Reprinted from

FINITE ELEMENTS IN ANALYSIS AND DESIGN

Finite Elements in Analysis and Design 17 (1994) 205–229

Algorithms and data structures for 2D and 3D adaptive finite element mesh refinement

K.C. Chellamuthu*, N. Ida

Department of Electrical Engineering, The University of Akron, Akron, OH 44325-3904, USA

Received January 1993; revised May 1994

ELSEVIER Finite Elements in Analysis and Design 17 (1994) 205–229

FINITE ELEMENTS IN ANALYSIS AND DESIGN

Algorithms and data structures for 2D and 3D adaptive finite element mesh refinement

K.C. Chellamuthu*, N. Ida

Department of Electrical Engineering, The University of Akron, Akron, OH 44325-3904, USA

Received January 1993; revised May 1994

Abstract

This paper introduces an effective mesh refinement strategy and a data structure for adaptive Finite Element (FE) computation of 2D and 3D problems. A numerical case study and an implementation to solve linear elliptic boundary value problems are presented, and the complexity and efficiency of the refinement algorithm and its data structure are analyzed. The proposed algorithm utilizes a hierarchical minimal tree based data structure for mesh refinement. The amount of tree traversal normally required during the mesh refinement process is minimized in this approach. The algorithm is implemented by imposing a *one-level* rule and also utilizing the adjacent neighbor (edge and face sharing) concept for recursive refinement. This technique generates mesh refinement data such as a connectivity matrix, an automatic local and global node numbering, a natural order of element sequence, and a coordinate array for the refined elements. The proposed data structure allows easy identification of elements in the tree and also the assimilation of characteristic mesh data necessary for refinement. The data structure facilitates identification of the types of nodes after each level of refinement for applying appropriate boundary conditions. The refinement algorithm and the associated data structure have been tested by solving a set of self adjoint boundary value problems using bilinear (quadrilateral) and trilinear (hexahedral brick or cubic) elements in 2D and 3D, respectively. The numerical case studies quantitatively establish the effectiveness of the proposed refinement algorithm and the data structure for practical problem solving.

1. Introduction

The discretization of a problem domain by the FE method to achieve solutions of specified accuracy has been a growing field of research in recent years. The traditional approach in numerical modeling using the FE method provides solutions with an accuracy of 5–10% for most problems in science and engineering. The discretization error present in the problem domain limits the accuracy of the solution. By uniformly refining the mesh, the solution accuracy can be

*Corresponding author.

improved to a certain extent, but more memory and greater computational effort are required. On the other hand an automatic adaptive mesh refinement and computation helps to achieve solutions with accuracy of 2% or less in a more economical way. Also in the presence of a singularity in the problem domain, the solution accuracy can be improved by a selective discretization process using an adaptive FE technique. This is achieved by identifying the critical regions of the problem domain, refining them locally, and computing the solution and the corresponding refinement parameters (error estimators) in some convenient norm. The recursive procedure of refining the mesh continues until the specified error criterion is met. In order to achieve an optimal adaptive mesh having a few degrees of freedom, and capable of providing a solution of required accuracy, it is necessary to have an efficient adaptive refinement strategy and an associated data structure.

An optimal (or nearly optimal) mesh generation procedure is a vital component of an efficient numerical technique employing a FE method. FE analysts used to generate improved meshes by applying the experience gained from the previous computation, subject to the rules on permissible element shapes or sizes or the behavioral characteristics of the results. Often this kind of trial and error judgement failed, producing erroneous input mesh data. It required remodeling the problem and also special expertise to apply the method.

In addition to the computational complexity involved in the fully automatic adaptive process, the mesh refinement also generates an enormous amount of data during the sequence of discretization steps. The adaptive refinement process involves various tasks such as organization of storage, identification of element location in the data tree, and labelling and computing the elemental data necessary for refinement.

Owing to the complexity of computation and also the volume of data involved in the process, it is essential to develop efficient algorithms and data structures for optimal mesh refinement. Besides the techniques necessary to model the complexities of a problem and its geometry, it is also essential to take care of the various data management functions during the adaptive refinement process. Since the efficiency of an adaptive mesh refinement algorithm depends on the accuracy of predicting the *discretization error* present in different regions on the problem domain, it is necessary to incorporate a reliable and robust *a posteriori* error estimation technique. Analysis and performance evaluation of the complexity and efficiency of the mesh refinement algorithm and data structure as applied to realistic problems will provide insight into the functionality of an automatic adaptive FE method.

Many algorithms and the associated tree data structure for automatic adaptive refinement of 2D problems, and to a limited extent 3D problems, have been reported in the past [1–8]. A comparison of different triangulation algorithms with various error indicators has been discussed with extensive numerical results in [7]. This paper presents the analysis and implementation of an algorithm and an associated data structure for the adaptive refinement of 2D and 3D elliptic boundary value problems. The algorithm utilizes a hierarchical minimal tree structure by imposing a *one-level* rule to produce a graded and smooth mesh. In this approach a parent element is refined into four subelements (quadrants) in 2D and eight subelements (octants) in 3D. The nodes in the refined mesh are classified as normal or regular nodes, constrained or irregular nodes and boundary nodes. In order to maintain the continuity of the solution across element boundaries, a special procedure is developed to process the constrained nodes on the common edges and faces of the refined elements. The algorithm outlined in the following sections adaptively generates a graded admissible mesh. A simple *a posteriori* error estimation technique using interpolation and post-processing of the solution is developed to predict and control the adaptive refinement process. A numerical case study has been carried out to evaluate the efficiency of the refinement algorithm and data structure by solving a set of 2D and 3D boundary value problems in electromagnetics. The paper is organized as follows: In the first part of the paper, the algorithm for 2D and 3D mesh refinement is discussed. The organization and functionality of a dynamic data structure which aids the refinement process is analyzed in the second part of the paper. A numerical case study involving the result and performance evaluation of the algorithm is provided in the last part of the paper.

2. Adaptive mesh refinement strategies

Domain discretization is an indispensable process in a FE computation. The discretization error depends on the size and the distribution of elements in the problem domain. Hence it is necessary to refine the problem domain selectively based on a reliable error estimation strategy in order to improve the solution accuracy. Thus *adaptation* is the procedure by which the problem mesh is recursively refined and mesh refinement parameters are computed. This recursive procedure is continued until a specified solution accuracy corresponding to an optimal (or nearly optimal) mesh is obtained. In an iterative numerical modeling of self adjoint problems, an optimal adaptive mesh is identified by the equi-distribution of discretization error among all the elements.

An adaptive process can be classified as a feedback procedure, because there is an initial uncertainty or insufficient *a priori* input data regarding the critical region of the domain and the refinement parameters. The uncertainty in the system is reduced by accumulating the data that becomes available during the automatic mesh refinement process. This accumulated data and knowledge gained can be used to control the process itself optimally to improve the input in a feedback procedure. The refinement parameters such as error indicator and error estimators are the primary data obtained during the feedback process to automatically control the recursive refinement. The optimality of the mesh and the desired accuracy of the solution can be attributed to the reliability of the data generated during the feedback process.

The various stages involved in the algorithmic procedure for adaptive FE strategy are summarized as follows:

(1) Generate an initial coarse mesh (Ω_0) and specify the error tolerance.
(2) Solve for an initial solution (Φ_0).
(3) Choose a suitable error estimation strategy and compute the error in the solution using an energy norm $\|\cdot\|$.
(4) Compare the error estimator $\| e \|_\Omega$ with the user defined error tolerance (accuracy of solution). If the error is less than the specified tolerance then exit, else continue.
(5) Locate the elements which have large errors (i.e., those in the region of singularity) and mark them for refinement.
(6) Based on the refinement policy, add new degrees of freedom to the element and refine the mesh.
(7) Use an appropriate solution technique to solve and iterate the solution.
(8) Go to step 3.

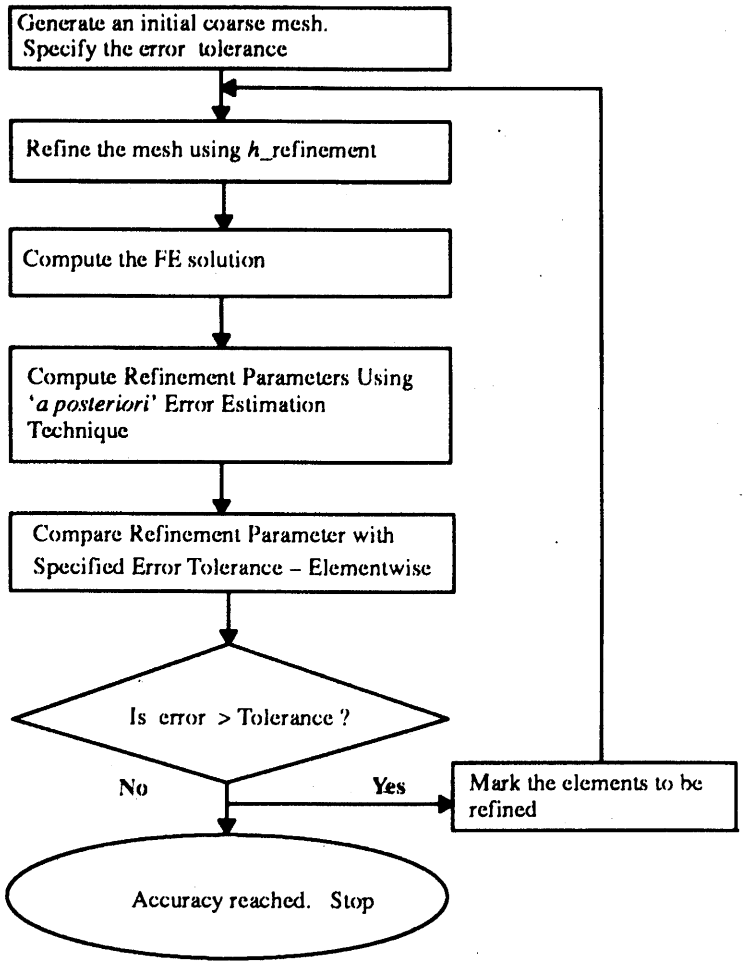

Generate an initial coarse mesh. Specify the error tolerance
Refine the mesh using h_refinement
Compute the FE solution
Compute Refinement Parameters Using 'a posteriori' Error Estimation Technique
Compare Refinement Parameter with Specified Error Tolerance – Elementwise
Is error > Tolerance ?
No
Yes
Mark the elements to be refined
Accuracy reached. Stop

Fig. 1. Basic flow diagram of an adaptive mesh refinement.

