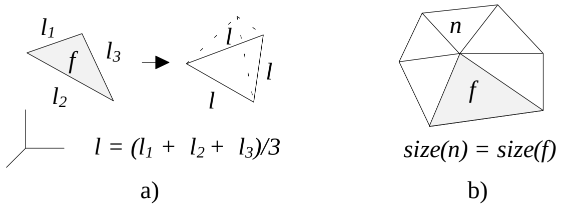

Fig. 12 a) Equilateral tetrahedron from facet b) node size

length is the average length of the face's edges (Fig. 12a). That is, if the edge lengths are l_1, l_2 and l_3 then the facet size is size(f) = ($\sqrt{2/3}$)l, where $l = (l_1 + l_2 + l_3)/3$. This tetrahedron is an approximation of the best tetrahedron to attach to the facet. In Fig. 12b, the size at node n is the size of the shaded facet f. The surface proximity procedure for the surface mesh $M = (N, E, F)$ is as follows:

```
surface_proximity ($M$)
    preprocess facets:
        assign size of each $f_i \in F$
        $B = \{b_i : b_i$ is bounding box of $f_i\}$
        sort $B$
    for each $n \in N$ do
        $s = \max_i(\{\mathrm{size}(f_i) : f_i$ is adjacent to $n\})$
        $b_n$ = bounding box of $n$
        $I$ = intersect($B$, $b_n$)
        $d_{min} = +\infty$
        for each $b \in I$ do
            $d$ = distance($n$, face($b$))
            if ($d < s$) then
                $d_{min} = \min(d_{min}, d)$
            endif
        enddo
        if ($d_{min} < s$) then
            level($n$) = $\lfloor log_2(s/d_{min}) \rfloor$
            $R = R \cup \{n\}$
        endif
    enddo
    if ($R \neq \emptyset$) then
        refine($M$, $R$)
    endif
end
```

The procedure requires that M be a consistently oriented conformal mesh, that is:

- the normal vectors of all facets are consistently oriented inward or outward from the domain;
- the intersection of any two facets is an edge connecting two nodes of the mesh, a node, or the empty set. This is not always guaranteed after the surfaces have been meshed, especially when they are very close to each other which can cause the facets to intersect in a nonconformal way.

These requirements guarantee a proper intersection test and computation of node-facet distances. They are also required for suppression of refinement of facets improperly reported by the intersect routine and of large facets that are very close but do not face each other – when the bounding boxes of the node and the facet intersect, the normals of the node (average normal of connected facets) and the facet are compared to check if they face each other).

In the preprocessing stage, the rectangular bounding boxes of the facets are *sorted* such that the intersection with the node bounding box can be determined. Each bounding box b_i is defined by open intervals $(x_{l,i}, x_{r,i})$, $(y_{l,i}, y_{r,i})$, and $(z_{l,i}, z_{r,i})$ along, respectively, the x, y, and z directions. These intervals are sorted separately in each dimension using their end values and intersections are performed against the node bounding box on a one-dimensional basis. The results for each dimension are then combined to determine the final set of intersected boxes.

The bounding box b_n of a node is a square centered at the node position $\mathbf{p}$ with edge length equal to two times the node size s (Fig. 13), $b_n = (\mathbf{p} - s, \mathbf{p} + s)$. Similar to curve proximity, this bounding box defines the nearby region of the node whose final size might be affected by the distance to the facets in its interior.

When no intersections occur, the node will not be marked

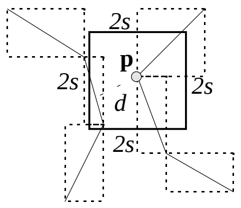

Fig. 13 Node bounding box and distance to a facet

for refinement. Otherwise, we find the minimum node-facet distance and if it is below the node size, we mark it with a level of refinement proportional to $log_2(s/d_{min})$, where log_2 is the logarithm base two. If not, the node remains unmarked.

Refinement is then done on all facets corresponding to marked nodes. The refinement used here is based on Schneiders and Debye algorithms (1995) as adapted and implemented by Staten (1996). The refinement is driven by a set of regular refinement templates, each one constructed according to the number of nodes marked for refinement in the triangle. Staten's work also includes topology cleanup and smoothing after each refinement iteration. Though the connectivity and position of nodes may slightly change between iterations, such modifications do not significantly alter the proximity between nodes and facets. Thus, the integrity of the node's refinement level computed before any refinement take place is preserved for all iterations.

