ADAPTIVE REFINEMENT OF UNSTRUCTURED FINITE-ELEMENT MESHES *

MARK T. JONES AND PAUL E. PLASSMANN†

Abstract. The finite element method used in conjunction with adaptive mesh refinement algorithms can be an efficient tool in many scientific and engineering applications. In this paper we review algorithms for the adaptive refinement of unstructured simplicial meshes (triangulations and tetrahedralizations). We discuss bounds on the quality of the meshes resulting from these refinement algorithms. Unrefinement and refinement along curved surfaces are also discussed. Finally, we give an overview of recent developments in parallel refinement algorithms.

Key words. adaptive refinement, finite-element meshes, parallel algorithms, unstructured meshes

1. Introduction.

The combination of the finite-element method and adaptive mesh refinement has proven effective in a variety of applications. This review focuses on algorithms for refining unstructured meshes composed of triangles or tetrahedra. We say a mesh is *structured* if it has a regular pattern of connections between elements. Consider the three mesh examples shown in Figure 1. Note that it is the pattern of connections that makes a mesh structured, not the geometric positions of the mesh points. The left and center meshes have the same local connectivity even though the center mesh has an irregular boundary. This regular structure can be exploited in algorithms and software to create highly efficient programs. It is not always possible, however, to create high-quality structured meshes for complex domains; in such cases, unstructured meshes must be used. This paper focuses on such unstructured meshes, which include meshes with no discernible local structure, such as the example on the right in Figure 1.

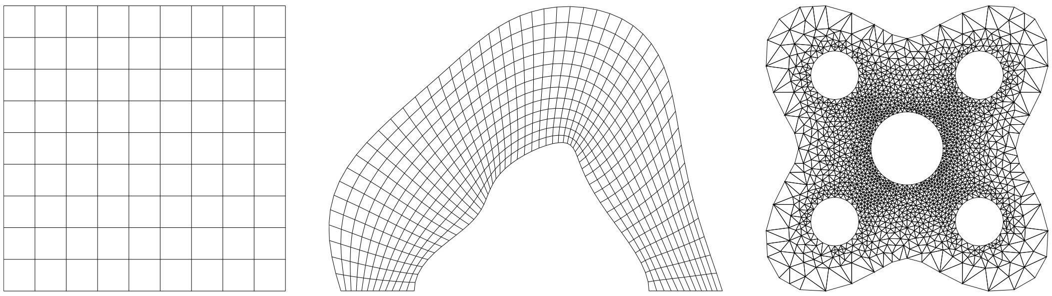

FIG. 1. *On the left is a structured mesh of a regular domain. In the center is a structured mesh on an irregular domain. On the right is an unstructured mesh on an irregular domain.*

* The first author received support from NSF grants ASC-9501583 and ASC-9411394. The second author was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

† The address of the first author is Computer Science Department, University of Tennessee, Knoxville, TN 37996. The address of the second author is Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439.

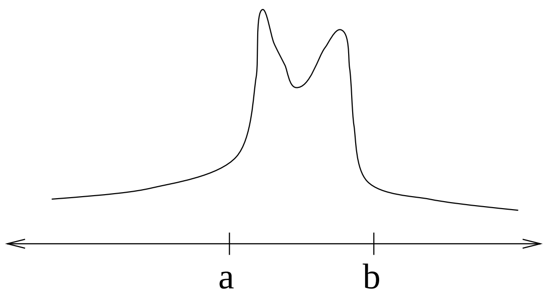

FIG. 2. *An optimal mesh for the linear finite-element approximation of this one-dimensional function would include a higher density of points in region (a, b) than in the rest of the domain.*

An unstructured mesh can be adaptively refined to improve the quality of a computed solution. Adaptive refinement is used most effectively in applications where some mesh elements are required to be much smaller than others yet the areas of refinement cannot be predicted prior to computation. This situation can arise when the computed solution is rapidly changing in small areas of the domain while in large parts of the domain the solution is relatively stable. For example, consider the one-dimensional function shown in Figure 2. An optimal mesh used for a linear finite-element approximation to this function would have a concentration of mesh points in the interval (a, b) but relatively few points in the remainder of the domain.

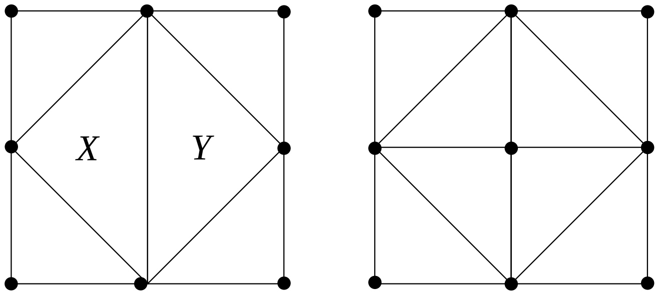

FIG. 3. *On the left, triangles X and Y are marked for refinement. The refined triangulation on the right is nested within the original triangulation on the left.*

In this paper we discuss adaptive refinement algorithms for generating meshes in two and three dimensions. We assume that an initial mesh, M_0, has been generated consistent with the topology of the problem domain; for a survey of the mesh generation problem we recommend the review by Bern and Eppstein [8]. We focus on algorithms for which the triangles or tetrahedra in the refined mesh, M_{i+1}, are nested within the old mesh, M_i. For example, on the left in Figure 3 is a mesh with two triangles marked for refinement; the resulting refined mesh is on the right. This nesting is motivated by the finite-element method. The finite-element spaces $V_0, V_1, \ldots, V_k$ corresponding to the nested meshes $M_0, M_1, \ldots, M_k$ are also to be nested, that is, $V_0 \subseteq V_1 \subseteq \ldots \subseteq V_k$. For many important classes of problems the Galerkin method used with these meshes leads to a sequence of discrete approximations with monotonic convergence properties. In addition, the

Construct a coarse mesh, M_0, consistent with the geometry
    of the problem domain
$i = 0$
While (Error estimates are not acceptable) **do**
    Compute a solution, S_i, on M_i
    Compute a local error estimate or indicator on each element of M_i
    Mark elements in M_i for refinement
    If (Marked elements exist)
        Construct M_{i+1} by refining marked elements of M_i
    Endif
    $i = i + 1$
Enddo

FIG. 4. *A typical adaptive refinement algorithm*

nested meshes can be used in conjunction with multigrid solution methods [3], [23].

An alternative refinement approach based on Delaunay triangulations inserts new mesh points in element interiors (Steiner insertion) and is popular for finite-volume methods in computational fluid dynamics and related areas. We recommend the papers by Chew [9], Ruppert [34], and Weatherill et al. [36] for details on this approach. We also note that the h-refinement discussed in this paper can be combined with order refinement of the basis functions; we recommend the papers [10], [24], and [28] for more information on hp-adaptive finite-element methods. A good overview of a number of research topics in adaptive methods can be found in the book by Flaherty et al. [12].

A typical finite-element adaptive refinement approach is summarized in Figure 4. We note that efficiency is the primary reason for selectively, rather than uniformly, refining a mesh. To demonstrate the gain in efficiency, consider solving Poisson's equation on the unit square with Dirichlet boundary conditions. We assume the solution given by

$$u(x,y) = x(x-1)y(y-1)e^{(-100((x-.5)^2+(y-.117)^2))} \tag{1.1}$$

and use a source for the discretized problem computed from this solution. We seek a discrete solution for which the estimated L_∞ error between the linear finite-element approximation and this exact solution is less than some fixed tolerance. This test problem is based on problem 1a in [22], which considers a number of error estimators. For a more detailed analysis of error estimates for the finite-element method, we recommend the paper by Babuška and Rheinboldt [2].