The flow diagram shown in Fig. 1 illustrates the various phases in an automatic adaptive refinement procedure.

2.1. Mesh refinement strategies

In order to adaptively refine a FE mesh, various mesh refinement techniques are available [6, 8–20]. The mesh refinement procedures are based on various parameters such as the size of the element, the spectral order of the approximating polynomial, or by moving the mesh and relocating the nodes. They are classified as h, p, h–p and r-methods respectively. In the h-method, the solution accuracy is improved by reducing the size of the element ($h_{\max} \to 0$). This means the error in the solution tends to zero (asymptotically) as the element size is reduced. Thus the h-method adds more degrees of freedom during each level of refinement. In the p-method, the size of the element remains the same but the order of the approximating polynomial over the element is increased. Normally the p-method is found to be nearly twice as efficient in convergence as the h-method, as verified and quantified in terms of the number of degrees of freedom by Babuska et al. [21, 22]. In an h–p method, the features of h and p methods are combined. In the r-method, also called the moving mesh technique, boundary nodes are moved along with h-refinement so as to reduce the discretization error by equi-distribution of error over the element. In this method nodes are moved in the region where more error is identified. Based on the procedure by which the error indicator is manipulated and used for adaptive control of the refinement process, adaptation can be achieved by single level adaptation, deterministic adaptation or stochastic adaptation.

The performance of h, p, h–p and r-refinement techniques are evaluated and quantified in [1, 6, 9, 11, 14–29]. Except the following finite element codes, most of the commercially available finite element systems do not incorporate adaptive improvement of solutions. The following are some of the special purpose finite element application systems incorporating, h, p and h–p versions of adaptive techniques: The systems such as the Finite Element Research Solver (FEARS) package using rectangular elements [2, 21, 22]; the EXPDES system; the PLTMG system based on triangular elements; and the TWODEPEP package of Sewell using bisection method of triangulation are based on the h-method. The PROBE is a commercial adaptive FE system. It is based on p- and h–p methods of adaptive refinement. FEARS, PLTMG, and PROBE are designed to solve elliptic boundary value problems in two dimensions [30].

2.2. *Characteristics of an optimal mesh refinement algorithm*

Various algorithms for adaptive mesh refinement of one and two dimensional elliptic boundary value problems based on h, p, and h–p methods have been proposed in recent years [1, 2, 4–14, 16–18]. The quantitative evaluation of the performance of one method over the other is analyzed. However, there are only a few efficient algorithms for mesh refinement and performance analysis of 3D adaptive finite element computation reported in the literature, because of the relative complexity of implementation of a 3D adaptive FE algorithm. In order to generate an optimal mesh which can substantially improve the solution accuracy within the constraints of limited computational resources, an adaptive refinement algorithm must possess the following characteristics:

(i) The algorithm should be simple, efficient and flexible and should generate a robust and reliable FE mesh capable of achieving a solution with a user defined accuracy.
(ii) It should be computationally inexpensive.
(iii) It should generate an optimal mesh, automatically and adaptively, with a reasonable rate of convergence using the data generated during the process, such that there will be an equi-distribution of error over all the elements in the domain.
(iv) The algorithm should embody efficient, robust, and inexpensive methods to compute *a posteriori* error estimation parameters which can work for different physical modeling and geometrical shapes to steer and control the refinement process.
(v) The data obtained during the refinement process should eliminate the problem of an *a priori* insufficient information for refinement and must create an optimal mesh, avoiding an over or under refinement.
(vi) The algorithm must have a provision to efficiently store the data generated during the refinement.

(vii) The algorithm should provide an asymptotic rate of convergence during refinement as the mesh size tends to zero ($h_{\max} \to 0$) or the order of approximating polynomial increases to infinity($p \to \infty$).
(viii) The algorithm should start with the definition of an initial geometry of a domain, material properties, and boundary conditions, and should consistently maintain compatibility during the recursive and adaptive feedback process.
(ix) As a result of iterative refinement, it must be able to add additional degrees of freedom in the mesh recursively and must produce subelements congruent to the parent element.
(x) Although the system is self adaptive, it should provide enough opportunity and access to the user so as to refine selected areas of the problem domain which are viewed to have a singularity of solution.
(xi) It should maintain basic characteristics of elements (non degeneracy), providing conformity and smoothness of a mesh.
(xii) The refinement algorithm must be flexible so as to apply to unstructured domains.

2.3. Two and three dimensional adaptive mesh refinement strategies

The type, shape and the characteristics of an element chosen in an adaptive FE computation has considerable influence over the efficiency of an optimal mesh. Triangular elements in 2D and tetrahedral elements in 3D based on the Delaunay triangulation concepts, provide very good approximations to curved boundaries and complex geometries of the problem domain. Often, the triangulation produces elements with obtuse angles which need to be improved by using Delaunay triangulation. This problem does not exist in quadrilateral and hexahedral brick elements. Also the quadrilateral elements in 2D and hexahedral brick elements in 3D are simple and can be easily visualized in higher order elements and refined meshes. By using isoparametric elements, approximations even to curved boundaries and irregular geometries are easily obtained.

Methods of adaptive mesh refinements in 2D and 3D based on the Delaunay triangulation procedure have been investigated and applied for linear boundary value problems [9–12, 24, 25, 31–38]. Mesh refinement techniques using quadrilateral elements in 2D and hexahedral brick elements in 3D are reported in [3, 4, 6, 20, 28, 39, 40]. Due to the simplicity and flexibility of quadrilateral elements in 2D and hexahedral brick elements in 3D, these elements are chosen for implementing the proposed mesh refinement algorithm. Also higher order quadrilateral and hexahedral brick elements with isoparametric mapping can be easily adopted for the refinement of unstructured domains to generate a graded mesh of optimal quality.

The refinement proceeds automatically utilizing the local property of elements and the refinement parameters computed from an *a posteriori* error estimation technique. A simple postprocessing and interpolation method of error estimation is used to activate the feedback process. The refinement algorithm, utilizes the local property of a quadrilateral element to subdivide an element generating subelements which are of the same shape as the parent element (congruent) in the refined mesh. In order to obtain a graded mesh with a smooth refinement, a *one-level* rule is imposed in the refinement process. This restriction provides a graded quadrilateral and hexahedral brick element meshes with a single constrained node on the boundary between two elements (common edge or face). The *one-level* rule allows the solution to be continuous across element boundaries. The application of a *one-level* rule produces a 1-irregular mesh which automatically follows the rule of gradual mesh transition.

2.4. Admissible meshes

In an adaptive FE procedure, a continuous boundary value problem is approximated for an optimal solution by generating a sequence of mesh discretization. A good mesh refinement algorithm provides an optimal (or nearly optimal) mesh by adding a minimum number of degrees of freedom during each level of refinement. An efficient algorithm should produce a sequence of admissible meshes during the course of refinement in order to satisfy the compatibility and continuity conditions at element boundaries. An admissible mesh in a set of meshes generated in the refinement sequence can be defined by adopting the following set of notations and definitions:

Let the smooth and bounded domain be represented by $\Omega \subset R^n$, where $n = 2, 3$ and also assume that the domain has a well defined regular boundary $\partial\Omega$. If the problem domain Ω is assumed to be the union of many finite elements which are closed and bounded subsets such that $\Omega = \Omega_1 \cup \Omega_2 \cup \Omega_3 \cup \cdots \cup \Omega_m$, where m is the number of subdomains in Ω. Each subset (element or subdomain) has nonempty interiors Ω_i such that $\Omega_i \cap \Omega_j = \Phi$, where $i = 1, m$, for $i \neq j$, and Φ is empty. This means the elements can share only a common edge or face and hence the mesh will be conforming. Based on the type of algorithm used, each of the subdomains Ω_i of Ω is refined into four congruent subdomains in 2D and eight congruent subdomains in 3D using quadrilateral and hexahedral brick elements respectively. In order to generate efficient, graded meshes maintaining the compatibility conditions, each of the subdomains Ω_i when refined should produce subdomains Ψ_k of the same type as Ω_i. By defining curvilinear elements on the selected reference figure and then mapping them onto the subdomain, the finite element meshes are constructed on each of the subdomains Ω_i. If the initial mesh Ω_0 is defined on the set of quadrilaterals q_{i0} such that $q_{i0} = q_1 \cup q_2 \cup q_3 \cup \cdots \cup q_n$, then the admissible meshes on q_{i0} may be defined as collections of N closed elements which are generated by recursively subdividing each of the subdomains Ω_i so that the subelements are congruent to the parent element.

