Refinement/Derefinement Procedures in 2-D

Ramamoorthy Ravichandran
Mark S. Shephard

Contents

Nomenclature [11]

G	refers to the geometric model, or, when used as left subscript, to indicate one or more entities associated with the geometric model
M	refers to the mesh, or, when used as left subscript, to indicate one or more mesh entities
${}_{\Upsilon}T$	set of all topological entities associated with model Υ, $\Upsilon = G, M$
${}_{\Upsilon}T_i^d$	topological entity i from model Υ of dimension d, $d = 0$ is a vertex which represents a point in space, $d = 1$ is an edge which represents a 1D locus of points, $d = 2$ is a face which represents a 2D locus of points, $d = 3$ is a region which represents a 3D locus of points
$\partial({}_{\Upsilon}T_i^d)$	boundary of topological entity ${}_{\Upsilon}T_i^d$, $\Upsilon = G, M$
$\overline{{}_{\Upsilon}T_i^d}$	closure of topological entity defined as $({}_{\Upsilon}T_i^d \cup \partial({}_{\Upsilon}T_i^d))$, $\Upsilon = G, M$
$\sqsubset$	classification symbol used to indicate the association of one or more entities from one model, typically M with a higher model, typically G

1 Introduction

An adaptive finite-element method can be defined as a feedback process where a domain is analyzed with a given mesh and the effectiveness of the approximation of the domain is measured though an error indicator. In this context, the selective and local refinement/derefinement of the mesh throughout

the computation arise as a natural feature. This has induced a change in the static classical grid generation approach, in which one fixed grid with adequate refinement is assumed. This chapter surveys some of the available techniques for the refinement and derefinement of the meshes in 2–D and discusses the edge based refinement and derefinement technique which has been implemented.

2 Existing algorithms for refinement/derefinement in 2D

2.1 Refinement

Refinement methods in 2–D can be broadly categorized into three different groups:

1. Using subdivision patters [2].
2. Bisection algorithms [9], [10].
3. Insertion of points into the mesh in Delaunay context [10].

The main requirement for refinement algorithm is that the resulting triangulation should be conforming, (i.e) in 2–D the intersection of two non-disjoint is either a common vertex or a common edge (see Figure 1).

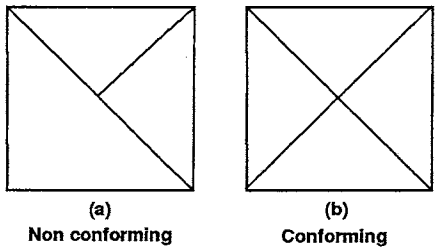


Figure 1. conforming and non conforming triangulations in 2D

There are many criteria which are used during the refinement process, for example [2] and [9] use the following criteria:

1. The size of the smallest angle of any triangle should be bounded away from 0.
2. The transition between large and small triangles in the grid should be "smooth".

3. The triangulation is nested in such a way that each refined triangle is embedded within one of the parent triangle.

2.1.1 Subdivision patterns In the subdivision pattern the parent triangle is subdivided into child triangles by using subdivision patterns or templates. In this context two different patterns are commonly used [2] for subdivision:

1. Regular subdivision (1:4), see Figure 2(a)
2. "Green" subdivision (1:2), see Figure 2(b)

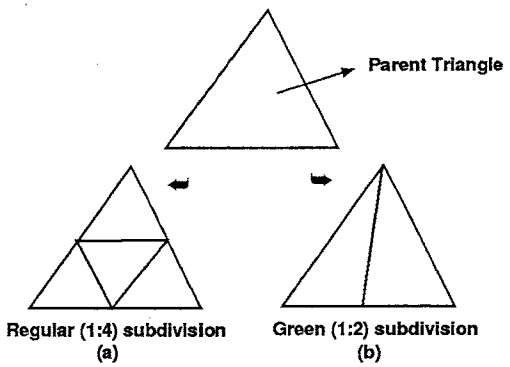


Figure 2. Regular and Green subdivision patterns

Bank and Sherman [2], use regular subdivision to divide the parent triangle into four smaller triangles by joining the mid points of the edges, see Figure 2(a). Each of the four new triangles is similar to the parent triangle, hence regular subdivision never reduces the size of the interior angles (satisfies criteria 1). Green subdivision (Figure 2(b)) can reduce the size of the interior angles and repeated use can violate criteria 1, hence they are used only in the clean up phase to main conformity, see Figure 3(a). During the clean up stage, to satisfy criteria 1 and 2, [2] uses regular subdivision to refine a triangle whenever two of its neighbors have been divided once or one neighbor has been divided twice, see Figure 3(b). Adequate history is also maintained whenever the triangle is "green" subdivided, if this triangle is marked for future division then the original triangle is reinstated and regular subdivision is applied.