The refined mesh is given in Figure 5; note the refinement in the region corresponding to the peak in the source function. This selectively refined mesh has 600 triangles. To achieve the same level of accuracy with a uniformly refined mesh, we would have had to refine the mesh to a spacing consistent with the smallest elements used in the adaptive mesh. This uniform mesh would have required approximately 65,000 triangles. Mitchell [22] notes that when he uses the linear

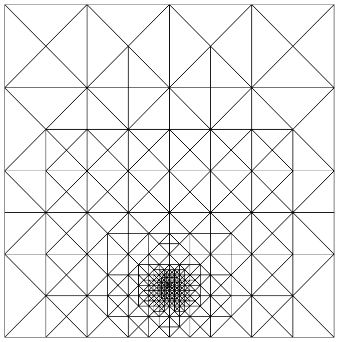

FIG. 5. *A selectively refined mesh for a test problem on the unit square. Note the refinement around the peak in the source function at* $(0.5, .117)$.

finite-element method for a number of similar problems, the L_∞ norm of the error behaves as $O(n^{-1})$, where n is the number of vertices used in the adaptive mesh. This efficiency results in much smaller memory requirements and large savings in computation times during the solution phase. The relative gain in efficiency can be even more compelling for three-dimensional problems.

The remainder of this paper is organized as follows. In §2, algorithms for refinement of triangulations are reviewed, along with bounds on the quality of those algorithms. The refinement of tetrahedralizations is covered in §3. Unrefinement and refinement of curved boundaries are discussed in §4. Parallel algorithms for refinement in two and three dimensions are given in §5. Finally, a summary and suggestions for future work are made in §6.

2. Refinement in Two Dimensions. In this section we consider the adaptive refinement of triangulations in two dimensions. Prior to discussing the refinement algorithms themselves, we examine the characteristics of a high-quality mesh for finite-element calculations.

For ease in the finite-element formulation and programming, it is desirable to maintain a conforming triangulation during refinement. A mesh is *conforming* if the intersection of any two triangles in the mesh M_k is a line segment connecting two nodal points, a nodal point, or the empty set. In Figure 6 a conforming mesh is shown on the left and a nonconforming mesh on the right.

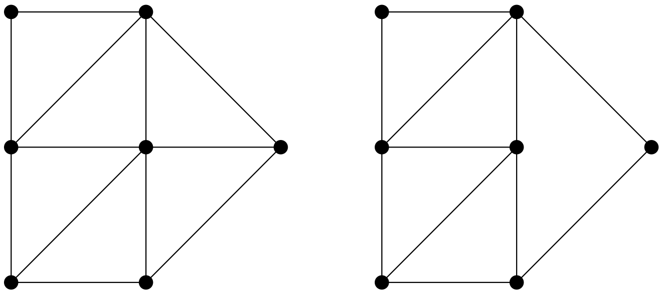

FIG. 6. *The mesh on the left is a conforming mesh. The mesh on the right is nonconforming. Note the midpoint on the left side of the rightmost triangle.*

To ensure the quality of a mesh during refinement, it is desirable that no very large or very small angles are generated. In [1], Babuška and Aziz show that the accuracy of the finite-element approximation degrades as the maximum angle approaches π. Small angles should be avoided because the condition number of the matrices that arise from the finite-element discretization grows as $O(\frac{1}{\theta_{\min}})$, where $\theta_{\min}$ is the minimum angle in the triangulation [13].

Finally, the mesh should be graded or smooth. In other words, the area of neighboring triangles should not differ dramatically; otherwise the finite element approximation may be quite far from the actual solution.

We note that in some applications (in particular, problems with nonisotropic physics) these quality measures may be violated. For example, when discretizing fluid flow in a boundary layer, an optimal triangulation may have a high aspect ratio corresponding to steep velocity gradients in directions normal to a surface. Special mesh generation techniques are required in these cases. We refer the reader to [6] for a discussion of the construction of accurate control volumes for nonisotropic flows.

2.1. Subdividing Triangles. The most obvious means of dividing a triangle to maintain a conforming mesh is to place a nodal point at the centroid of the triangle and connect it to the three existing nodal points. This process creates three new triangles, as shown in Figure 7. Unfortunately, repeated refinement of a region clearly results in angles that go toward 0 and π; ungraded meshes also can result, as illustrated on the right of this figure.

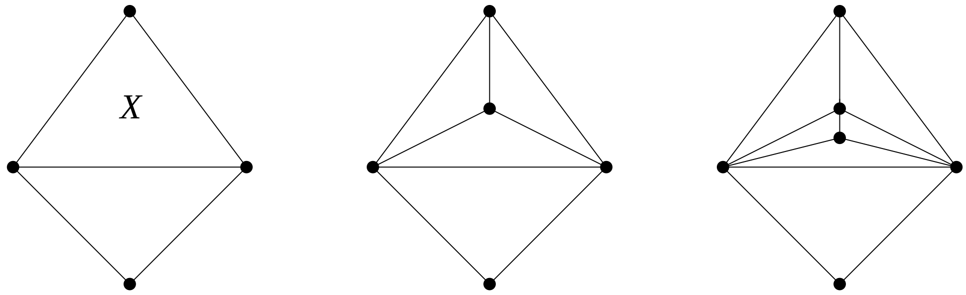

FIG. 7. *Triangle X on the left has been divided into three triangles. Repeated division results in poor angles and lack of mesh smoothness.*

Another means of triangle division is bisection, as shown in Figure 8. Bisection divides the triangle area exactly in half, and the bisected angle is also halved. This approach can result in a nonconforming mesh, as illustrated on the right of Figure 8. However, as we show below, this problem can be solved by propagating the refinement to nonconforming triangles. Moreover, if triangles are bisected only across their longest edge, one can bound the maximum and minimum angles of the resulting triangles independently of the number of times the resulting triangles are bisected. If a triangle and its descendants are repeatedly bisected across their longest edges, the smallest resulting angle is bounded by at worst one-half the smallest angle in the original triangle [33]. A simple corollary is that the largest resulting angle is also bounded away from π. Moreover, the angles of M_k tend to go toward $\frac{\pi}{3}$ as $k \to \infty$ [35].

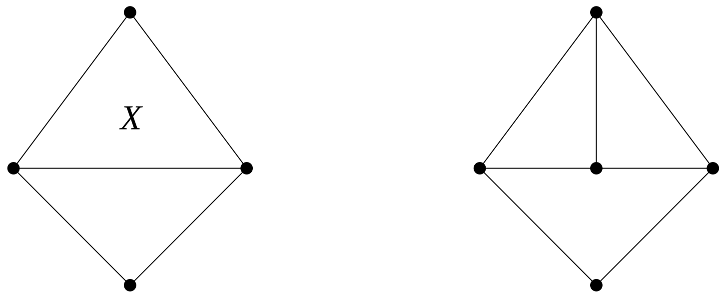

FIG. 8. *Triangle X on the left has been bisected into two triangles. At this stage the resulting triangulation is nonconforming.*

A third means of triangle division is regular refinement [4], as illustrated in Figure 9. Again, this approach can result in a nonconforming mesh, as shown on the right of this figure. A conforming mesh can be obtained by temporarily refining triangles with one nonconforming edge through bisection. This bisecting edge, known as a *green* edge, is removed before the next level of refinement; regular refinement is used on triangles that have two or more nonconforming edges. The four triangles resulting from regular refinement are all similar to the original triangle. Thus, no refined mesh angle can be less than half the smallest initial mesh angle.