Based on the initial mesh q_{i0}, it is possible to generate a nested sequence of quadrilateral or hexahedral brick element meshes. The mesh M_{i+1} can be generated from mesh M_i as a result of refinement guided by an error estimator. If mesh M_0 consisting of q_{i0}, is admissible, then mesh M_R generated by subdividing any one of the quadrilateral or hexahedral brick elements in M_0 into four or eight congruent elements is also admissible. Thus the admissibility condition maintains the compatibility, the type and shape of elements during the sequence of mesh refinement.

2.5. Regular and irregular meshes

During mesh refinement using h-adaptivity following a one-level rule, irregular nodes (constrained nodes) are introduced into the mesh. A nodal point in the mesh is termed regular, if it acts as a common nodal point for each of the neighboring elements; otherwise it is called an irregular nodal point. If a mesh contains nodes which are all regular, it is a regular mesh. Refined meshes with irregular nodal points are called irregular meshes. The solution at irregular nodes is constrained so that the continuity of the solution is maintained at the boundary between elements. The solution at irregular modes is obtained by interpolating from the adjacent regular nodes forming the edge or face of an element. The *one-level* rule allows only one irregular node on the edge of an element in 2D and on an edge or face in 3D. The number of irregular (constrained) nodes allowed per edge in 2D (edge or face in 3D) of an element in the refined mesh is called an index of irregularity.

The one-level rule imposes the following restriction on the refined mesh in order to satisfy the continuity and compatibility criteria:

(i) There cannot be more than one constrained node between elements sharing a common edge in 2D. Similarly there cannot be more than one edge or face constrained node between elements sharing a common edge or a face in 3D. Thus a *one-level* rule allows a maximum of four edge constrained nodes and one face constrained node between elements sharing a common face in 3D.
(ii) Between adjacent neighbors, the difference in refinement level (generation) cannot be more than one to achieve a graded mesh.
(iii) Solutions for edge and face constrained nodes are obtained by interpolating the values of solutions of two adjacent regular nodes forming the edge and four corner regular nodes which form the corresponding face (3D) respectively.

The application of *one-level* rule on a 1-irregular mesh imposes a restriction on the way the *h*-refinement procedure is implemented. Before proceeding to refine an element, a check is made on the large neighbors. If a large neighbor exists, it is refined first and only then the element in question is refined. In the 1-irregular mesh, a large neighbor of an element can have no more than two small neighbors on a side in 2D and four small neighbors per face and two small neighbors on an edge in 3D. Thus a 1-irregular mesh can also be defined in terms of the refinement levels as; no two neighbors of a 1-irregular mesh can have a refinement level difference of more than one. The constrained nodes and regular nodes are shown in Fig. 2(a) for 2D. The constrained nodes are marked by squares and the regular nodes are marked by circles in 2D. Fig 2(b) shows the face and edge constrained nodes (fcn, ecn) in a 3D mesh. In order to maintain the compatibility and continuity of the solution, the constrained nodes are processed separately using different algorithms.

To implement the algorithm, only quadrilateral in 2D and hexahedral brick elements in 3D are considered in order to maintain the shape and type of the parent element after refinement. The use of *one-level* rule in the algorithm guides refinement by restricting the number of smaller and larger neighbors for an element. The use of *one-level* rule is also mandated from the fact that, it is necessary to maintain a family of tree-based approach for mesh refinement and data structure.

The *quad-tree* and *oct-tree* based algorithms and data structures for the discretization of a problem geometry has been extensively investigated by Shephard et al. [20, 41–43]. In the *quad-tree* and *oct-tree* based approach, the geometry is placed inside a bounding square or a box having an integer coordinate system. The square or the box is recursively subdivided into four quadrants in 2D and eight octants in 3D and the refined subelements are placed in the specific level of an integer data tree. By verifying the location of the refined quadrant or the octant in the geometry, further recursive refinement takes place. Matching of the boundary of the geometry is done by template mapping using cut quads and cut octants procedure.

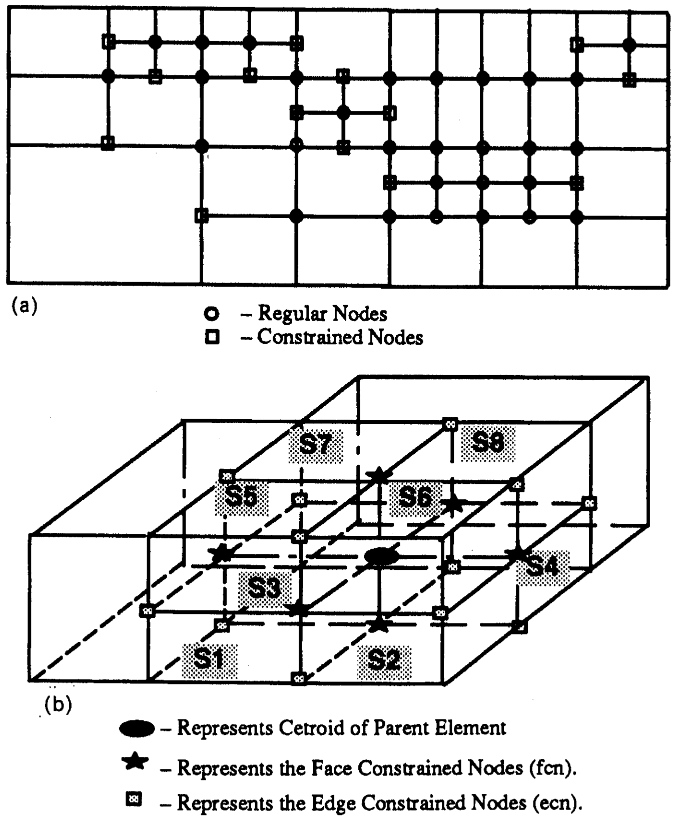

(a)
○ – Regular Nodes
□ – Constrained Nodes
S1
S2
S3
S4
S5
S6
S7
S8
(b)
– Represents Cetroid of Parent Element
– Represents the Face Constrained Nodes (fcn).
– Represents the Edge Constrained Nodes (ecn).

Fig. 2. Constrained and regular nodes: (a) in a quadrilateral adaptive mesh (2D); (b) in an adaptive mesh of a hexahedral brick element (3D) (S1 to S8 are octets of the parent element refined).

In the proposed approach, the basic subdivision of an element into four quadrants in 2D and eight octants in 3D resembles that of a *quad-tree* and *oct-tree* procedure. However the refined elements are represented by means of a hierarchical minimal tree. The proposed method differs from the *quad-tree* and *oct-tree* techniques in the way the mesh refinement algorithm is implemented and the data structure is organized. The data structure maintains a minimal tree of maximum two levels so that the tree traversal is minimized during the refinement process. The refinement is carried out by using the minimal hierarchical tree and the neighboring element information computed during the refinement process.

The refinement technique follows the hierarchical approach based on the tree structure concept by which each super element called macrofather (mf) is divided into four (2D) or eight (3D) fathers (f) and each one of the fathers is again refined into four (2D) or eight (3D) son (s) and again the same way each son is refined into four (2D) and eight (3D) grandsons (gs) in a recursive manner. A hierarchical tree based data structure along with a set of integer array tables efficiently handle the task of organizing and managing the data base necessary for mesh refinement. The algorithm and the data structure also perform the book keeping jobs such as local and global node numbering, keeping track of various types of nodes and updating their types during mesh refinement and applying the appropriate boundary conditions. In order to locate and identify the position of an element in the domain and also in the tree level, the data structure maintains the natural order of element sequence, and its connectivity information.

The proposed adaptive mesh refinement algorithm works by utilizing the set of basic data created from the initial coarse mesh. The initial mesh data includes the element order list, node numbers for each of the elements, nodal coordinates, and integer array for neighbors, type of nodes in each element. The refinement parameter is obtained from the post-processing and interpolation error estimation technique. The minimal tree data structure provides information of the natural order of element sequence and its corresponding refinement level for the subsequent refinement to proceed. Due to the use of an auxiliary data array specifying the element location in the list and the corresponding node numbers, the amount of tree traversal which is necessary to compute the required data for refinement has been minimized to a greater extent in this approach.

2.6. *Two dimensional quadrilateral mesh refinement*

In the case of a 2D, 1-irregular mesh based on *one-level* rule, using quadrilateral elements, adjacent elements can meet only along the edges and there can be equal, smaller or larger neighbors depending on the refinement levels. The refinement is performed by connecting the midpoints of the four sides of a quadrilateral to the element centroid. After the refinement, the element order list is generated to maintain the sequence in order to identify the location of an element in the domain. Also various arrays for node numbers, nodal coordinates and node types are generated and updated during this phase. During the process of refinement, many new constrained nodes are created and the existing constrained nodes become regular nodes when the adjacent elements sharing the edge is also refined in the same refinement level. The nodes are identified as regular nodes (rn), constrained nodes (cn) and boundary nodes (bn). A quadrilateral element in a 1-irregular mesh can have a maximum of 16 possible neighbor configurations sharing all the four edges of an element during the different levels of refinement: four equal neighbors and eight small neighbors with a refinement level one more than the element and also four large neighbors. The various possible neighbor configurations in a 1-irregular quadrilateral mesh are shown in Fig. 3.