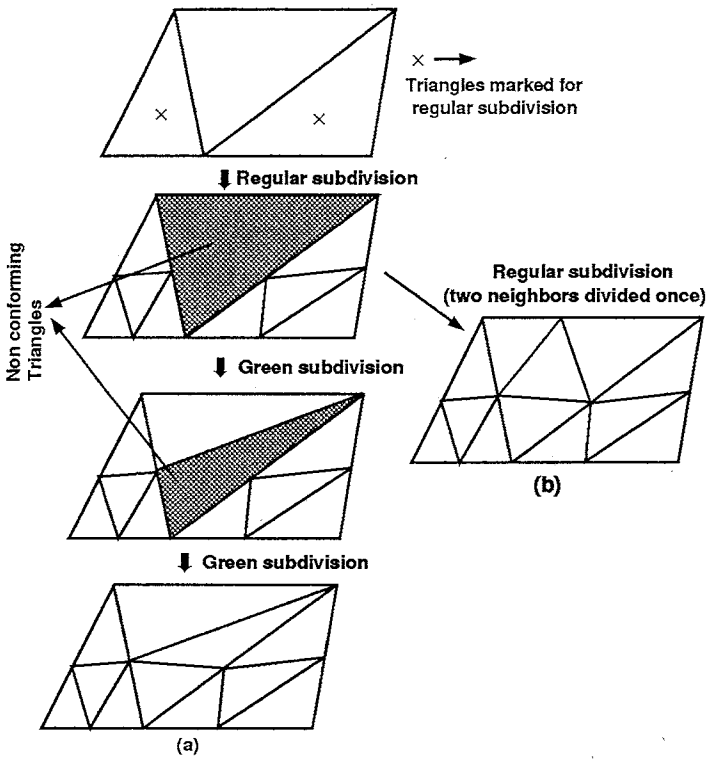


Figure 3. Green and regular subdivision used for maintaining conformity

2.1.2 Bisection algorithms In the bisection algorithms, Rivara [9], [6] use bisection of the triangles by the longest side. They use two variations of the bisection algorithm:

1. The 2–triangles algorithm [7].
2. The 4–triangles algorithm [8].

In the 2–triangle algorithm, for any triangle ${}_MT_t^2$ marked for refinement, the longest edge of the triangle is bisected first (generalized bisection) and let ${}_MT_P^0$ be the midpoint generated on the longest edge (see Figure 4(a)). If ${}_MT_P^0$ is a nonconforming midpoint of a triangle ${}_MT_{t^*}^2$ (see Figure 4(a)), first bisect the ${}_MT_{t^*}^2$ by the longest side to obtain ${}_MT_Q^0$, Figure 4(b). If ${}_MT_P^0 \ddagger {}_MT_Q^0$, then do a simple bisection (bisection at the mid point of any edge) to make the non conforming side conforming (see Figure 4(c)). This process is recursive and the process stops when all the triangles have been made conforming.

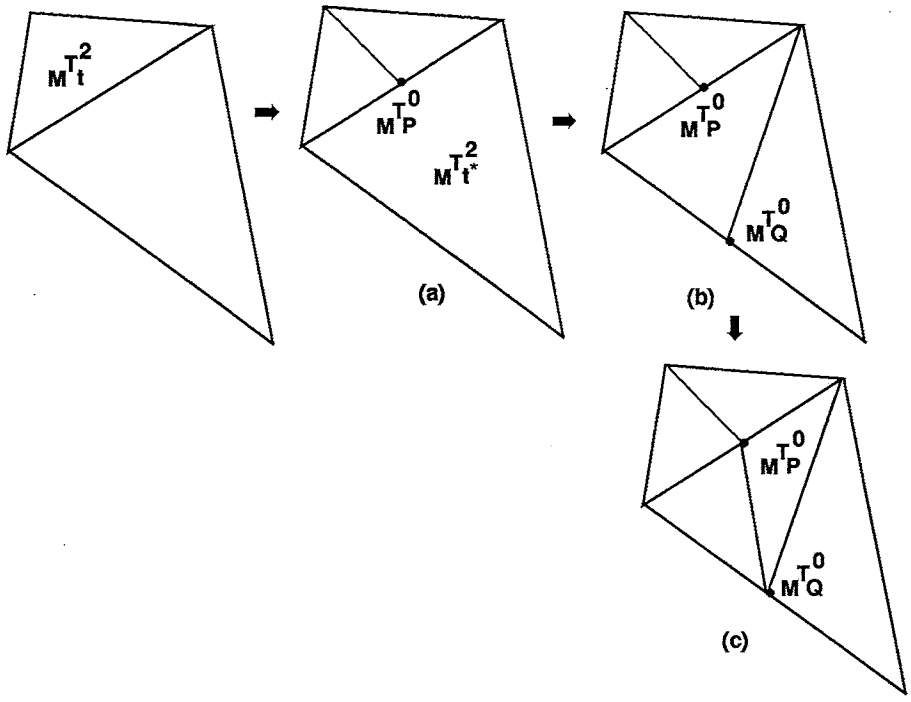


Figure 4. Generalized bisection

The four triangle algorithm is similar to the two triangle algorithm, but in this case the triangle is divided into four new triangles instead of two and the resulting nonconforming triangles are made conforming as given in the previous procedure. The main difference between the two algorithms is depicted in Figure 5(a) and (b). In this algorithm also adequate history information is stored regarding the parent and the child triangles.

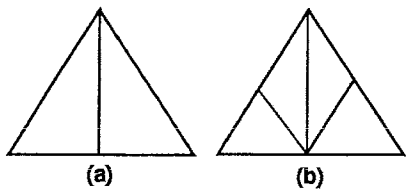


Figure 5. 2 triangle subdivision and 4 triangle subdivision

2.1.3 Insertion type algorithms Insertion type algorithms mainly deal with inserting a point into a Delaunay triangulation. Inserting a node into a Delaunay triangulation can be achieved using the Watson's [12] algorithm. In Watson's algorithm all elements which contain the point (interms of circumcircle containment) are deleted and a convex polyhedral cavity is formed. New triangles are created by connecting the boundary of the cavity to the newly inserted node (see Figure 6).

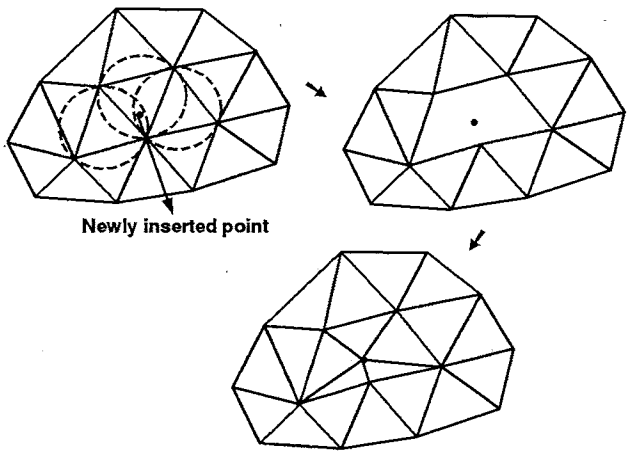


Figure 6. Point insertion in a Delaunay triangulation

The point insertion can also the achieved by splitting the entity on which the points fall, for example if the point falls on an edge then the concerned edge is split or if it falls on the face then the concerned face is split (see Figure 7).