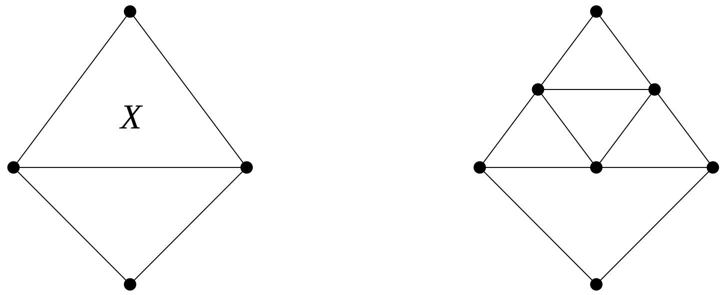

FIG. 9. *Triangle X on the left has been regularly refined into four triangles. At this stage the resulting triangulation is nonconforming.*

2.2. Refinement Algorithms. Rivara has described an effective algorithm for mesh refinement based on bisection in [30]. The algorithm assumes that an initial set of triangles in M_i have been marked for refinement based on error estimates/indicators. As triangles become nonconforming, they are also marked for refinement. The algorithm, given in Figure 10, continues until a conforming mesh, M_{i+1}, has been constructed. Rivara shows that this algorithm will terminate; however, no useful bound exists for L, the number of times the while loop is executed [30]. In Figure 11, we give an example for which L is $O(n)$, where n is the number of triangles in M_i. In practice, however, we have found L to be a small constant independent of n.

Let T_0 be the set of marked triangles
$i = 0$
While $(T_i \neq \emptyset)$ **do**
    Bisect triangles in T_i across their longest edge
    Let T_i be the set of nonconforming triangles
    $i = i + 1$
Enddo

FIG. 10. *The longest-edge bisection algorithm of Rivara*

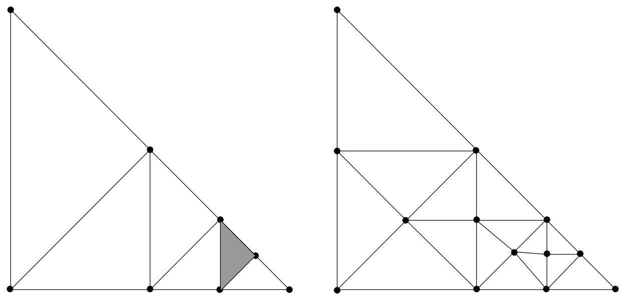

FIG. 11. *A worst-case example of propagation of refinement. On the left, the shaded triangle is marked for refinement; on the right, the resulting conforming mesh after refinement. Note that longest-edge refinement has propagated through every triangle in the mesh but one.*

Rivara has described variants of this algorithm, including one in which simple bisection is combined with bisection across the longest edge to reduce L [30]. By *simple bisection*, we mean bisection across an edge that may not be the longest. In this algorithm, a triangle is first bisected across its longest edge. If either of the two resulting triangles become nonconforming, as the result of bisection of a neighbor, they are bisected across the nonconforming edge (see Figure 12). The algorithm, given in Figure 13, yields the same angles bounds as the longest edge-only algorithm. Note that the algorithm in Figure 13 is for the first refinement step; subsequent steps must assign triangles to V_i and T_i based on whether the triangle resulted from a longest-edge bisection or not.

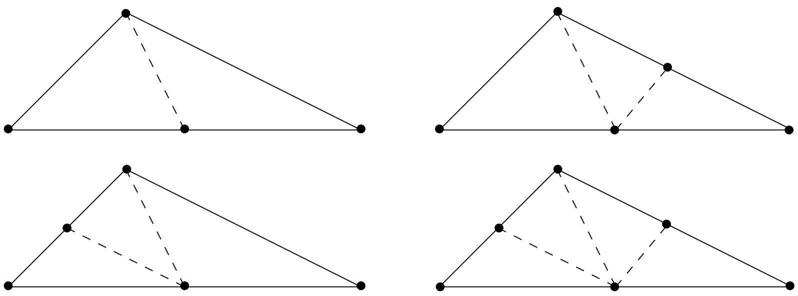

FIG. 12. *The possible children of a triangle for the modified bisection algorithm of Rivara. After the initial longest-edge bisection, the remaining two edges are bisected if they are nonconforming. Note that these edges may not be the longest edges in the bisected triangle.*

A group led by Bank [4] has developed a refinement algorithm based on regular

Let T_0 be the set of marked triangles
  { *T will denote triangles not yet refined* }
$V_0 = \emptyset$ { *V will denote children of refined triangles* }
$i = 0$
While $((T_i \cup V_i) \neq \emptyset)$ **do**
    Bisect triangles in T_i across their longest edge
    Bisect triangles in V_i across a nonconforming edge
    Let V_i be the set of nonconforming triangles embedded in $\cup_{j=0}^{i} T_i$
    Let T_i be the set of all other triangles
    $i = i + 1$
Enddo

FIG. 13. *The modified bisection algorithm of Rivara*

refinement with selected temporary bisections. This algorithm, given in Figure 14, has been used in the software package PLTMG [3]. Triangles initially marked for refinement are refined by using regular refinement. As refinement propagates, any triangle with at least two nonconforming edges is also regularly refined. When only triangles with one or less nonconforming edges remain, the remaining nonconforming edges are bisected. Prior to the next refinement, these bisected triangles are merged. Thus, every triangle in M_{i+1} either is similar to a triangle in the original mesh or is similar to a triangle obtained by the bisection of a triangle in the original mesh. Note that because the bisected triangles are temporary, the refined meshes are not strictly nested.

All bisected triangles are merged
Let T_0 be the set of marked triangles
$i = 0$
While $(T_i \neq \emptyset)$ **do**
    Regularly refine triangles in T_i
    Let T_i be the set of triangles with at least two nonconforming edges
    $i = i + 1$
Enddo
Bisect remaining nonconforming triangles across nonconforming edge

FIG. 14. *The regular refinement algorithm of Bank*

In the modified version of Sewell's algorithm proposed by Mitchell, triangles are always bisected across the edge *opposite* the newest node in the triangles [22]. Triangles are refined only in compatible pairs; a neighboring pair is *compatible* if the edge selected by each triangle is the same (see Figure 15). If a triangle to be refined does not have a compatible neighbor, the neighbor opposite the newest node is recursively refined until a compatible neighbor is created. Because an angle is never divided twice, the angles generated by the algorithm are necessarily bounded away from 0 and π (see Figure 16). Further, the length of the recursive division of a neighbor is bounded [22].

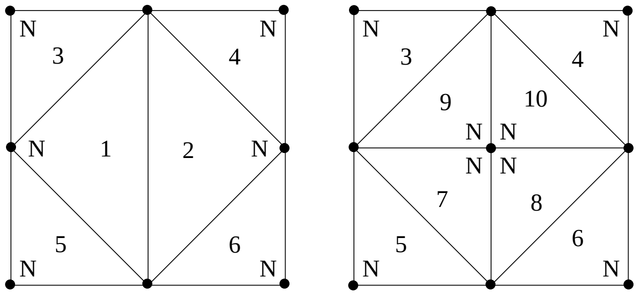

FIG. 15. *An example demonstrating Sewell's newest-node algorithm. In the mesh on the left, only triangles* 1 *and* 2 *may be refined. After they are refined, all four pairs of triangles, e.g.,* 5 *and* 7*, may be refined.*

Let T be the set of marked triangles
Call *New_Refine(t,∅)* for each triangle, t, in T

Procedure *New_Refine(t,d)*
  Let n be the neighbor opposite the newest node in t
  If (t is compatible with n)
    If ($n = d$)
      Return
    Else
      Bisect the pair to form t_1, t_2 and children of n
      Remove t and n (if necessary) from T
      If ($d \neq \emptyset$)
        Call *New_Refine(s,d)*, where s is the triangle in t_1, t_2
          that shares an edge with d
      Endif
      Return
    Endif
  Else
    Call *New_Refine(n,t)*
    Bisect t and its newly compatible neighbor
    Remove t and neighbor (if necessary) from T
  Endif
Endif

FIG. 16. *The modified newest-node refinement algorithm of Mitchell*

Mitchell compared these refinement algorithms in several numerical experiments [22]. He found that all these methods performed well. In general, the regular refinement algorithm was preferable when the initial triangles are nearly equilateral, whereas bisection was slightly more efficient otherwise. He also found that the longest-edge bisection algorithm and Rivara's variant that uses simple bisection perform nearly identically. In most of the test cases presented, the newest-node algorithm and the longest-edge bisection algorithms were equivalent. In the other test cases, the longest-edge bisection algorithm slightly outperformed the newest-node algorithm. Mitchell notes that the newest-node algorithm does not require any computations to select an edge for division. Further, he notes that the regular refinement algorithm is the only one in which M_{i+1} is not likely to be nested in M_i, because of the merger of bisected triangles prior to refinement.