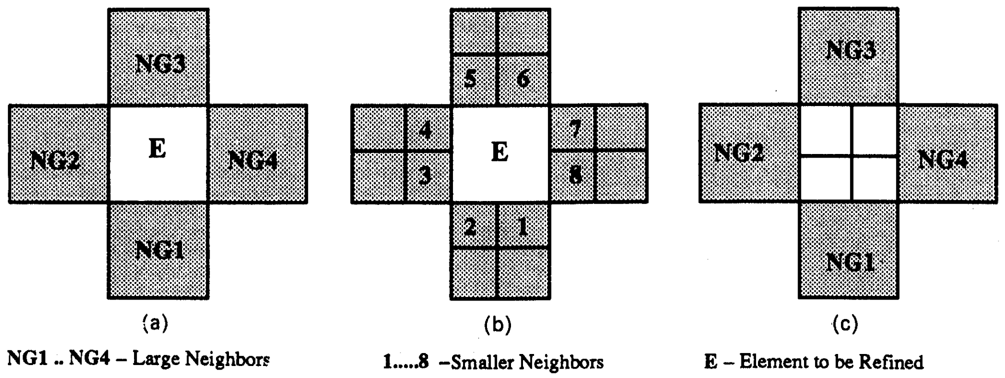

NG3
NG2
E
NG4
NG1
(a)
NG1 .. NG4 – Large Neighbors
5
6
4
3
7
8
2
1
E
(b)
1.....8 –Smaller Neighbors
NG3
NG2
NG4
NG1
(c)
E – Element to be Refined

Fig. 3. The possible neighbor configurations of a quadrilateral element: (a) equal neighbors; (b) smaller neighbors; and (c) larger neighbors.

2.7. Three dimensional mesh refinement of a hexahedral brick element

The following are the set of rules to be adhered while implementing a 3D adaptive mesh refinement algorithm using hexahedral brick elements,

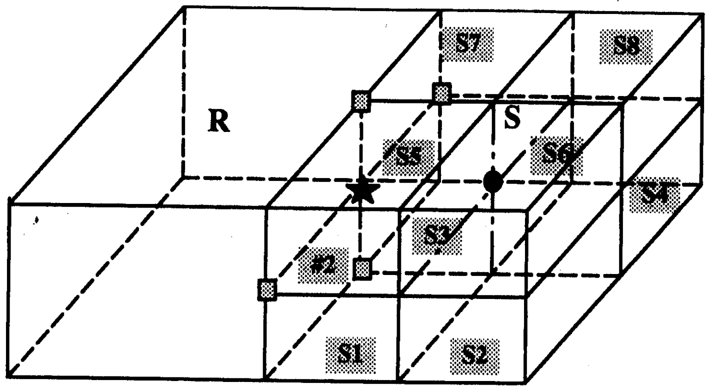
- A *one-level* rule is imposed on the refinement algorithm so as to generate a 1-irregular mesh.
- There cannot be more than four small neighbors per face and not more than two small neighbors per edge of an element sharing a face or an edge respectively.
- The initial mesh is a structured mesh.
- The subdomains and elements generated are of the same type and shape as that of the parent element.
- If the initial mesh satisfies the rules of the refinement and if it is admissible, then the set of meshes generated during the subsequent phases of refinement are also admissible.

A hexahedral brick has six faces and twelve edges. The refinement is performed by connecting the midpoints of all the four edges on each one of the faces to the center point of the corresponding faces. By connecting the midpoints at the center of the faces to the centroid of an element, eight congruent elements of the same type are generated. The facet mid points and the centroids are computed from the eight corner coordinates of the element. Many new nodes are created during the refinement process. Based on the error distribution, the elements for refinement are identified. When the elements are refined, constrained nodes may become regular nodes and new constrained nodes may also be introduced.

The implementation of a *one-level* rule forces a smooth and gradual transition of a mesh avoiding abrupt changes in the mesh. Thus it avoids the large variation of element sizes in the adjoining or neighboring elements. The nodes are identified as rn, fcn, ecn, bn. As per the definition of a 1-irregular mesh, a face of a 3D hexahedral brick element can have maximum of four smaller neighbors and similarly an edge of a large element can have maximum of two smaller neighbors sharing an edge. Suppose there is a face and edge constrained node on face #2 (see Fig. 4(a)) of the hexahedral brick element and if the corresponding neighbor sharing the face #2 is also refined in the subsequent refinement level, then the face and edge constrained nodes become rn as shown in Fig. 4(b).

In a 1-irregular mesh configuration, a 3D hexahedral brick can have a maximum of 84 possible neighbor configurations with different combinations during various levels of refinement: six face neighbors of same refinement level, six large face neighbors (one level less), 24 small face neighbors (one level more), 12 equal level edge neighbors, 12 large edge neighbors (one level less) and 24 small edge neighbors (one level more). The various neighbor configurations in a 3D hexahedral brick element are shown in Fig. 5.

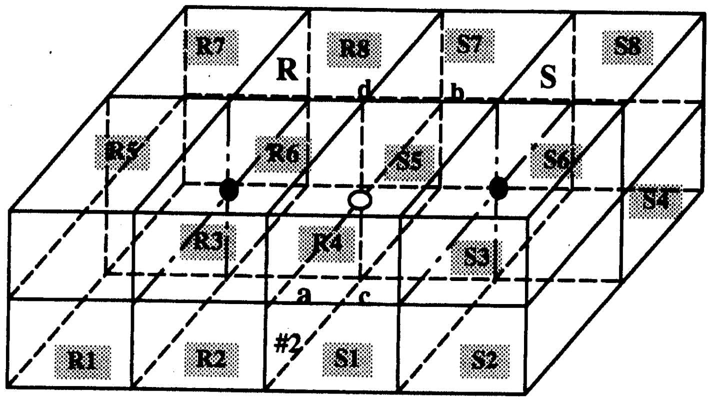
The mesh refinement algorithm for a 3D domain proceeds by utilizing the following data: Arrays for face and edge neighbors for each element, arrays for face and edge constrained nodes, parent element node array and the nodal coordinate arrays and an array for the natural order of element sequence. The natural order of element sequence helps predict the generation and location of an individual element before refinement proceeds. Prior to refining an element, the algorithm identifies the potential face and edge neighbors whose refinement level is one less compared to the level of the element to be refined. This is decided by a look up table using the aforementioned arrays.



● –Centroid, ▨ – Edge Constrained Nodes on Face #2

★ – Face Constrained Node (fcn) on Face #2 S1..S8 are refined octets

(a)



● –Centroid,

○ – Face Constrained Node (fcn) on Face #2 Changes to Regular Node

S1..S8 are refined octets in S and R1 .. R8 are refined octets in R

a,b,c,d – Edge Constrained Nodes Change to Regular Nodes after Refining Adjacent Element (R)

(b)

Fig. 4. Refinement of a hexahedral brick element and change of node types: (a) at Level-1; (b) at Level-2.

2.8. *Edge/Face constrained node processing*

Initially when the coarse mesh is created there will not be any constrained nodes. There exists only two types of nodes viz, regular and boundary nodes. During the course of refinement, many constrained nodes on edges in 2D and faces and edges in 3D are created. The constrained nodes pose a challenge in generating a stiffness matrix maintaining the continuity condition. As explained