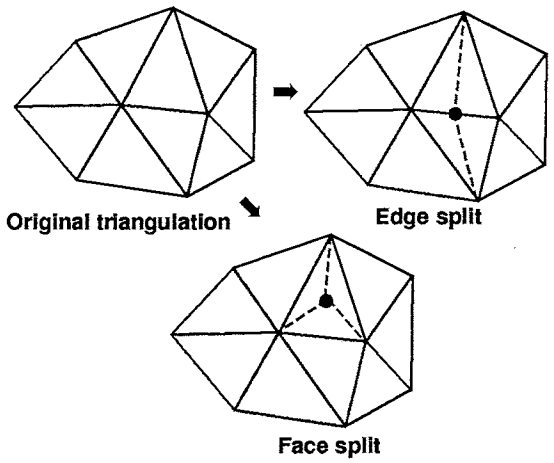


Figure 7. Edge/face split in 2D

2.2 Derefinement

Derefinement can be seen as an inverse process of refinement algorithms defined above. All the refinement methods discussed above maintain history information as to which original triangle the refined triangles belong to. This history information is used in the derefinement process. For example Rivara [9] modifies several discretizations of the sequence in order to maintain the nestedness of the meshes. For example in Figure 8(a) in order to derefine the shaded triangle (${}_M T_t^2$) which was obtained by successive refinement, every triangulation of the sequence that contains the ${}_M T_t^2$ is modified. That is the derefinement procedure also modifies the history of the refinement as shown in Figure 8(b).

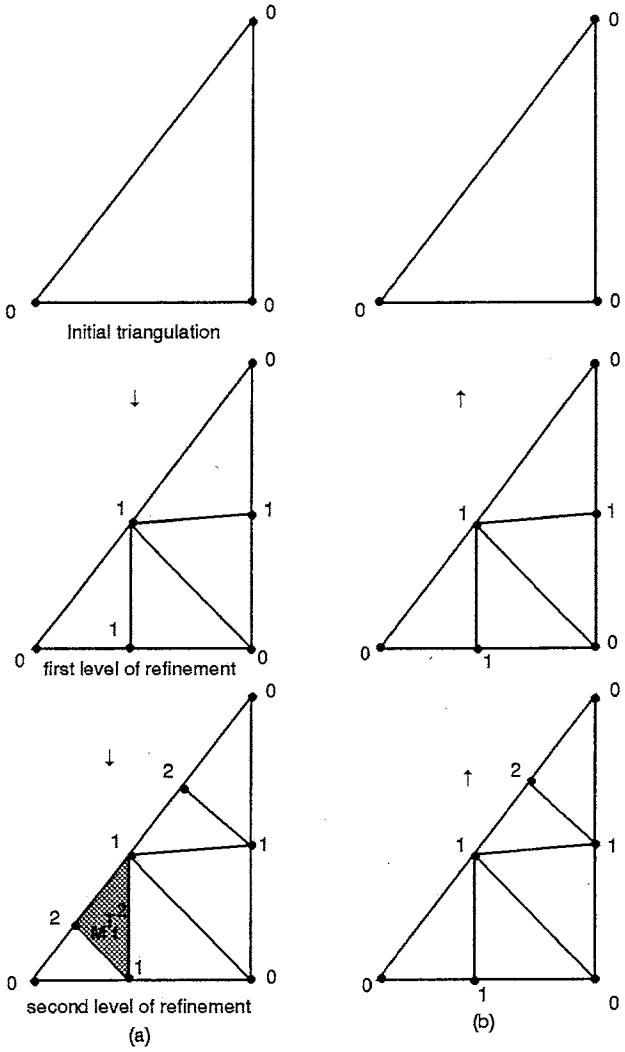


Figure 8. History information used for derefinement

3 Edge based refinement and derefinement procedures

3.1 Refinement

In this approach refinement is achieved by marking appropriate mesh edges for refinement. Once the edges have been marked for refinement the triangle is

refined using sub division patterns (templates). Up to three edges can be marked for refinement for each triangle. There are four possible templates for refinement based on the number of edges in the triangle that needs to be refined. The refinement templates are,

1. 1 edge needs to be refined (see Figure 9(a)).
2. 2 edge needs to be refined (see Figure 9(b)).
3. 3 edge needs to be refined (two templates) (see Figure 9(c) and (d)).