3. Refinement in Three Dimensions. In this section we review algorithms for the adaptive refinement of three-dimensional tetrahedral meshes. As we observed in the two-dimensional case, a conforming mesh is required for use in finite-element discretization. A tetrahedralization is *conforming* if the intersection of any two tetrahedra in the the mesh is a triangle, a line segment connecting two nodal points, a nodal point, or the empty set. In addition, the mesh should be graded in the same sense as for the two-dimensional case; that is, the volumes of neighboring tetrahedra should not differ dramatically.

Developing a refinement scheme with guaranteed angle bounds is more problematic in three dimensions than in two dimensions. We first illustrate that the two-dimensional measure of mesh quality, the angles of the triangulation formed by the faces of the tetrahedra, is not an adequate mesh quality measure. Consider a tetrahedra with the following coordinates:

$$\begin{aligned} v_0 &= (0,0,0), \\ v_1 &= (0,1,0), \\ v_2 &= (0.5,0.5,\epsilon), \\ v_3 &= (0.5,-0.5,\epsilon). \end{aligned} \tag{3.2}$$

The face angles of this tetrahedron are good in the two-dimensional sense; they are bounded away from 0 and π independent of ϵ. However, as $\epsilon \to 0$, the interior solid angles and volume of the tetrahedron go to zero.

Many measures of tetrahedron quality have been proposed, most of which have a maximum value for an equilateral tetrahedron and a minimum value for a degenerate tetrahedron. One commonly used measure is the minimum solid angle. The solid angle at a vertex is defined to be the surface area of the spherical triangle on a unit sphere obtained by placing the vertex at the center of the sphere and projecting the three edges originating at that vertex onto the sphere. Another measure is the aspect ratio of the tetrahedron: the ratio of the sphere contained within the tetrahedron to the sphere containing the tetrahedron, divided by three. A third measure, the mean ratio ν, is defined in [19]; this measure is based on the eigenvalues of the transformation of a given tetrahedron to a regular tetrahedron. Joe and Liu show in [20] that each of these measures is equivalent in some sense.

Before describing specific refinement algorithms, we review the process of bisection in three dimensions. To bisect a tetrahedra, t, composed of vertices (v_0, v_1, v_2, v_3) across the edge (v_0, v_1), we first create the new vertex v_{01} at the midpoint of (v_0, v_1). We bisect the tetrahedron with the plane defined by the new vertex, v_{01}, and the vertices v_2 and v_3. This bisection creates two new tetrahedra, (v_0, v_{01}, v_2, v_3) and (v_{01}, v_1, v_2, v_3), as shown in Figure 17. This operation bisects faces (v_0, v_1, v_2) and (v_0, v_1, v_3) of t, which (unless these faces are on a domain boundary) are each shared with adjacent tetrahedra. Thus, for correctness, any bisection algorithm must uniquely define how a given face is to be bisected. Two tetrahedra that share a face must agree on how it is to be bisected; otherwise a nonconforming mesh will be constructed.

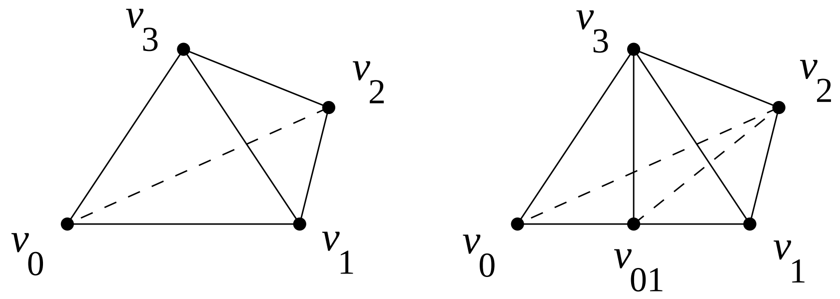

FIG. 17. *The tetrahedron on the left is bisected to form two new tetrahedra*

We note that a single bisection of tetrahedron can arbitrarily degrade the quality of a tetrahedron, whereas in two dimensions the minimum angle of a triangle is decreased by no more than a factor of two. Consider the tetrahedron with the following coordinates:

$$\begin{aligned} v_0 &= (0, \epsilon, 0), \\ v_1 &= (0, -\epsilon, 0), \\ v_2 &= (1, 0, \epsilon), \\ v_3 &= (1, 0, -\epsilon). \end{aligned} \tag{3.3}$$

Bisection of the longest edge of this tetrahedron creates a new tetrahedron with a minimum solid angle of order $O(1/\epsilon)$ times that of the initial minimum solid angle.

Rivara and Levin have suggested an extension of the longest-edge Rivara refinement algorithm to tetrahedra [32]. This algorithm corresponds to the two-dimensional algorithm for longest-edge bisection given in §2. By splitting the longest edge in the tetrahedron, we have that the longest edge on two shared faces is also split (ties can be broken by vertex labels). Thus, the bisection of the faces of shared tetrahedra is uniquely defined. Unfortunately, it is an open question whether the tetrahedra resulting from this algorithm have bounded solid angles (or any other measure) [26]. Rivara and Levin provide experimental results showing that the minimum angle in the tetrahedralization tends to converge to a fixed point or at least does not drop below a fixed point [32]. These results suggest that, at least in practice, this algorithm would not result in tetrahedra that continue to degenerate in quality as a mesh is refined.

The major difficulty in proving that the Rivara and Levin algorithm does not continue to degrade mesh quality is that it is not known whether the algorithm generates a finite number of similar tetrahedron. However, a bisection algorithm first introduced by Bänsch [5] does generate a finite number of similar tetrahedra. Before describing the algorithm in detail, we sketch the proof given by Liu and Joe [19] which motivates the algorithm.

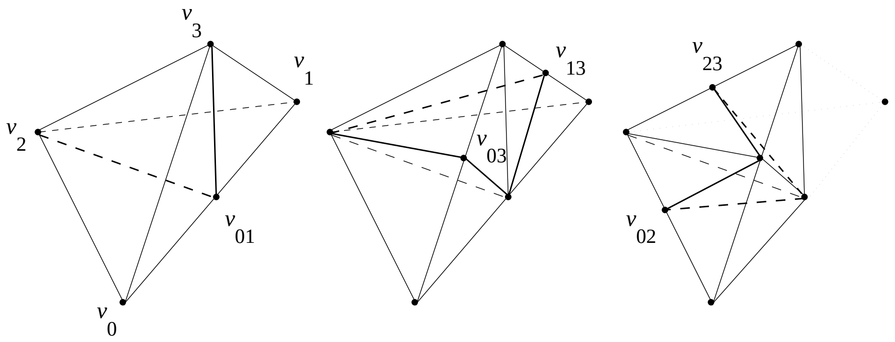

FIG. 18. *The first three levels of longest-edge bisection of the canonical tetrahedron. Note that the tetrahedra generated at each level are similar. For the final level of refinement we show only the four tetrahedra obtained from* (v_0, v_1, v_{12}, v_3). *Four similar tetrahedra are obtained from* (v_0, v_{12}, v_2, v_3).