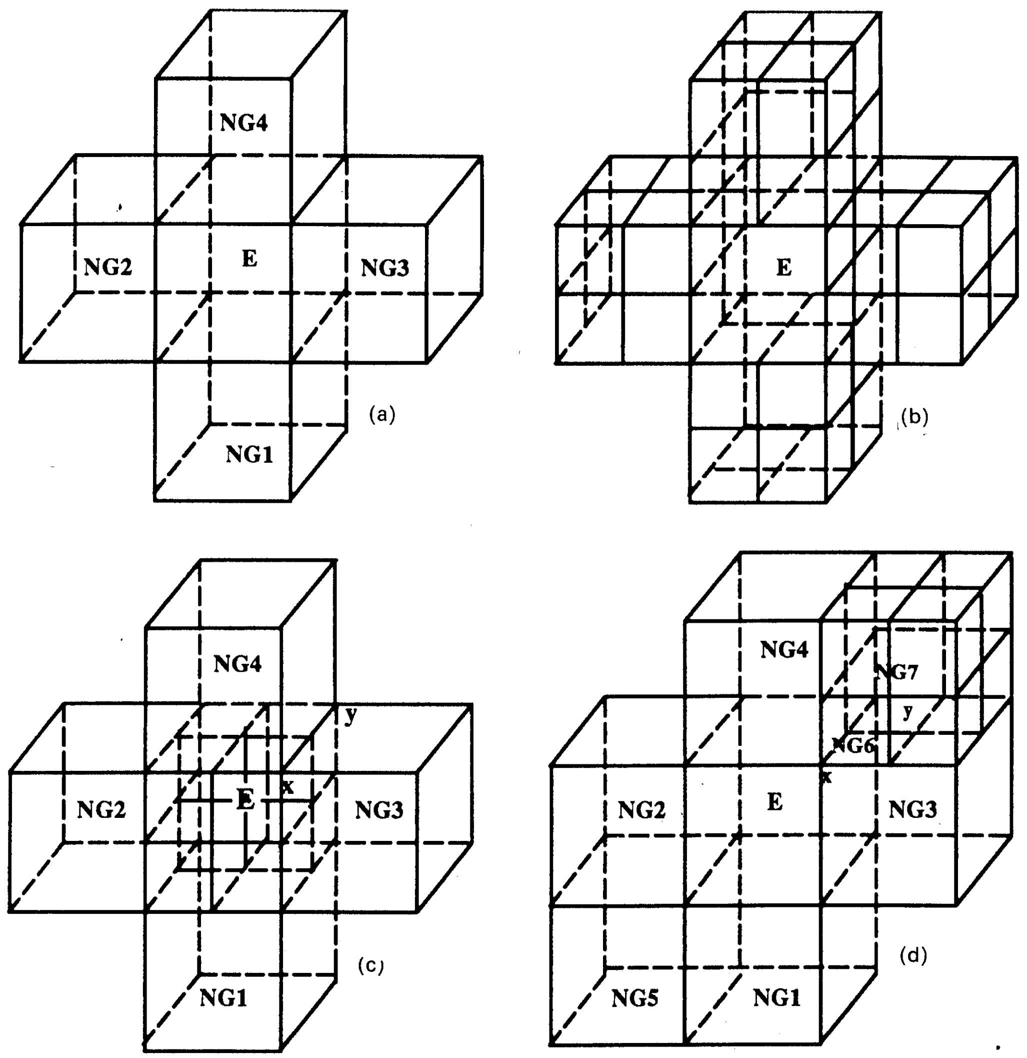

NG4
NG2
E
NG3
NG1
(a)
(b)
y
x
(c)
NG7
NG6
NG5
(d)
E – Element to be Refined
NG1 to NG4 – Equal Neighbors
Neighbor (NG5) and Two Smaller Edge Neighbors (NG6, NG7) on Edge xy

Fig. 5. Various neighbor configurations in a hexhedral brick element mesh refinement: (a) equal face neighbors (only fours faces); (b) four smaller neighbors per face; (c) four larger face neighbors; and (d) four equal face neighbors.

earlier, the constrained nodes occur due to the difference in the level of refinement between adjacent neighbors. In a 1-irregular mesh in 2D, there exists only edge constrained nodes between smaller and higher level neighbors. In a 3D brick element mesh the situation is entirely different due to the occurrence of face and edge constrained nodes at the face common between large and small neighbors. In order to process the constrained nodes, different algorithms will have to be used for constrained nodes on edges and faces of an element. The solution for the edge constrained node is obtained by interpolating the solution of adjacent regular nodes forming the corresponding edge.

3. A dynamic data structure for adaptive mesh refinement

The design of an efficient, dynamic data structure is a vital component for an optimal adaptive refinement algorithm. Basically a data structure in an adaptive FE refinement performs various book keeping operations for aiding the refinement process. The data management functions include numbering of elements using natural element sequence, keeping track of elements, and their locations, node numbering and maintaining nodal coordinates, computing element neighbors, refinement level, degrees of freedom of each of the nodal locations and connectivity information and boundary conditions. The hierarchical refinement strategy adopted in the adaptive refinement algorithm generates various sets of mesh data and pointer information during different stages of refinement. The data base is also updated dynamically during and after the mesh refinement.

3.1. Essential features of an efficient and dynamic data structure

In order to efficiently manage a refinement algorithm, a data structure should possess the following set of features.

(i) It should use less storage for keeping track of the varieties of data necessary to implement an algorithm of comparable complexity.
(ii) It should maintain a uniformity of complexity in mesh generation and updating the dynamic data structure during mesh refinement.
(iii) It must minimize the number of tree traversal necessary to compute the various data required for mesh refinement.
(iv) It should be dynamic and should have a minimal growth for each level of problem discretization.
(v) For a reasonable overhead of storage, it should not introduce complexity of code in the algorithm.

The computational efficiency of a refinement algorithm depends upon the organization of data structure and the number of tree traversal necessary to perform the refinement. Generally a data structure can be represented by means of a set of tables of data pertaining to the sequence of adaptive meshes. The data structure can be organized using the basic set of elementary data structures such as tables, lists, pointers, stacks and queues. The tree data structure representation aided by the tables and pointers is a more appropriate technique for facilitating a hierarchical adaptive mesh refinement algorithm. This is due to the fact that by identifying the level of a given element and its position in the mesh, complete geometrical characteristic data of an element can be computed. Also the refinement process allows gradual construction of a tree data structure.

In a tree based data structure, the nodes represent elements of the mesh and the various branches of the tree correspond to the subelements generated as a result of recursive mesh refinement. At any level of refinement, the lower-most nodes (leaves) represent the unrefined elements of the mesh. The tree based data structure helps to compute information necessary to refine an element using the neighbors list and also the position of an element in the tree and its generation number.

The *quad-tree* and *oct-tree* based mesh refinement algorithms use a family of tree based data structure. Due to the inherent nature of these algorithms, it becomes a tedious task to store and organize the data structure efficiently. The *quad-tree* and *oct-tree* techniques necessitate recursive subdivision of an element into 2^k congruent elements. By using a switching function representation approach [44], the number of subelements generated as a result of element subdivision can be reduced considerably. Despite its efficient handling of data for hierarchic and optimal adaptive refinement, the hierarchic tree based data structure is computationally very expensive especially when the mesh becomes larger. This difficulty increases when the tree becomes large or the nested recursive refinement demands too much of tree traversal. In order to minimize the tree travel during refinement and also to make the algorithm computationally efficient, a minimal tree based data structure is proposed and used in this investigation. According to the minimal hierarchical tree, there will be only two levels in the tree as macrofather (unrefined) and fathers (refined) or fathers (unrefined) and sons (refined) or sons (unrefined) and grandsons (refined) at any point in time. The Fig. 6 illustrates a typical minimal tree based data structure and the corresponding natural order of element sequence of a 3D mesh refinement.

The following are the basic and essential data necessary to perform an adaptive mesh refinement procedure:

- Element numbers (to find the location of an element in the mesh).
- Node numbers of each element obtained from the consecutive numbering scheme from one level of refinement to another.
- Nodal coordinates.
- The solution at each nodal points.
- Various neighbor configurations and data pertaining to the pointers to the corresponding element and node numbers.
- Various node types such as rn, ecn, fcn, bn.
- The minimal tree based data structure which maintains the information such as the parent element number and the corresponding element number and refinement level of the children.

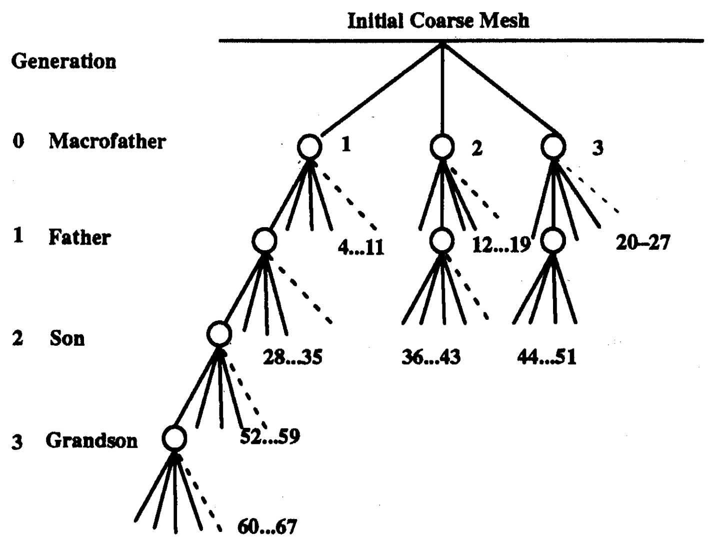


Fig. 6. Labelling procedure for element sequence in a minimal tree based data structure.

The proposed approach differs from other data structure used for adaptive refinement [1–3, 6, 41–43] in the sense that a minimal data tree is created during the refinement process. It helps to reduce the amount of tree travel necessary to identify the location of an element in order to compute the mesh data and thus makes it computationally efficient. The tree is a minimal tree with only two levels at any point in time during refinement. Since the refinement is guided by the imposition of a *one-level rule* and the mesh refinement parameters, the amount of mesh data and the related information generated depends on the nature of a problem domain and the level of refinement. Although the imposition of a *one-level* rule weakens the local property of the mesh, it maintains a relatively simple data structure. In addition, the *one-level* rule facilitates generation of a graded and smooth mesh. The tree data structure used in the mesh refinement maintains the congruency of refined elements facilitating efficient implementation of the adaptive refinement strategy.

3.2. Data arrays and pointers

Assuming that the macrofather MF_i at level L is refined into four equal subelements (quadrants) in 2D and eight subelements (octants) in 3D, then the father (the children of MF_i) can be linked by a simple pointer i.e. MF_i at level L for a 2D case $MF_i \rightarrow F_j, F_{j+1}, F_{j+2}, F_{j+3}$ at level $L + 1$. By storing the pointer j at level $L + 1$ the remaining subelement numbers in the quartets or octets can be easily found. Similarly the father to son and son to grandson refinements are also carried out using the same procedure to create a minimal tree data structure. This situation is represented in the Fig. 7.

The data structure facilitates the flow of information for performing the following operations during the course of adaptive mesh refinement.

- To identify the location of an element in the tree and also in the domain.
- To identify element number and its level number so that the geometric characteristic data such as the refined element number, node number, nodal coordinates, node types and constrained nodes can be easily computed.
- To identify and compute the various types of neighbors in 2D and 3D associated with the element in question. This happens due to the enforcement of a *one-level* rule and also due to the recursive refinement.
- Distinguishing the constrained nodes (edge or face) and interpolating for their solutions.