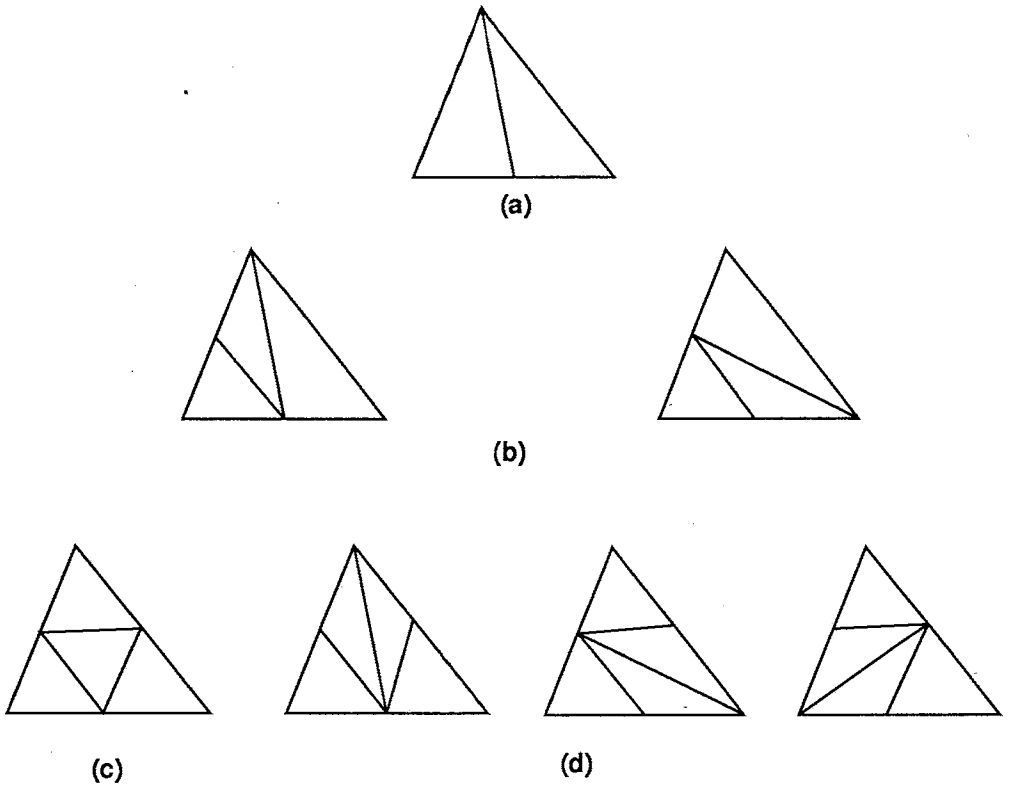


Figure 9. Templates for refinement in 2D

There are two ways in which the triangle can be split if two edges are marked for refinement, Figure 9(b), and similarly there are three ways in which the triangle can be split if three edges of it are marked for refinement, Figure 9(d). The template given in Figure 9(c) is the same as the regular subdivision (1:4) as given in [2] and the templates given in Figure 9(d) is same as the 4-triangle bisection algorithm given in [8].

3.2 Derefinement

Derefinement is achieved by marking the mesh edges for derefinement and the marked edges are derefined by collapsing the edge to one of the ending vertices (hence there are two ways in which an edge can be collapsed, see Figure 10).

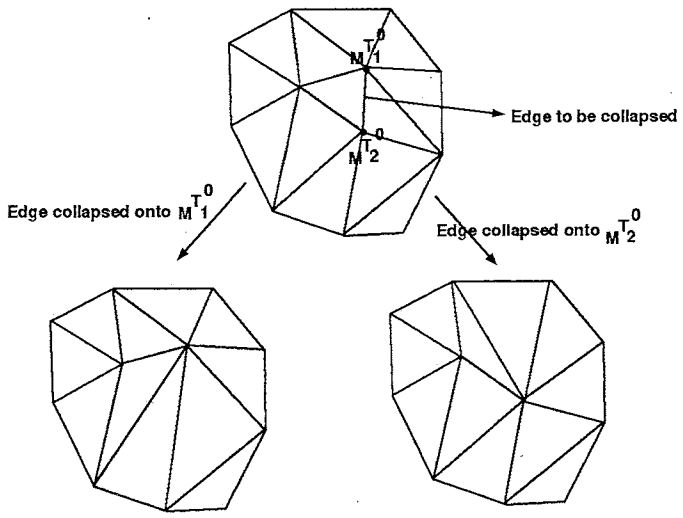


Figure 10. Two different collapsed configurations

Before collapsing one of the ending vertices ($_M T_d^0$, deleted vertex) onto the other vertex ($_M T_t^0$, target vertex) the collapse is evaluated for topological compatibility and geometric validity (a detailed description of the topological and geometric checks done is given in [5]).

For the final mesh to be geometrically valid after collapsing in 2D the signed area of the newly created mesh faces should be positive if the vertices of the faces are taken in a particular order. Figure 11 shows one of the geometrically invalid situations for collapsing.

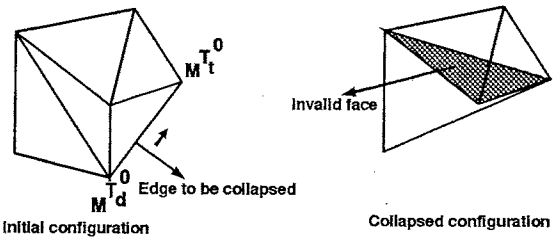


Figure 11. Invalid configuration for collapsing

If both the vertices are valid for collapsing then that vertex is chosen for collapsing which gives better triangulation. During derefinement the metric of the resulting triangulation is not allowed to fall below some threshold value for a user specified criteria (element aspect ratio, face angles etc.)

Once the target vertex is chosen after the topological and geometric checks, [5], collapsing can be broken down into the following steps:

1. Delete all the mesh faces connected to ${}_M T_d^0$, which forms a convex cavity (see Figure 12(b)).
2. Connect the vertices of the cavity to the ${}_M T_t^0$ forming new mesh faces and mesh edges (see Figure 12(c)).