The key observation is that there exists an affine transformation that maps any tetrahedron to a canonical tetrahedron for which longest-edge bisection generates only a finite number of similarity classes of tetrahedra. Consider the canonical tetrahedron with coordinates:

$$\begin{aligned} v_0 &= (-1, 0, 0), \\ v_1 &= (1, 0, 0), \\ v_2 &= (0, 1/\sqrt{2}, 0), \\ v_3 &= (0, 0, 1). \end{aligned} \tag{3.4}$$

In Figure 18 we illustrate the first three levels of longest-edge bisection of the canonical tetrahedron. It can be shown that all the tetrahedra generated at each level of refinement are similar *and* that the eight tetrahedra generated after three levels of refinement are similar to the initial tetrahedron.

However, the inverse of the affine mapping does not map similar tetrahedra in the canonical space into similar tetrahedra in the original space unless the similar tetrahedra have the same orientation. Thus, Liu and Joe define two tetrahedra to be in the same *similarity class* in the canonical space if they can be mapped onto each other by a combination of scaling and translation. Liu and Joe show by direct calculation that a finite number of similarity classes of tetrahedra are generated by longest-edge bisection of the canonical tetrahedron. Therefore, if this same bisection order is used in the original space, only a finite number of similar tetrahedra are generated.

Both Bänsch [5] and Liu and Joe [21] give equivalent algorithms that generate this bisection order. The algorithm proposed by Bänsch is given in Figure 19. In Bänsch's algorithm each face in every tetrahedra in the mesh has an edge marked as a "refinement edge" such that (a) the marking on a face is consistent between the two tetrahedra that share it, and (b) at least one edge in a tetrahedron is marked by two faces (such a shared edge is called a "global refinement edge"). Bänsch notes that this requirement is initially accomplished by marking the longest edge in every face (ties can be broken in any consistent manner; for example, by vertex or edge label order). With this method there is a unique global refinement edge for each tetrahedron.

Mark the refinement edge of every face in M_k
Let T_0 be the set of marked tetrahedra
$i = 0$
While $(T_i \neq \emptyset)$ **do**
    Bisect tetrahedra in T_i across their global refinement edge
    In each new tetrahedron, mark the "old" edges in each bisected face
        as refinement edges
    Mark the refinement edges for the resulting new faces
    Let T_i be the set of nonconforming tetrahedra
    $i = i + 1$
Enddo

FIG. 19. *The tetrahedra refinement algorithm of Bänsch*

In the Bänsch algorithm given in Figure 19, tetrahedra marked for refinement always are bisected across the global refinement edge. The unbisected edges in the two bisected faces in each bisected tetrahedron are marked as refinement edges. This bisection and marking process is shown in Figure 20, where an x represents the marked side for a face and a * denotes that the marked side is unspecified in a face. In these figures we have unfolded the faces of the tetrahedron to show the marking of the face edges, as done in [5]. The bisection creates a new face shared by the two new tetrahedra; the key to the algorithm is how to choose the refinement edges for these new faces.

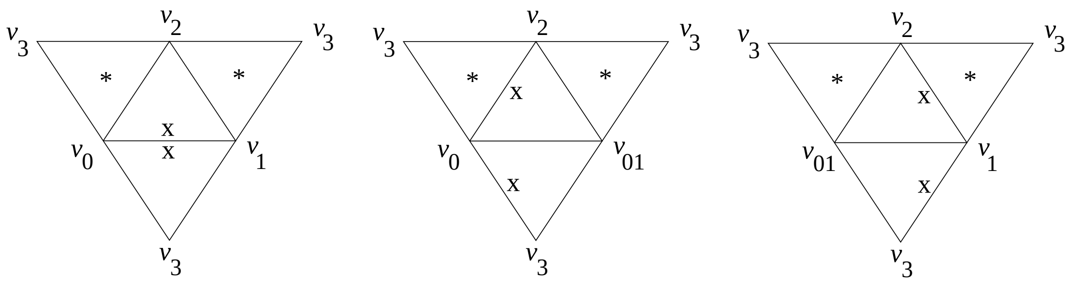

FIG. 20. *The tetrahedron on the left is bisected along edge (v_0, v_1) to form the two tetrahedra on the right.*

To accomplish the choice of refinement edges for new faces, Bänsch classifies tetrahedra as either red or black; see Figures 21 and 22 for the definition of each type. For each of the types in these figures, edge (v_0, v_1) is the global refinement edge, and the new face is (v_{01}, v_2, v_3). Let t be the bisected tetrahedron and $\hat{t}$ be the parent, if any, of t. The refinement edge for the new face is chosen according to the following rules:

- if t is red, then choose edge (v_2, v_3);
- if t is black and $\hat{t}$ is red, then choose edge (v_2, v_3);
- if $\hat{t}$ is black and t is either black case 1 or black case 2, choose edges (v_2, v_{01}) and (v_{01}, v_3), respectively; and
- if t is black and $\hat{t}$ does not exist, choose edge (v_2, v_3).

In every case, the new tetrahedra are either black or red. Note that in the initial mesh, it is possible for some tetrahedra to be neither black nor red. In this case edge (v_2, v_3) is chosen, and the resulting tetrahedra are either red or black.

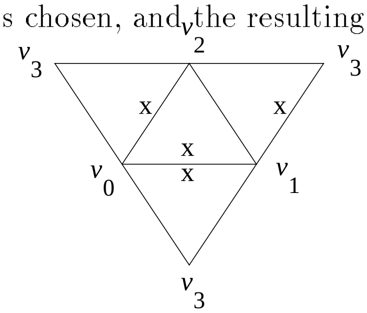

FIG. 21. *Tetrahedra classified as red; case 1 is on the left, and case 2 is on the right.*

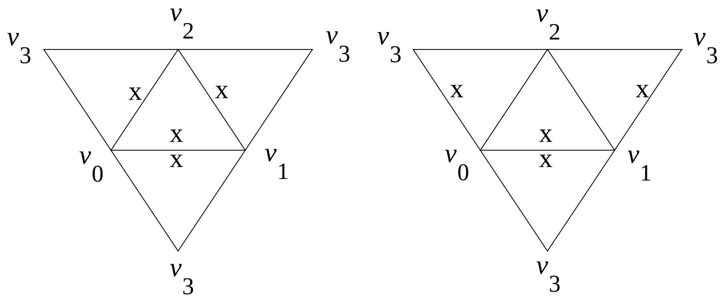

FIG. 22. *Tetrahedra classified as black; case 1 is on the left, and case 2 is on the right.*

Detailed examination of this algorithm shows that tetrahedra are bisected in the same manner as the canonical tetrahedron bisection depicted in Figure 18. Bänsch proves that this algorithm results in a conforming mesh in a finite number of steps. He further claims that only a finite number of similar tetrahedra are generated in the refined mesh, although the proof is incomplete. A complete proof of this result is given by Liu and Joe [19]. The conclusion of this result is that the angles of all resulting tetrahedra are bounded independently of the number of levels of refinement [5].

Liu and Joe have a more detailed description and analysis of this algorithm from a different perspective and prove additional properties of the algorithm [19] [21]. They show that the shape measure, ν, of any child, t_i, formed with this algorithm from a tetrahedron t is bounded independently of the number of levels of refinement as

$$\nu(t_i) \geq c\nu(t) \ , \tag{3.5}$$

where $c \approx 0.0974$. In addition, they show that two tetrahedra sharing a face in a mesh refined with this algorithm can differ by at most two levels of refinement. They further show that any two tetrahedra sharing an edge in such a mesh will differ by at most four levels of refinement. This fact implies that, in some sense, the resulting mesh should be smooth [21].