In order to effectively perform various book keeping tasks such as labelling, local and global node numbering and element numbering and computing the geometrical characteristic mesh data of an element, the minimal tree refers to a number of integer data arrays and tables in the data structure. The following is the list of integer data arrays used to store the tree data and other related information.

(i) Element sequence array,
(ii) Node number array,
(iii) Arrays for X, Y and Z coordinates,
(iv) Node type array,
(v) Constrained node array.

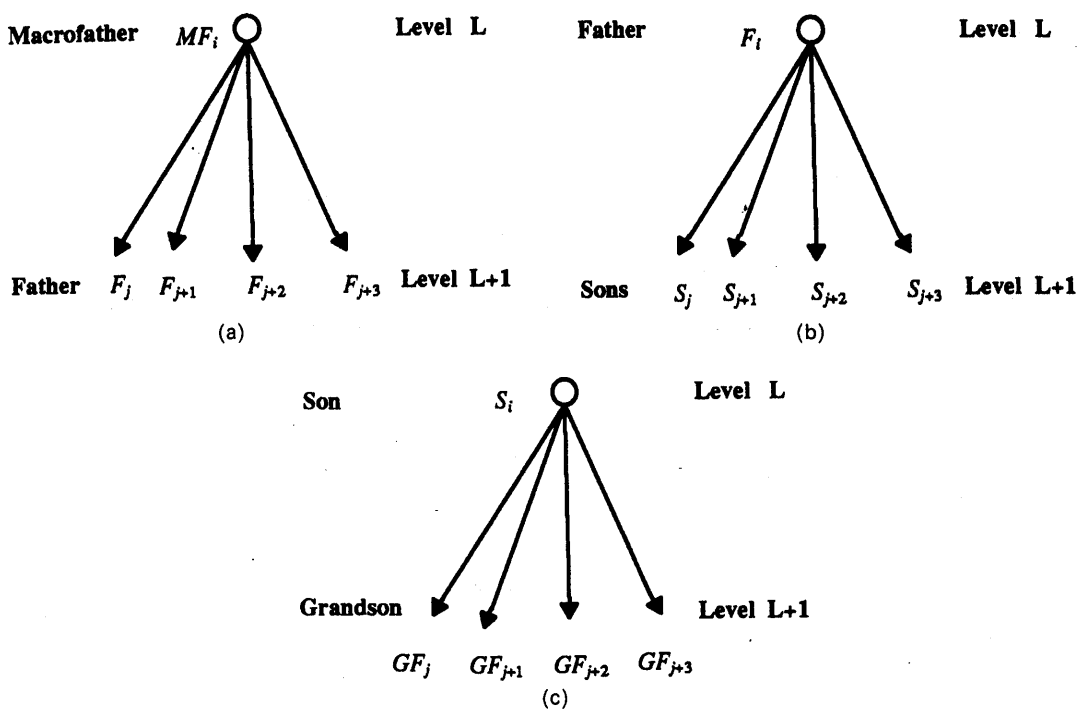


Fig. 7. Components of a minimal tree data structure: (a) Macrofather–Father's; (b) Father–Sons; and (c) Son–Grandsons.

Refinement proceeds by using various arrays of data structure and utilizing a look up table procedure to locally refine an element. This process involves labelling the refined elements, assigning global and local node numbers, keeping track of the nodes, node types and the corresponding nodal coordinates and computing the refinement levels of elements and the corresponding neighbors. Additional information necessary to compute the location of elements, connectivity information and computation of stiffness matrix and assemblage of global matrix and solution techniques are performed by different procedures aided by the data structure.

The technique implemented here starts with an initial coarse mesh data assuming that the coarse mesh is at a refinement level of '0' (macrofather). The solution is obtained and subsequently error estimation parameters are computed. The mesh data along with the solution and the error estimators help to identify the region to be refined in the next phase. The neighbors are identified as the one sharing four edges of an element in 2D and sharing six faces and 12 edges in 3D. In the subsequent refinement levels, based on the difference in the level of refinement, information such as the list of neighbors, node numbers and the corresponding nodal coordinates, constrained nodes, node types like regular nodes, constrained nodes or boundary nodes are computed. Fig. 8 illustrates the various arrays and the index pointers used in the storage organization.

The element sequence array maintains the natural order of element sequence which helps to identify the locations and their corresponding refinement levels of refined and unrefined elements. This is created by assigning a continuous number in all subsequent refinement levels. For instance referring to the tree data structure with different levels in Fig. 6, the labelling procedure to maintain the natural order of element sequence can be represented as shown in Table 1. Fig. 9 shows the minimal tree data structure and its interaction with different integer arrays.

The main problem in the use of tree data structure is that of the amount of tree traversal involved in computing the data necessary for adaptive refinement [1, 3, 6, 41–43]. This has been minimized in this implementation by the use of a minimal tree based data structure. In this approach, although the refinement is hierarchic, the tree does not 'grow' as the refinement proceeds deeper in a particular branch of the tree. By maintaining the connectivity information between the elements and their neighbors and using a natural order of element sequence (element location) and its generation, the algorithm checks only the parent element and the corresponding children at any level of refinement. This means the minimal element tree preserves only two levels at any point in time during refinement. The level may pertain to a macrofather and father or a father and son or a son and a grandson. After refining the parent element, data pertaining to the parent element is lost and the corresponding data arrays are updated by the details of the children elements. By this technique the 'growth' of the tree to different levels is curtailed and the data related to the complete refinement tree need not be stored.

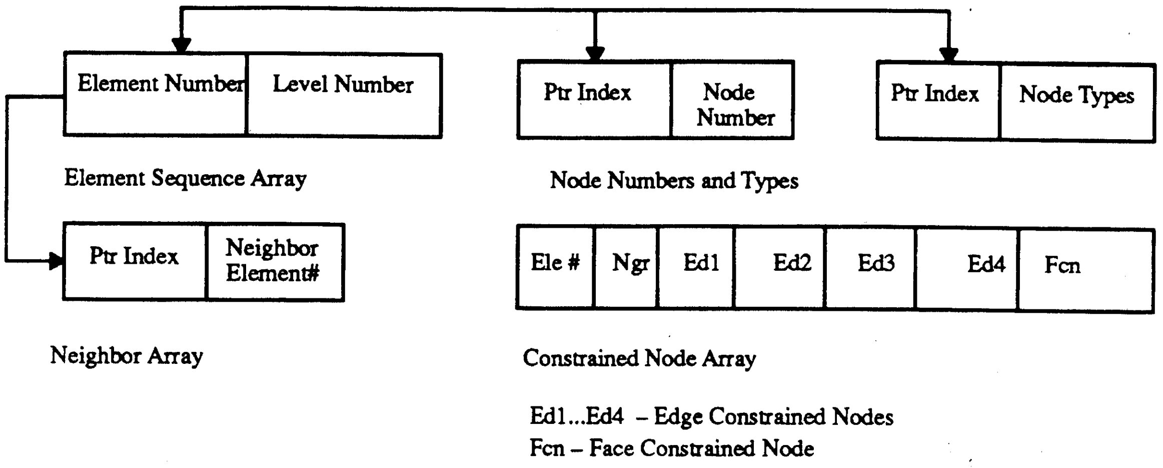


Fig. 8. Integer arrays used in the data structure.

Table 1
Minimal hierarchical data tree and natural order of element sequence

Level number	Family hierarchy	Sequence number
0	Macrofather	1, 2, 3
1	Father	4–11, 12–19, 20–27
2	Son	28–35, 5–11, 12, 36–43, 14–19, 20, 44–51, 22–27
3	Grandson	52–59, 29–35, 5–11, 36–43, 14–19, 20, 44–51, 22–27

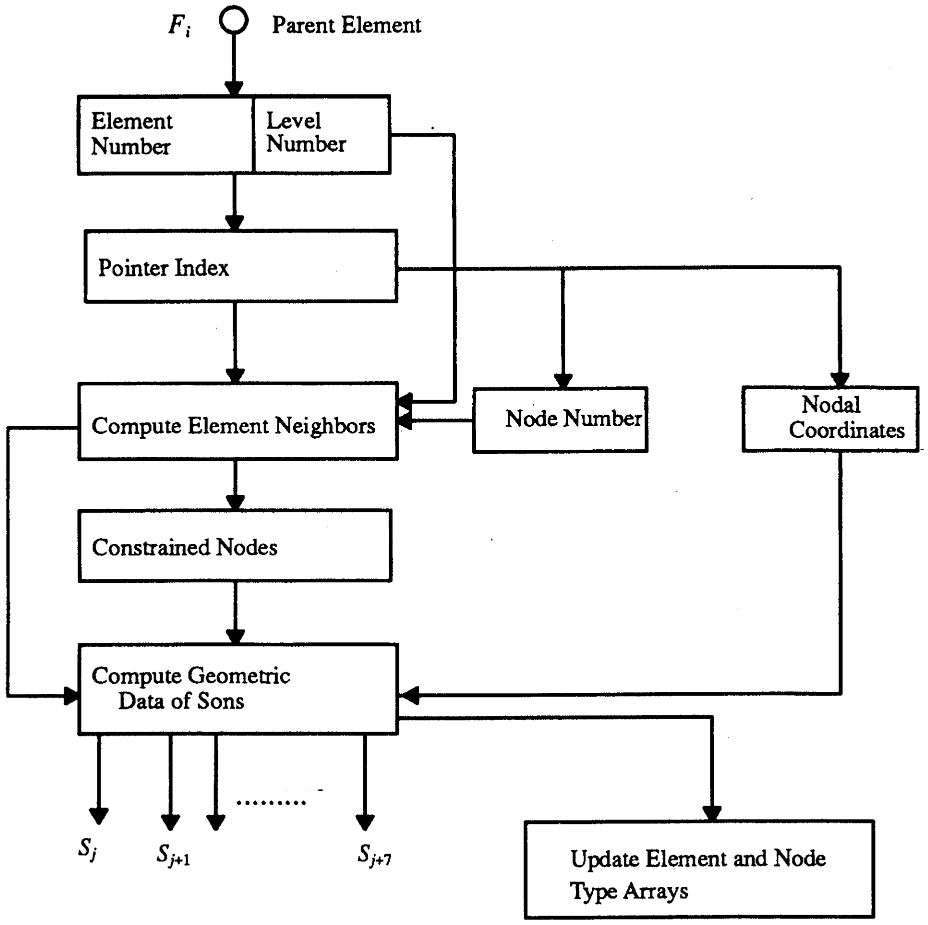


Fig. 9. Functional data flow and interaction in the data structure.