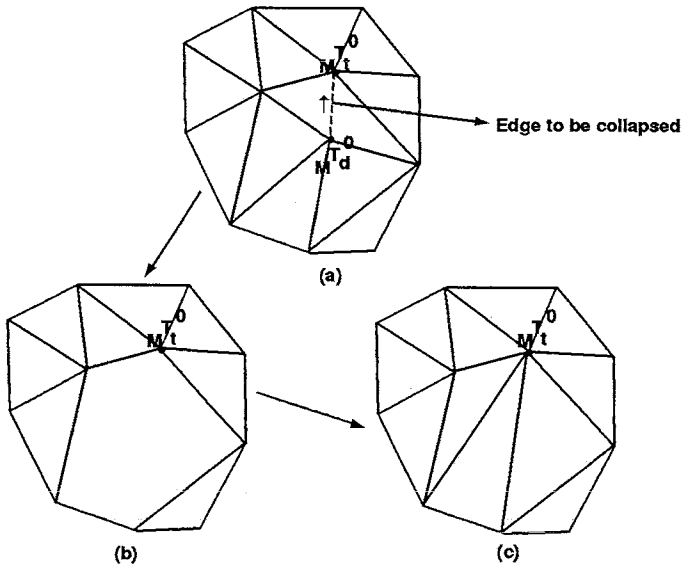


Figure 12. Steps involved in collapsing an edge

4 Implementation details

This section describes the implementation details of the refinement/derefinement procedures discussed in the previous section. These procedures have been written as a stand alone module without any tie to a particular mesh generator or a modeler. To achieve this the module which query or change the mesh database have been made operator driven. The current implementation uses the SCOREC mesh database operators [3] to interact with the modeler and the mesh.

The main interaction with the geometric modeler is to get the proper spatial coordinates at the mid point of the edge when the edge is split. When a mesh edge classified on a boundary edge (${}_GT_i^1$) is split the newly created vertices should be snapped onto the appropriate model entity instead of living it on the straight sided segment (see Figure 13).

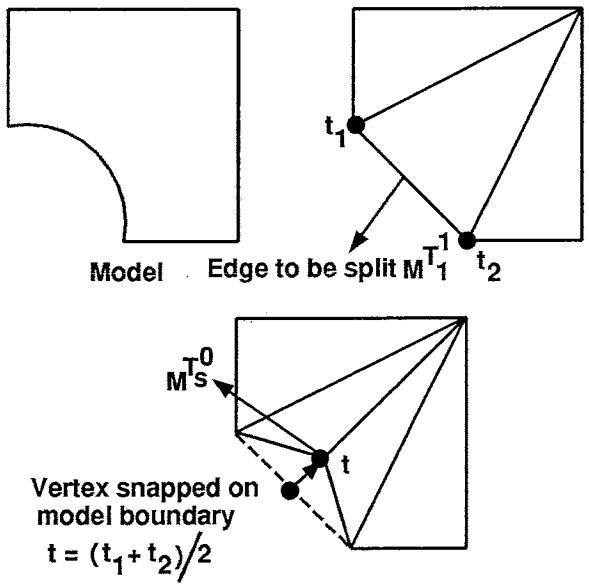


Figure 13. Snapping the vertex on the boundary of the model entity

A linear interpolation in the parametric space of ${}_GT_i^1$ is done to obtain the parameter values of the split point (see Figure 13) and the modeler functionality is used to obtain the spatial coordinates given the parametric value on the model entity. In the Figure 13 to obtain the parameter values (t) of the split point

${}_M T_s^0$ (which is in the middle of the edge ${}_M T_1^1$) a linear interpolation between the parameter values of the two end points of the edge is done to obtain $\mathbf{t} = (\mathbf{t_1} + \mathbf{t_2})/\mathbf{2}$. Modeler functionality is also used to get the parameter values on a higher order entity given the parameter value on a lower order entity.

The pseudo code describing the refinement process is given in Figure 14 and the pseudo code describing the derefinement process is given in Figure 15.

```
Procedure 2DRefine
{
/* This module does the overall refinement in 2D */
/* Loop over all the mesh faces, and find if any
   of its edges needs to be refined */
   for ( i = 0 ; i < numFaces ; i++ )
    {
      /* Check if any of the edges needs to
         be refined */
      numRefine = refineEdges(i) ;
      /* Switch to the appropriate template
         based on the number of edges to be refined */
      if ( numRefine == 1 )
       {
         /* Only one edge needs to be refine, call the
            1 edge template */
          1_edgeTemplate(i) ;
       }
      else if ( numRefine == 2 )
       {
         /* Two edges of the face needs to be refined, call the
            2 edge template */
          2_edgeTemplate(i) ;
       }
      else if ( numRefine == 3 )
       {
          /* Three edges of the face needs to be refined, call the
             3 edge template */
          3_edgeTemplate(i) ;
       }
    } /* End of loop over all the faces */
}
```

Figure 14. Pseudo code describing the refinement process

```
Procedure 2DDerefine
{
  /* This is the main module which does the derefinement of
     mesh edges */
  /* Loop over all the mesh edges and check if the edge has
     been marked for derefinement */
     for ( i = 0 ; i < numEdges ; i++ )
      {
        if (OKto(Derefine,i))
         {
            /* Edge is o.k to derefine, check for topological
               and geometric validity */
            /* Vertex 0 of the edge is the collapsed vertex */
            status1 = checkValidity(vertex0,vertex1,&metric1) ;
            /* Vertex 1 of the edge is the collapsed vertex */
            status2 = checkValidity(vertex1,vertex0,&metric2) ;
            /* Check if both of the collapse is valid */
            if ( status1 && status2 )
             {
               /* Choose the best among the two collapse */
               if ( metric1 > metric2 )
                  2dCollapse(i,vertex0,vertex1) ;
               else
                  2dCollapse(i,vertex1,vertex0) ;
             }
            /* Check which configuration is valid */
            else if ( status1 )
              2dCollase(i,vertex0,vertex1) ;
            else if ( status2 )
              2dCollapse(i,vertex1,vertex0) ;
         } /* Edge is o.k to refine */
      } /* End of loop over all the edges */
}
```

Figure 15. Pseudo code describing the derefinement process

5 Results of implementation

Figure 16 shows a thick walled cylinder subjected to uniform external pressure. One quarter of the cylinder was modeled due to symmetry. The geometric model was generated using the PARASOLID solid modeler and the initial mesh shown in Figure 17(a) was generated using Finite Octree [11]. ABAQUS was used for assembly and solution and an interior residual error estimator which utilizes the flux balancing algorithm of Ainsworth and Oden [1] has been used to estimate the error. The decisions about which elements to split for refinement was predicted by an algorithm given in [4]. The refinement/derefinement procedures given in Section 3 have been used for the local refinement of the mesh. The mesh shown in Figure 17(e) is final mesh generated after the second adaptive step and the mesh shown in Figure 17(c) is an intermediate mesh. The relative error in the results of the analysis using the mesh in Figure 17(e) was less than the allowable relative error of 5% which was specified by the user.