Liu and Joe experimentally compared meshes formed by longest-edge bisection as in [32] with meshes obtained from their algorithm [21]. They found that the longest-edge bisection algorithm often results in meshes with many more tetrahedra than generated by their algorithm.

4. Coarsening and Refinement on Curved Surfaces. It is often useful, particularly in time-dependent computations, to unrefine regions of the mesh that were refined in previous steps of the computation. For example, if a shock front is moving across the domain, one might wish to refine around the wave and unrefine in its wake. Reversing the refinement algorithms presented in the preceding sections has been considered [5] [31].

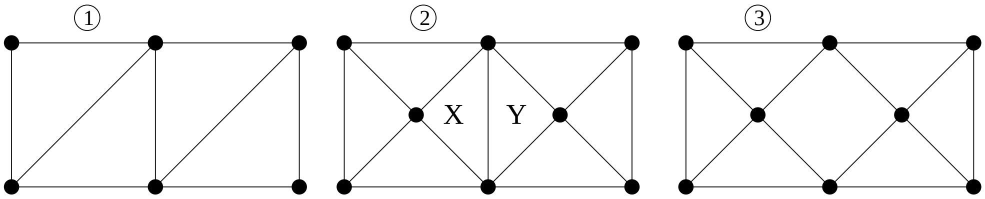

FIG. 23. *The triangulation in step 1 is refined to form the triangulation in step 2. However the unrefinement suggested in step 3 does not form a legal triangulation.*

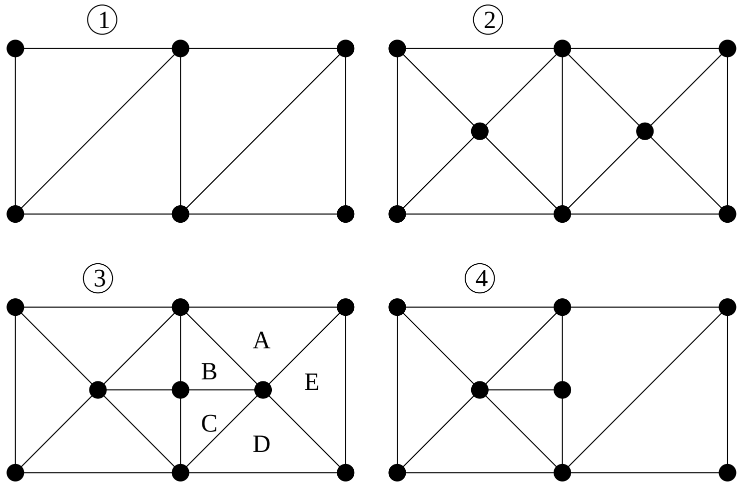

FIG. 24. *The triangulation in step 1 is refined to form the triangulation in step 2 and then further refined to form the triangulation in step 3. However, the unrefinement suggested in step 4 does not form a conforming triangulation.*

Reversing the refinement cannot be done arbitrarily, however. One must exactly reverse the process because triangles or tetrahedra cannot be arbitrarily merged and still be guaranteed to form a conforming triangulation. For example, consider Figure 23 where triangle X and Y are merged and a quadrilateral is formed rather than a triangle. In Figure 24 note that the vertex in the center of triangles A, B, C, D, E is removed and the triangles are merged, yet a conforming triangulation has not been formed. Rivara describes a labeling algorithm that correctly allows for the unrefinement of a adaptive mesh created with the bisection algorithm [31].

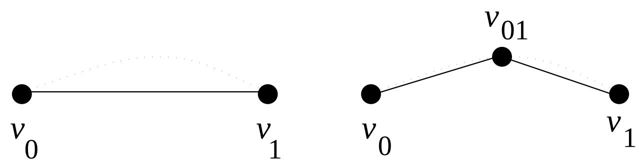

FIG. 25. *The curved domain on the left is approximated by the line segment connecting v_0 and v_1. When the line the segment is bisected, vertex v_{01} is the new midpoint.*

In practice, these refinement algorithms can be used not only on polygonal regions, but also on regions with curved boundaries and surfaces. One can associate line segments of the initial triangulation with portions of the curved boundaries. Then, for example, if line segment (v_0, v_1) is to be refined and it lies on the curved boundary, rather than using average of the positions of the points as the midpoint, we would use the point midway between v_0 and v_1 that lies on the curved surface. This process is depicted in Figure 25. Care must be taken, however, since several potential pitfalls exist. First, the angle bounds by the refinement algorithms for polygonal regions do not hold for these curved boundaries. Second, this approach can result in illegal triangulations. Below, we give two examples of such problems. However, given that the initial triangulation is a good approximation of the domain being discretized, these problems usually can be avoided and good results obtained.

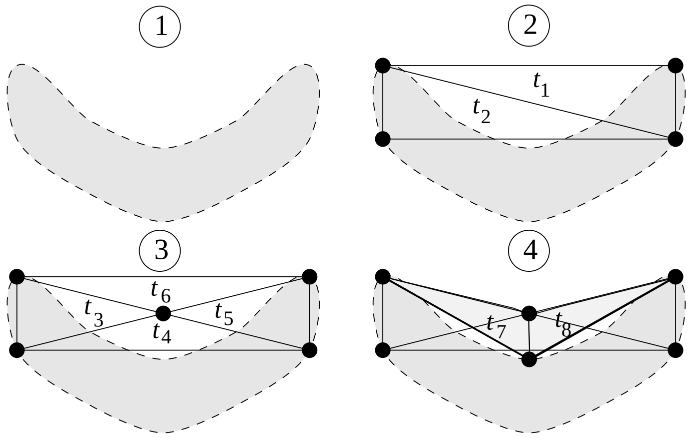

FIG. 26. *Triangles of negative area resulting from the refinement of a concave boundary. Bisection of the longest edge of triangle t_6 introduces two children, triangles t_7 and t_8, that are inverted because of the creation of a new vertex along the concave boundary.*

In Figure 26 we give an example of how a concave boundary can result in triangles with negative area. The domain to be discretized is shown in step 1. In

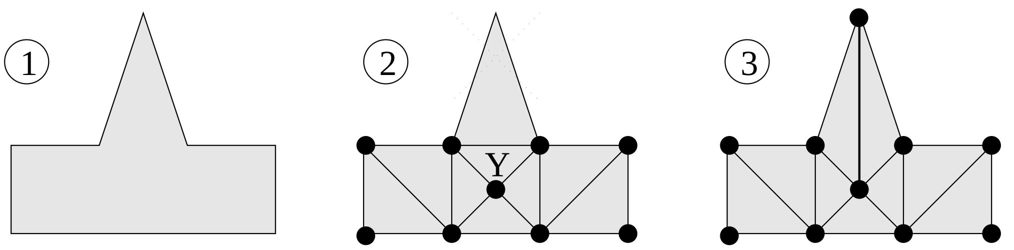

FIG. 27. *Triangles with potentially very large angles resulting from the refinement of a convex boundary*

step 2, an initial triangulation is provided by some method of grid generation. In step 3 this triangulation is refined further, maintaining a legal triangulation. Note that the upper edge of t_6 corresponds to the concave boundary of the domain. In step 4, triangle t_6 is refined with the midpoint of the refined side defined by the midpoint of the concave boundary. This refinement results in t_6 being refined into two triangles of negative area.

In Figure 27 we give an example of how a convex boundary can result in triangles with large angles. The domain to be discretized is shown in step 1. In step 2, an initial triangulation is provided; note that the triangular region on the top, denoted by the dotted X, is *not* covered by the initial triangulation. We see that the refinement of triangle Y results in a potentially large angle in the two child triangles in step 3.