4. Numerical case study and performance analysis

In order to evaluate and verify the efficiency of the refinement algorithm and its associated data structure, a set of self adjoint, elliptic boundary value problems in 2D and 3D have been solved. A classical electrostatic problem with a L-shaped domain having a corner singularity has been chosen for implementation and performance evaluation of the proposed algorithm. The boundary value problem solved in 2D using the *h*-adaptive algorithm is defined in the Fig. 10.

An initial coarse mesh on the L-section with 12 quadrilateral elements is used to initiate the refinement process. As explained earlier, the post-processing and interpolation *a posteriori* error estimation strategy is used to control the refinement process. The error in energy norm and the relative error have been computed locally and globally during each refinement sequence. Fig. 11 shows the initial coarse mesh, intermediate mesh and the final mesh for the 2D test problem. The singularity lies near the re-entrant corner of the problem domain. As expected, it is found that the mesh is denser near the re-entrant corner and is sparser in the remaining areas of the domain. The

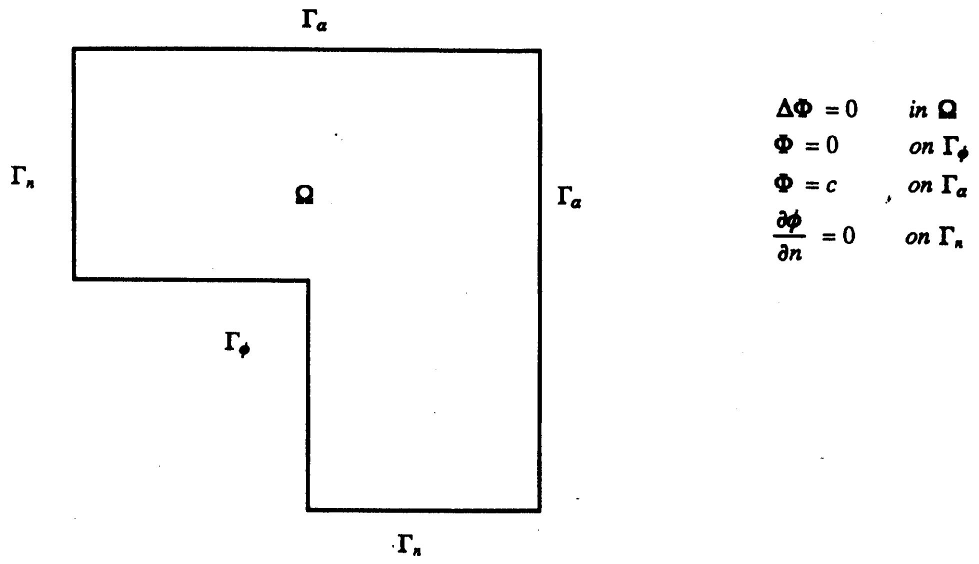

Γ_a
Γ_n
Ω
Γ_a
Γ_ϕ
Γ_n
$\Delta\Phi = 0$ in Ω
$\Phi = 0$ on Γ_ϕ
$\Phi = c$ on Γ_a
$\frac{\partial\phi}{\partial n} = 0$ on Γ_n

Fig. 10. L-shaped domain with corner singularity.

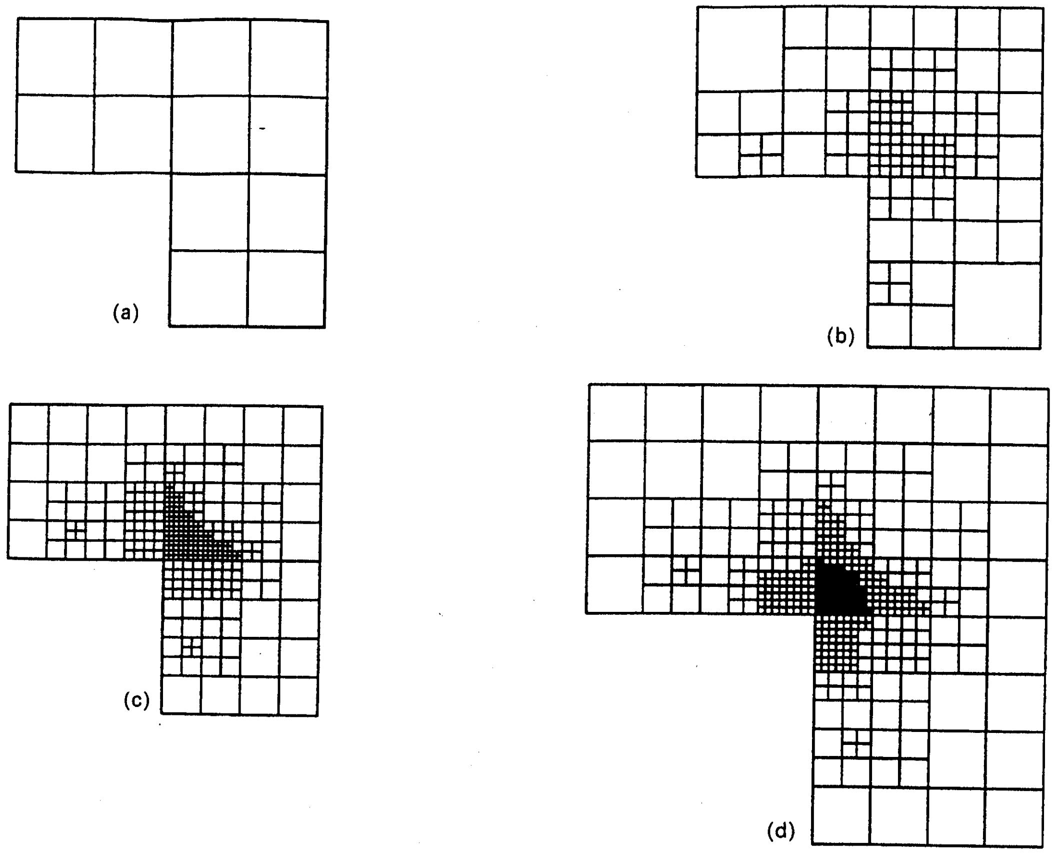


Fig. 11. Sequence of adaptive meshes on an L-shaped domain: (a) initial coarse mesh – 12 elements; (b) third level refinement – 120 elements; (c) fourth level refinement – 306 elements; and (d) fifth level refinement – 534 elements.

solution plot for the corresponding meshes are shown in Fig. 12. The flux plot shows that the equi-potential lines become smoother from one mesh refinement to another indicating an improvement in the accuracy of the solution.

Fig. 13 shows the convergence of the error in energy norm and the local and global relative error convergence as the refinement proceeds. As optimal mesh is one in which the error in energy norm

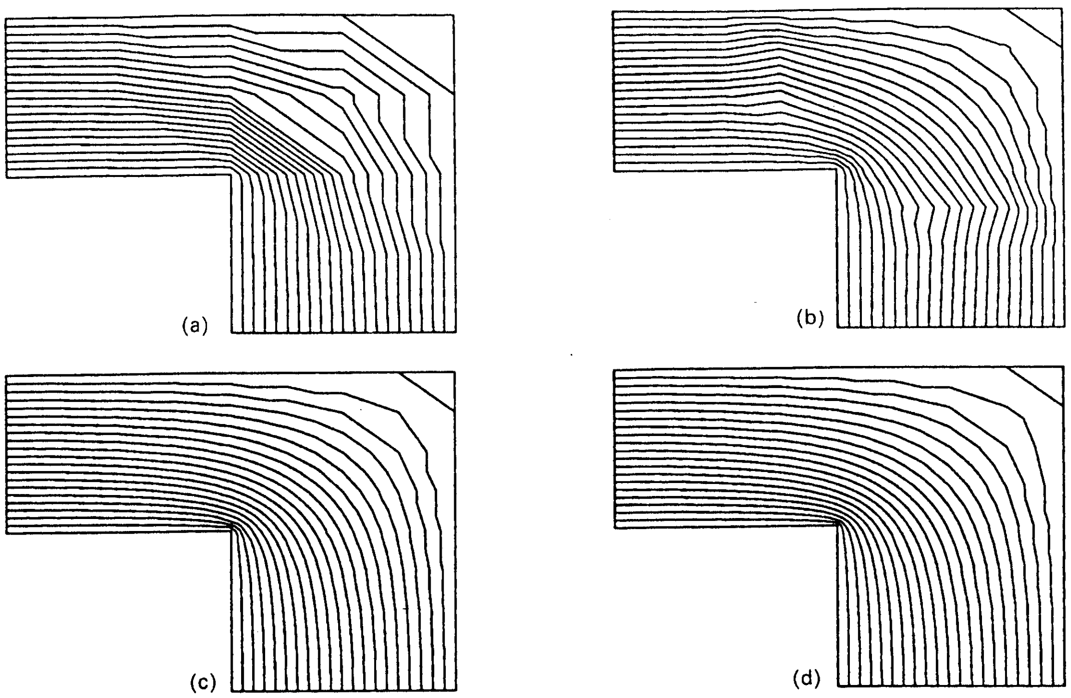


Fig. 12. Adaptive solution plot on an L-shaped domain: (a) coarse mesh – 12 elements; (b) Level 3 – 120 elements; (c) Level 4 – 306 elements; and (d) Level 5 – 534 elements.