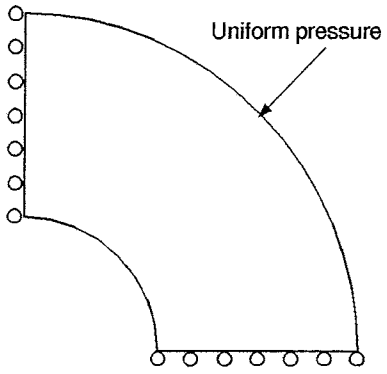
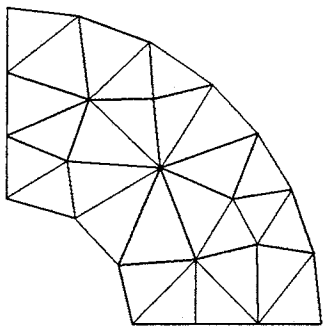
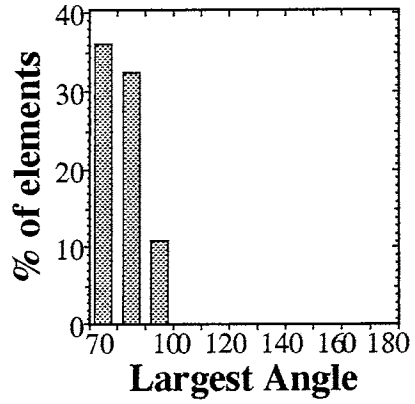


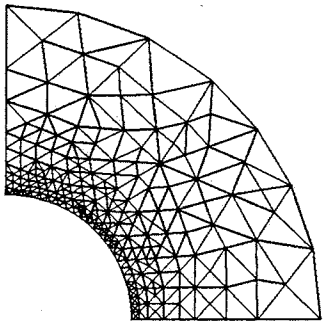
Figure 16. Thick wall cylinder, boundary conditions



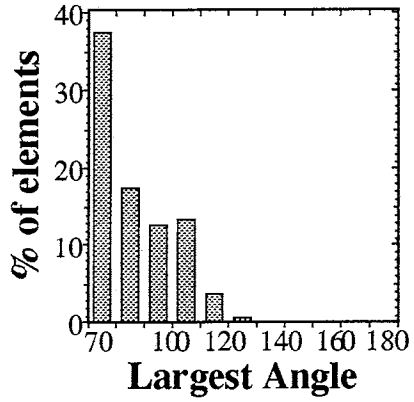
(a)



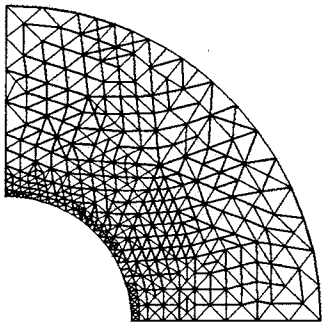
(b)



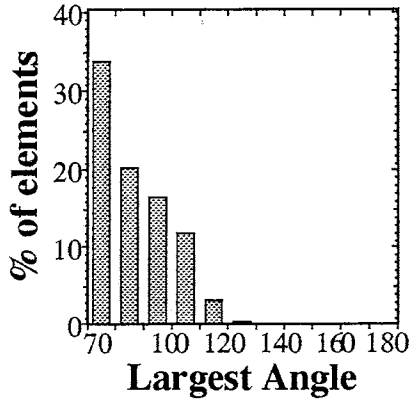
(c)



(d)



(e)



(f)

Figure 17. Initial and final meshes used in the analysis

Figure 18 shows a square plate with symmetric cracks subjected to a uniform pressure (tension) load. One quarter of the plate was modeled due to symmetry. The mesh shown in Figure 19(a) shows the initial mesh which was used in the analysis. The finial mesh shown in Figure 19(e) was generated after the second adaptive step. The allowable relative error was specified as 5%. Figure 19(c) shows one of the intermediate meshes in the adaptive analysis.

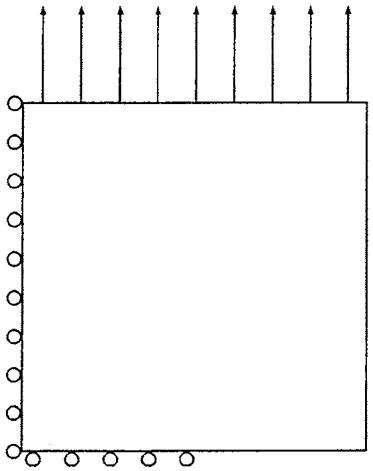
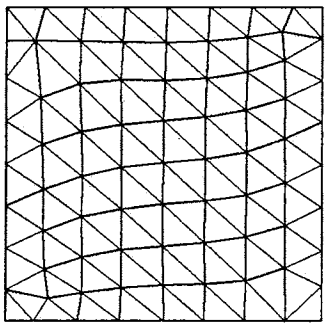
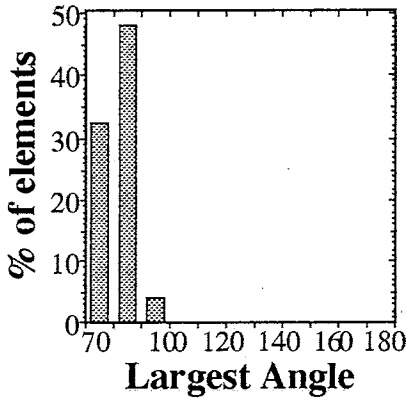