A single refinement at a time is done on these facets and after each iteration new facets and new nodes are introduced into the mesh. The refinement level of these new nodes will be the average refinement level of their adjacent nodes. At each iteration, the refinement level of a node decreases by one because the size of its largest adjacent facet is cut by half in the refinement. The iteration stops when all marked nodes have a refinement level equal to one – when the distance of nearby facets is approximately equal to their sizes.

5 INTERFACE

The smart sizing user should be able to easily get a fine or coarse mesh without having to know the details involved in the development of the technique. Assigning the correct parameter values on a trail and error basis could be a time consuming task, especially for a novice user. In order to make things easier and more transparent, a set of predefined parameter values have been grouped, making up a collection of ten smart sizing levels, each one assigning different values for some of the input parameters. The user can then chose from one of these sizing levels by simply picking a number from one (fine) to ten (coarse). Table 1 shows the parameters used in each level.

Parameter	Description
μ	*scaling factor applied to the computed default meshing size; values in the interval [0.2, 5.0] are acceptable; 1.0 is default.*
ϕ_1	*maximum spanning angle for linear elements; values in the interval [9.0, 30.0] are used. 15.0 is default.*
ϕ_2	*maximum spanning angle for quadratic elements; values in the interval [15.0, 30.0] are used; 30.0 is default.*
λ	*growth ratio for line and area element size expansion and used for proximity checking; values in the interval [1.2, 5.0] are allowed; 1.2 is default.*
I	*maximum number of sizing iterations*

Table 1 Interface parameters for each smart sizing level

In table 2, these parameter values are shown for levels 3 and 8, and also for the intermediate level 6. Note that neither an interpolation scheme nor another analytical approach was used to set up these values. They are simply the results of experimental work done with a collection of models and have been tuned to handle most situations. To illustrate the variation on the meshes obtained when levels are changed, a simple geometry is meshed (Fig. 14). Note that the element quality, as expected, is proportional to the number of elements achieved. Typically, a very coarse mesh (level 10) would produce a larger proportional number of poorly shaped elements than a finer one (levels 6 or 1).

Level	μ	ϕ_1	ϕ_2	λ	I
8	1.875	22.0	30.0	1.7	4
6	1.000	15.0	30.0	1.5	4
3	0.300	10.0	18.0	1.3	4

Table 2 Parameter values at levels 8, 6, and 3

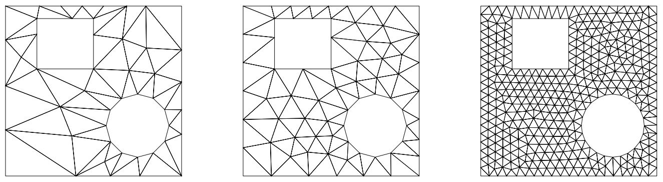

Fig. 14 Meshes at levels (from left to right) 8, 6 and 3

6 CONCLUSION

A technique to construct well sized boundary meshes is presented and several examples are shown. Successive refinement due to curvature and proximity are the keys to reasonably distributing nodes along curves so as to form well shaped elements. Smart sizing does not take the place of geometric defeaturing – it simply meshes the geometry given to it as best it can. Surface proximity refinement is performed when surface facet sizing may lead to poorly shaped tetrahedrons. A somewhat arbitrary set of smart sizing levels are also presented so that the user can easily choose one that is most suitable for his needs.

Acknowledgments

The authors would like to thank CNPq/Brazil and Ansys, Inc. for their financial support to make this work possible.

7 REFERENCES

(Baehmann et al., 1987) Baehmann, P. L., Wittchen, S. L., Shephard, M. S., Grice, K. R., and Yerry, M. A., *Robust geometrically-based automatic two dimensional mesh generation*, Intl. J. Num. Meth. Eng., 24, pp. 1043-1078, 1987.

(Bern et al., 1994) Bern, M. J., Eppstein, D., and Gilbert, J. R., *Provably good mesh generation*, J. Comp. System Sciences, 48, pp. 384-409, 1994.

(Chew, 1989a) Chew, L. P., *Constrained Delaunay Triangulations*, Algorithmica, 4, pp. 97-108, 1989.

(Chew, 1989b) Chew, L. P., *Guaranteed-quality triangular meshes*, Tech. Report TR-89-983, Cornell University, 1989.

(Donaghy et al., 1996) Donaghy, R. J., McCune, W., Bridgett, S. J., Armstrong, C. G., Robinson, D. J., and McKeag, R. M., *Dimensional reduction of analysis models*, In Proc. 5th International Meshing Roundtable, pp. 307-329, 1996.