5. Parallel Algorithms for Refinement. Many scientific computations are demanding both in terms of processor time and memory. The use of massively parallel computing can make previously intractable computations possible. In a parallel environment, however, aspects of the computation are more interrelated than in a sequential setting. For example, the mesh partitioning must be dynamic to ensure correct load balancing when refining and then solving the resulting linear systems. We do not discuss these other problem aspects here; however, they must be considered in the same context of the refinement algorithm. We note that much work has been done on the parallel solution of linear systems, using both iterative methods [18] and direct methods [14] [29].

In this section, we review recent work on algorithms for the parallel refinement of unstructured meshes. We limit our discussion to triangulations; however, both of the methods described apply to tetrahedra as well. In addition, we describe the methods primarily with respect to bisection, but these approaches apply to all the local refinement algorithms discussed in this paper.

Two primary problems need to be addressed to maintain a correct parallel mesh data structure when constructing a local refinement algorithm. First, when two triangles on different processors share an edge marked for refinement, they must avoid (or undo) creation of a redundant vertex. This problem is illustrated in Figure 28. Second, triangles in the mesh must maintain correct neighbor information. When two adjacent triangles on different processors are simultaneously

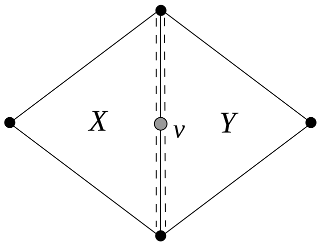

FIG. 28. *Neighboring triangles X and Y are assigned to different processors. They share an edge marked by both for refinement. A parallel refinement algorithm must ensure that a redundant copy of the new vertex v on this edge is not created.*

refined, one must ensure that the neighborhood information is maintained correctly. This problem is illustrated in Figure 29.

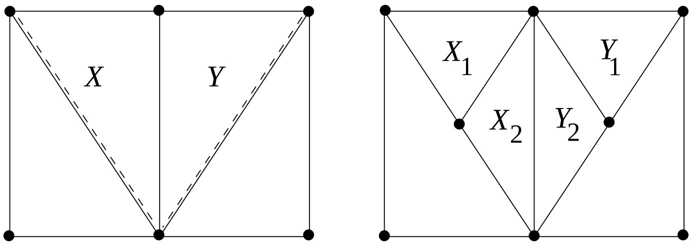

FIG. 29. *On the left, triangles X and Y are marked for refinement and assigned to different processors. After refinement, we get the triangles on the right, where the new triangle X_2 is now neighboring the new triangle Y_2. The parallel refinement algorithm must ensure that this neighbor information is maintained correctly during refinement.*

Williams [37] [39] has given an approach for parallel mesh refinement and has implemented it in the parallel software package DIME. To address the two questions above, he maintains a parallel *voxel* database, using vertex coordinate information to help resolve point identity and triangle neighborhood information. He does not, however, present a running time analysis of the algorithm.

An alternative approach has been taken by Jones and Plassmann [17]. In this approach, independent sets of triangles are maintained to ensure that neighboring triangles are *never* simultaneously refined on different processors. Recall that a set of triangles is said to be independent if they do not share an edge. The independent sets can be efficiently chosen in parallel by a randomization strategy. Random numbers are assigned to every triangle marked for refinement; a triangle is in the independent set if its random number is larger than any neighboring marked triangles. The expected running time of this algorithm under the PRAM computational model is $O(\frac{\log N}{\log\log N}) + O(L)$, where N is the number of triangles marked for refinement and L is the number of propagation steps required for the refinement algorithm (see §2). Recall that the PRAM computational model assumes that processors communicate through a common shared memory; the above analysis assumes that we have as many processors as we have triangles.

This bound implies that, in practice, the running time of the algorithm is a slowing growing function of the number of processors. This fact is verified experimentally in [17] where scalable performance is demonstrated by the algorithm on the Intel DELTA parallel computer. In [16] the performance of the adaptive refinement algorithm is examined in the context of solving problems involving higher-order shell elements. The computational cost of the algorithm is shown to be small relative to the time required for mesh partitioning, matrix assembly, and linear system solution.

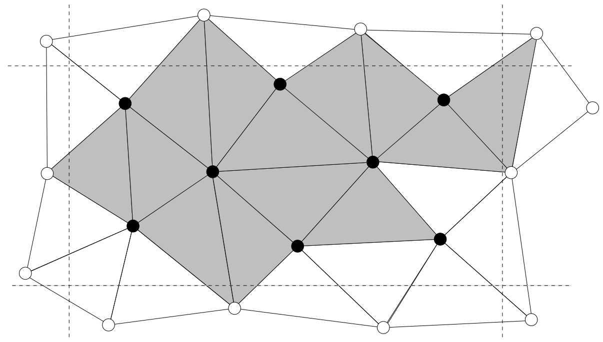

FIG. 30. *An illustration of the set of vertices and elements maintained by one processor. The set of vertices assigned to processor is shown as the set of filled vertices, and the set of shaded triangles are the assigned elements. The union of the set of shaded and unshaded triangles is the set of elements required to assemble the portion of the linear system corresponding to the assigned vertex unknowns. The dashed lines demonstrate one way to partition the vertices; in this case, geometric cuts were used.*

In Figure 30 we illustrate the information that must be maintained by each processor for a parallel implementation of the finite-element method. The mesh vertices and elements must be partitioned among the processors; we assume that the processor assignments are unique. In the figure, vertices that are assigned to a processor are shown as solid, and triangles assigned to the same processor are shown as shaded. To assemble the matrix row corresponding to the unknown at a mesh vertex, a processor must have information from all elements sharing that vertex. Elements in this set not assigned to the processor are shown as unshaded, and additional vertices in these elements are shown as unfilled circles.

To assemble the linear system in parallel, one can either (1) evaluate only elements assigned to a processor and communicate element stiffness matrices for row assembly, or (2) keep all the element information necessary to assemble rows corresponding to the assigned vertices and avoid this communication. The former approach requires interprocessor communication; the latter approach requires no communication, but redundant element evaluation and more processor memory. The latter approach was used for the results given in [16].

Note that the efficient parallel performance of the methods discussed in this section rely on good mesh partitions. The initial mesh must be partitioned to equalize the load on each processor and to minimize the number of triangles that share edges but are assigned to different processors. An additional problem is that as the computation proceeds, some processors are more likely than other processors to refine triangles. Thus, the mesh must be dynamically rebalanced to maintain the same load distribution as the initial partitioning, while minimizing the interprocessor communication required to move triangles between processors. Many effective partitioning heuristics have been proposed in the literature [15]; Williams compares several partitioning methods [38]. The choice of partitioning methods strongly depends on the sparse solver used. If a direct sparse factorization is used, minimization of the separator size is of primary importance, and spectral methods produce excellent results [27].

Finally, we note that interesting work has been done on the parallelization of alternative adaptive schemes to those discussed in this paper. Devine et al. has considered a parallel p-refinement approach on regular grids [11], and Bell et al. [7] have considered parallel adaptive methods for explicit finite-volume schemes on nested orthogonal meshes. Parallel hp-refinement finite-element methods have been considered by Oden and Patra [25].

6. Summary and Concluding Remarks. Adaptive refinement of finite-element meshes can significantly improve the computational efficiency of scientific and engineering calculations involving solutions with large gradients, complicated geometries, and other special problem characteristics. In this paper we have reviewed a number of algorithms that generate nested, or nearly nested, adaptive finite-element meshes. These algorithm ensure the generation of meshes whose quality is within a constant factor to the initial mesh quality in two-dimensions. In three dimensions, the theoretical bounds are not as good, and a number of open problems remain. However, good experimental results have been obtained for these algorithms [21], [32].