Figure 18. Cracked plate with boundary conditions

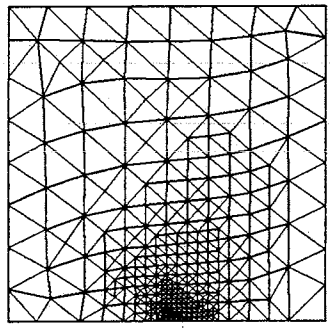


(a)

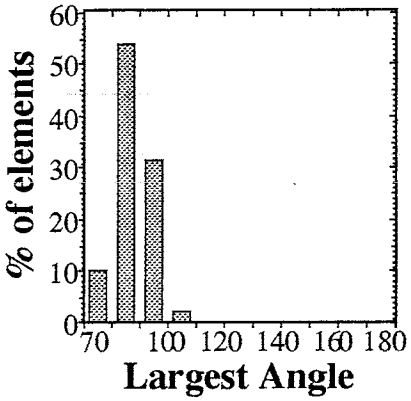


(b)

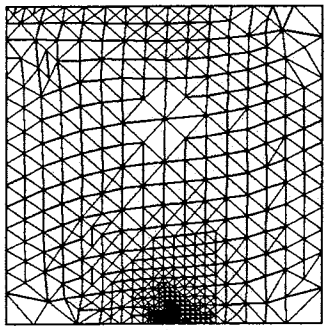
Figure 19. Initial and final meshes used in the analysis (Continued) . . .



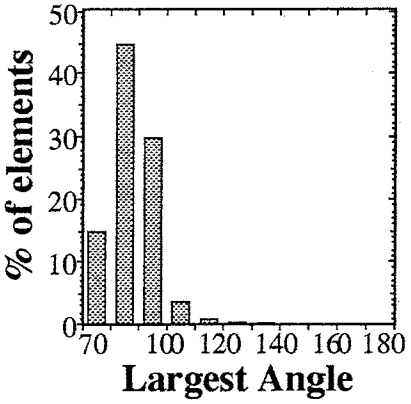
(c)



(d)



(e)



(f)

Figure 19. Initial and final meshes used in the analysis

Figure 20(a) shows a bracket subjected to a uniform pressure (tension) load. The mesh shown in Figure 20(b) shows the initial mesh which was used in the analysis. The finial mesh shown in Figure 20(d) was generated after the second adaptive step. The allowable relative error was specified as 5%. Figure 20(c) shows one of the intermediate meshes in the adaptive analysis. Figures 21(a), 21(b) and 21(c) shows the histograms of the largest angles in the three meshes respectively.

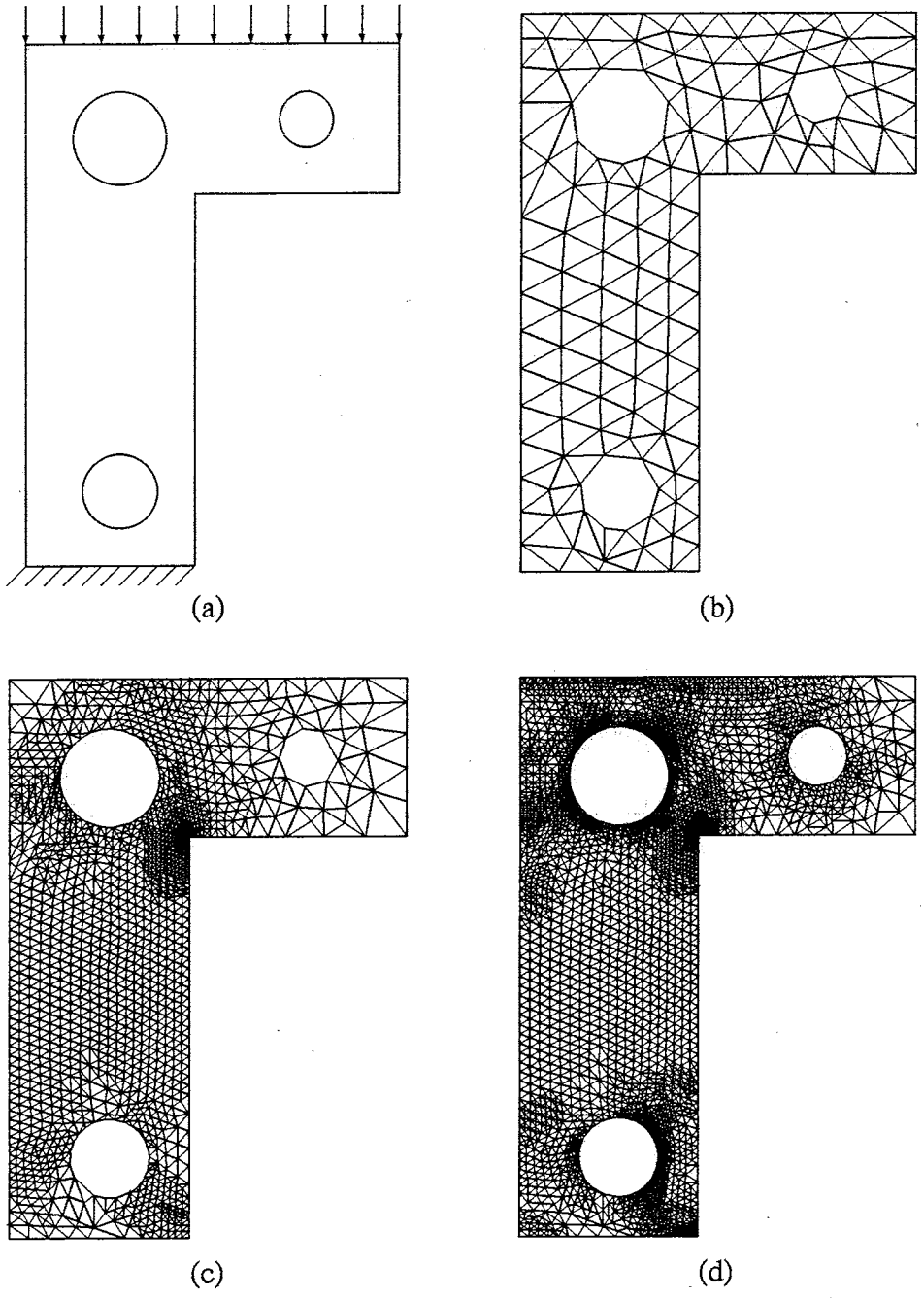


Figure 20. Bracket example. (a) Geometric model and boundary conditions. (b) Initial discretization. (c) Discretization after first adaptive step. (d) Discretization after second and final adaptive step.