(George et al., 1990) George, P. L., Hecht, F., and Saltel, E., *Automatic 3D mesh generation with specified boundary*, IEEE Transactions on Magnetics, 26(2), pp. 771-774, 1990.

(George et al., 1991) George, P. L., Hecht, F., and Saltel, E., *Automatic mesh generation with specified boundary*, Comp. Meth. Applied Mech. Eng., 92, pp. 269-288, 1991.

(George and Hemerline, 1992) George, P. L. and Hemerline, F., *Delaunay's mesh of a convex polyhedron in dimension d: application for arbitrary polyhedron*, Intl. J. Num. Meth. Eng., 33, pp. 975-995, 1992.

(Joe and Simpson, 1986) Joe, B. and Simpson, R. B., *Triangular meshes for regions of complicated shape*, Intl. J. Num. Meth. Eng., 23, pp. 751-778, 1986.

(Marcum and Weatherill, 1995) Marcum, D. And Weatherill, N. P., *Unstructured grid generation using iterative point insertion and local reconnection*, AIAA J., 33(9), pp. 1619-1625, 1995.

(Mitchell and Vavasis, 1992) Mitchell, S. A., and Vavasis, S., *Quality mesh generation in three dimensions*, In Proc. 8th ACM Symp. Comp. Geom., pp 212-221, 1992.

(O'Neill, 1966) O'Neil, B., *Elementary differential geometry*, Academic Press, New York, 1966.

(Peruchio et al., 1989) Peruchio, R., Saxena, M., and Kela, A., *Automatic mesh generation from solid models based on recursive spatial decompostion*, Int. J. Num. Meth. Eng., 28, pp. 2469-2502, 1989.

(Ruppert, 1995) Ruppert, J., *A Delaunay refinement algorithm for quality 2-dimensional mesh generation*, Journal of Algorithms, 18(3), pp. 548-585, 1995.

(Samet, 1990) Samet, H, *Design and analysis of spatial data structures*, Addison Wesley, 1990.

(Shewchuk, 1996a) Schewchuk, J. R., *Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator*, In Proc. 12th ACM Symp. Computational Geometry, 1996.

(Shewchuk, 1996b) Schewchuk, J. R., *Triangle Home Page*, http://www.cs.cmu.edu/~quake/triangle.html.

(Schneiders and Debye, 1995) Schneiders, R., and Debye, J., *Refinement algorithms for unstructured quadrilateral or brick element meshes*, Modeling, Mesh Generation, and Adaptive Num. Meth. for PDEs, IMA - Springer-Verlag, 1995.

(Shephard, 1989) Shephard, M. S., *Algorithmic approach to eliminating small features from the finite octree: dual representation of features*, Tech Rep., SCOREC, Rensselaer Polytechnic Institute, Troy, N.Y., 1989.

(Shephard and Georges, 1991) Shephard, M. S. and Georges, M. K., *Automatic three dimensional mesh generation by the finite octree technique*, Int. J. Num. Meth. Eng., 32, pp. 709-749, 1991.

(Staten, 1996) Staten, M. L., *Selective refinement of two and three-dimensional finite element meshes*, Master's Thesis, Brigham Young University, 1996.

(Vavasis, 1996) Vavasis, S., *QMG Web Page*, 1996. Http://www.cs.cornell.edu/Info/People/vavasis/qmg-home.html.

(Weatherill and Hassan, 1992) Weatherill, N. P., and Hassan, O., *Efficient three dimensional grid generation using the Delaunay triangulation*, In Proc. 1st European CFD Conference, Elsevier, 1992.

(Weatherill and Hassan, 1994a) Weatherill, N. P., and Hassan, O., *Efficient three dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints*, Int. J. Num. Meth. Eng., 37, pp. 2005-2039, 1994.

(Weatherill et al., 1994b), Weatherill, N. P., Hassan, O., Marcum, D. L., and Marchant, M. J., *Grid generation by the Delaunay triangulation*, von Karman Institute for Fluid Dynamics, 1994.

(Yerry and Shephard, 1983) Yerry, M. A. and Shephard, M. S., *A modified quadtree approach to finite* element *mesh generation*, IEEE Computer Graphics and Applications, 3, pp. 39-46, 1983.

(Yerry and Shephard, 1984) Yerry, M. A. and Shephard, M. S., *Automatic three dimensional mesh generation by the modified octree technique*, Int. J Num. Meth. Eng., 20, pp. 1965-1990, 1984.