Parallel algorithms for adaptive refinement are encouraging. However, a parallel implementation requires consideration of other aspects of the problems solution, such as the partitioning of elements to processors and linear system solution. Preliminary results indicate that these problems can be solved effectively on large-scale parallel machines. Nevertheless, much more effort is required for the development of portable and efficient software to solve these problems.

Acknowledgment. We thank Lori Freitag and Carl Ollivier-Gooch for a number of helpful suggestions that greatly improved the presentation of this paper.

REFERENCES

[1] I. Babuška and A. K. Aziz, *On the angle condition in the finite element method*, SIAM Journal of Numerical Analysis, 13 (1976), pp. 214–226.

[2] I. Babuška and W. C. Rheinboldt, *Error estimates for adaptive finite element computations*, SIAM Journal of Numerical Analysis, 15 (1978), pp. 736–754.

[3] R. E. Bank, *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations. Users' Guide 6.0*, SIAM Publications, Philadelphia, Penn., 1990.

[4] R. E. Bank, A. H. Sherman, and A. Weiser, *Refinement algorithms and data structures for regular local mesh refinement*, in Scientific Computing, R. Stepleman et al., ed., IMACS/North-Holland Publishing Company, Amsterdam, 1983, pp. 3–17.

[5] E. Bänsch, *Local mesh refinement in 2 and 3 dimensions*, Impact of Computing in Science and Engineering, 3 (1991), pp. 181–191.

[6] T. Barth, *Aspects of unstructured grids and finite-volume solvers for the Euler and Navier-Stokes equations.* Von Karman Institute for Fluid Dynamics, Lecture Series 1994-05, NASA Ames Research Center, March 1994.

[7] J. Bell, M. Berger, J. Saltzman, and M. Welcome, *Three-dimensional adaptive mesh refinement for hyperbolic conservation laws*, SIAM Journal on Scientific Computing, 15 (1994), pp. 127–138.

[8] M. Bern and D. Eppstein, *Mesh generation and optimal triangulations*, in Computing in Euclidean Geometry, 2nd edition, D.-Z. Du and K. F. Hwang, ed., World Scientific, Singapore, 1995, pp. 47–123.

[9] L. Chew, *Guaranteed-quality triangular meshes*, Tech. Rep. TR-89–983, Computer Science Department, Cornell University, 1989.

[10] L. Demkowicz, J. Oden, W. Rachowicz, and O. Hardy, *Toward a universal h-p adaptive finite element strategy, part 1. Constrained approximation and data structures*, Computer Methods in Applied Mechanics and Engineering, 77 (1989), pp. 79–112.

[11] K. D. Devine, J. E. Flaherty, S. R. Wheat, and A. B. Maccabe, *A massively parallel adaptive finite element method with dynamic load balancing*, Tech. Rep. SAND 93–0936C, Sandia National Labs; Albuquerque, N.M., 1993.

[12] J. E. Flaherty, P. J. Paslow, M. S. Shephard, and J. D. Vasilakis, *Adaptive Methods for Partial Differential Equations*, Society for Industrial and Applied Mathematics, Philadelphia, 1989.

[13] I. Fried, *Condition of finite element matrices generated from nonuniform meshes*, AIAA Journal, 10 (1972), pp. 219–221.

[14] M. T. Heath and P. Raghavan, *Distributed solution of sparse linear systems*, Tech. Rep. UIUCDCS-R-93-1793, University of Illinois, Feb. 1993.

[15] B. Hendrickson and R. Leland, *The Chaco user's guide: Version 2.0*, Sandia Report SAND94–2692, Sandia National Laboratories, Albuquerque, N.M., June 1995.

[16] M. T. Jones and P. E. Plassmann, *Computational results for parallel unstructured mesh computations*, Computing Systems in Engineering, 5 (1994), pp. 297–309.

[17] ——, *Parallel algorithms for adaptive mesh refinement*, Preprint MCS-P421-0394, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., 1994.

[18] ——, *Scalable iterative solution of sparse linear systems*, Parallel Computing, 20 (1994), pp. 753–773.

[19] A. Liu and B. Joe, *On the shape of tetrahedra from bisection*, Mathematics of Computation, 63 (1994), pp. 141–154.

[20] ——, *Relationship between tetrahedron shape measures*, BIT, 34 (1994), pp. 268–287.

[21] ——, *Quality local refinement of tetrahedral meshes based on bisection*, SIAM Journal on Scientific Computing, 16 (1995), pp. 1269–1291.

[22] W. F. Mitchell, *A comparison of adaptive refinement techniques for elliptic problems*, ACM Transactions on Mathematical Software, 15 (1989), pp. 326–347.

[23] ——, *Adaptive refinement for arbitrary finite element spaces with hierarchical bases*, J. Comp. Appl. Math., 36 (1991), pp. 65–78.

[24] J. Oden, L. Demkowicz, W. Rachowicz, and T. Westermann, *Toward a universal h-p adaptive finite element strategy, part 2. A posterior error estimation*, Computer Methods in Applied Mechanics and Engineering, 77 (1989), pp. 113–180.

[25] J. T. Oden and A. Patra, *A parallel adaptive strategy for hp finite element computations*, Tech. Rep. 94-01, TICAM, University of Texas; Austin, Texas., May 1994.

[26] J. O'Rourke, *Computational geometry column 23*, Internat. J. Comput. Geom. Appl., 4 (1994), pp. 239–242. Also in SIGACT News 25:3 (1994), 24–27.

[27] A. Pothen, H. Simon, and K.-P. Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM Journal on Matrix Analysis, 11 (1990), pp. 430–452.

[28] W. Rachowicz, J. Oden, and L. Demkowicz, *Toward a universal h-p adaptive finite element strategy, part 3. Design of h-p meshes*, Computer Methods in Applied Mechanics and Engineering, 77 (1989), pp. 181–212.

[29] P. Raghavan, *Distributed sparse gaussian elimination and orthogonal factorization*, SIAM Journal on Scientific Computing, 16 (1995), pp. 1462–1477.

[30] M.-C. Rivara, *Mesh refinement processes based on the generalized bisection of simplices*, SIAM Journal of Numerical Analysis, 21 (1984), pp. 604–613.

[31] ——, *Selective refinement/derefinement algorithms for sequences of nested triangulations*, International Journal for Numerical Methods in Engineering, 28 (1989), pp. 2889–2906.

[32] M.-C. Rivara and C. Levin, *A 3-d refinement algorithm suitable for adaptive and multi-grid techniques*, Communications in Applied Numerical Methods, 8 (1992), pp. 281–290.

[33] I. G. Rosenberg and F. Stenger, *A lower bound on the angles of triangles constructed by bisecting the longest side*, Mathematics of Computation, 29 (1975), pp. 390–395.

[34] J. Ruppert, *A new and simple algorithm for quality 2-dimensional mesh generation*, in Proc. 4th ACM-SIAM Symp. on Disc. Algorithms, 1993, pp. 83–92.

[35] M. Stynes, *On faster convergence of the bisection method for all triangles*, Mathematics of Computation, 35 (1980), pp. 1195–1201.

[36] N. Weatherill, O. Hassan, D. Marcum, and M. Marchant, *Grid generation by the delaunay triangulation.* Von Karman Institute for Fluid Dynamics, 1993-1994 Lecture Series, NASA Ames Research Center, January 1994.

[37] R. Williams, *DIME: Distributed Irregular Mesh Environment*, California Institute of Technology, 1990.

[38] ——, *Performance of dynamic load balancing algorithms for unstructured mesh calculations*, Concurrency: Practice and Experience, 3 (1991), pp. 457–481.

[39] ——, *A dynamic solution-adaptive unstructured parallel solver*, Report CCSF-21-92, Caltech Concurrent Supercomputing Facilities, California Institute of Technology, Pasadena, Calif., 1992.