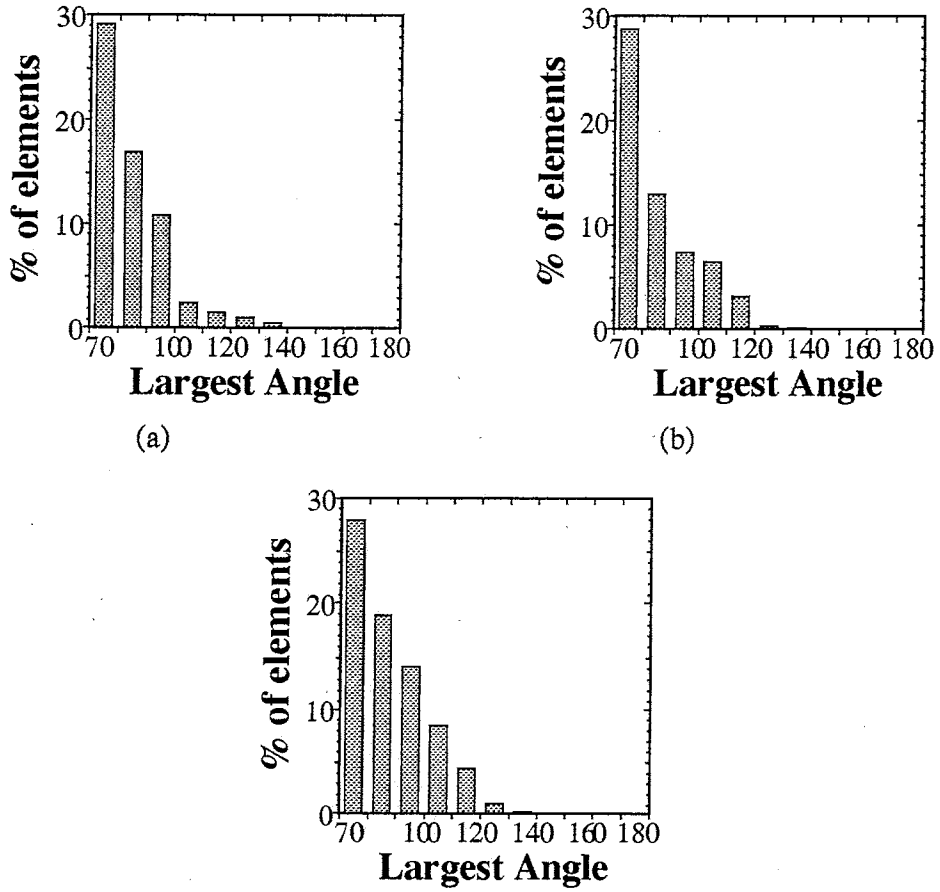


Figure 21. Figure showing the mesh quality for the bracket example

Bibliography

[1] M. Ainsworth and J. T. Oden. A procedure for a posteriori error estimation for h-p finite element methods. *Comp. Meth. Appl. Mech. Engng.*, 101:73–96, 1992.

[2] R. E. Bank and A. H. Sherman. An adaptive multi-level method for elliptic boundary value problems. *Computing*, 26:91–105, 1981.

[3] M. W. Beall. Scorec mesh database user's guide, version 2.2 - draft. Technical Report SCOREC Report # 26-1993, Rensselaer Polytechnic Institute, Troy, NY 12180-3590, January 1994.

[4] R. Collar, R. Wentorf, and Shephard M.S. A multiple level h-refinement adaptive analysis procedure using patran, p/fea and finite octree. Technical Report SCOREC Report # 27-1993, Rensselaer Polytechnic Institute, Troy, NY 12180-3590, January 1994.

[5] Hugues L. de Cougny and Mark S. Shephard. Parallel mesh adaptation by local mesh modification. 1995. Submitted for publication, SCOREC Report.

[6] R. V. Nambiar, R. S. Valera, and K. L. Lawerence. An algorithm for adaptive refinement of triangular element meshes. *Int. J. Numer. Meth. Engng.*, 36:499–509, 1993.

[7] Maria-Cecilia Rivara. Algorithms for refining triangular grids suitable for adaptive and multigrid techniques. *Int. J. Numer. Meth. Engng.*, 20:745–756, 1984.

[8] Maria-Cecilia Rivara. A grid generator based on 4-triangles conforming mesh-refinement algorithms. *Int. J. Numer. Meth. Engng.*, 24:1343–1354, 1987.

[9] Maria-Cecilia Rivara. Selective refinement/derefinement algorithms for sequences of nested triangulations. *Int. J. Numer. Meth. Engng.*, 28:2889–2906, 1989.

[10]Maria-Cecilia Rivara. A 3-D refinement algorithm suitable for adaptive and multi-grid techniques. *Communications in Applied Numerical Methods*, 8:281–290, 1992.

[11]M. S. Shephard and M. K. Georges. Automatic three-dimensional mesh generation by the Finite Octree technique. *Int. J. Numer. Meth. Engng.*, 32(4):709–749, 1991.

[12]D. F. Watson. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *The Computer J.*, 24(2), 1981.