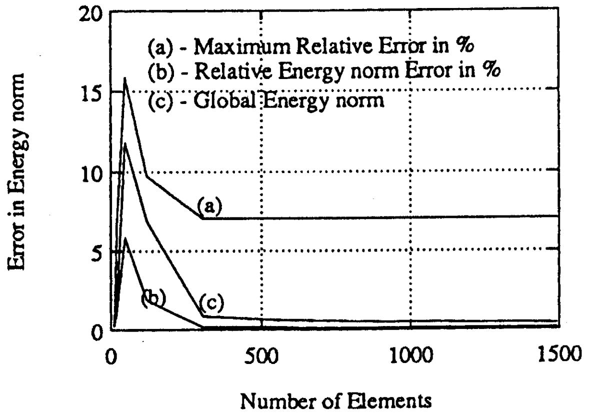


Fig. 13. Error convergence plot.

is equally distributed. Comparing the mesh plot and its corresponding flux plot and the error convergence plot, it can be concluded that an optimal (or nearly optimal) mesh is obtained after four levels of refinement with 306 elements and 369 degrees of freedom. However for the sake of verifying the convergence and the optimality of the mesh, two more levels of refinement have been performed.

The fifth and sixth levels of mesh refinement did not contribute anything significant to improve the solution since the mesh is already optimal (or nearly optimal). The density of elements near the re-entrant corner increases as the refinement proceeds to the higher level,

The 3D mesh refinement algorithm is used to solve for the potential distribution inside a metallic cube. The coarse mesh with $2 \times 2 \times 2$ elements in a cube of $1 \times 1 \times 1$ units is used to initiate the adaptive refinement. The sequence of meshes generated during the refinement is shown in Fig. 14.

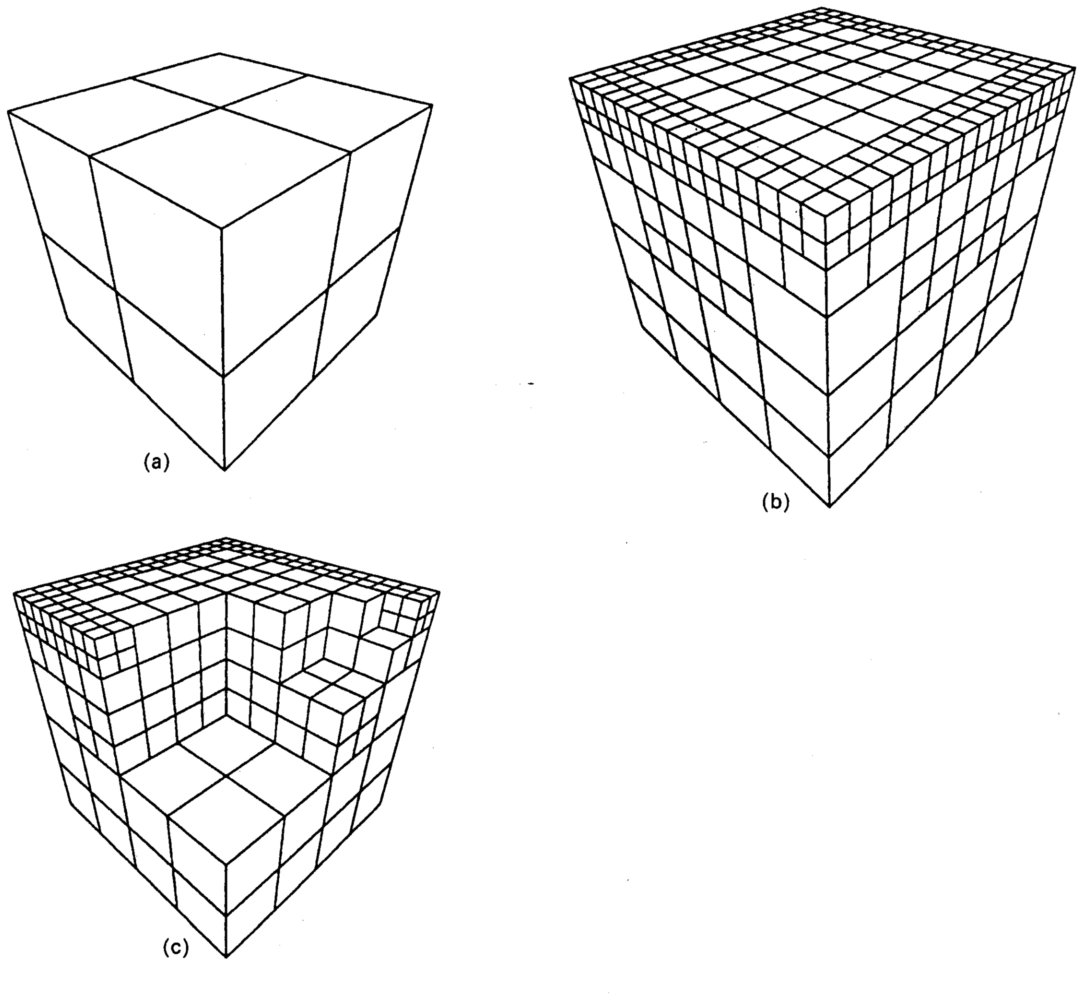


Fig. 14. Potential distribution inside a metallic cube – sequence of 3D adaptive meshes: (a) initial coarse mesh with $2 \times 2 \times 2$ elements; (b) Level 3 meshes with 456 elements; and (c) cut out view of the Level 3 mesh.

Since there is more error on the upper plane of the cube due to the discontinuity, it is found that the refinement concentrates on the upper portion of the domain compared to other regions. Fig. 14(c) represents the cross sectional view of the domain representing the three dimensional adaptive refinement in the interior of the problem domain.

5. Conclusion

A flexible and efficient algorithm for the computation of 2D and 3D adaptive mesh refinement is proposed and implemented. A simple *a posteriori* error estimation using post-processing and interpolation techniques is used to guide the mesh refinement process. The hierarchical mesh refinement algorithm is implemented by imposing a *one-level* rule. The algorithm reduces the amount of tree traversal by using a minimal tree based data structure and thus a trade-off between the storage and computation is achieved. The organization of the dynamic data structure and its functional interaction for data flow to facilitate the refinement algorithm is also analyzed. A numerical case study is carried out by implementing the algorithm and data structure to solve linear elliptic boundary value problems in electrostatics. The numerical test results and the sequence of adaptive meshes verify the application potential of the proposed algorithm and the data structure for generating two and three dimensional adaptive meshes.

References

[1] M.C. Rivara, "Design and data structure of fully adaptive, multigrid, finite element software", *ACM Trans. Math. Software* **10** (3), pp. 242–264, 1984.

[2] W.C. Rheinboldt and C.K. Mesztenyi, "On a data structure for adaptive finite element mesh refinements", *ACM Trans. Math. Software* **6** (2), *pp.* 166–187, 1980.

[3] G.F. Carey, M. Sharma and K.C. Wang, "A Class of Data Structures for 2-D and 3-D Adaptive Mesh Refinement", *Int. J. Numer. Methods Eng.* **26**, pp. 2607–2622, 1988.

[4] E.K. Buratynski, "A fully automatic three dimensional mesh generator for complex geometries", *Int. J. Numer. Methods Eng.* **30**, pp. 931–952, 1990.

[5] S.H. Lo, "Automatic mesh generation and adaptation by using contours", *Int, J. Numer. Methods Eng.* **31**, pp. 689–707, 1991.

[6] L. Demkowicz, J.T. Oden, W. Rachowicz and O. Hardy, "Toward a universal *h–p* adaptive finite element strategy, part 1. Constrained approximation and data structure", *Comput. Methods Appl. Mech. Eng.* **77**, pp. 79–112, 1989.

[7] W.F. Mitchell, "A comparison of adaptive refinement techniques for elliptic problems", *ACM Trans. Math. Software* **15** (4), pp. 326–347, 1989.

[8] H. Shahnasser, W. Morgan and A. Raghuram, "A dynamic data structure suitable for adaptive mesh refinement in finite element method", *Finite Elements in Analysis and Design* **4**, pp. 237–247, 1988.

[9] A. Raizer, G. Meunier and J.L. Coulomb, "An approach for automatic adaptive mesh refinement in finite element computation of magnetic fields", *IEEE Trans. Magnetics* **25** (4), pp. 2965–2967, 1989.

[10] Z.J. Cendes, D.N. Shenton and H. Shahnasser, "Magnetic field computation using Delaunay triangulation and complementary finite element methods", *IEEE Trans Magnetics* **19** (6), pp. 2551–2554, 1983.

[11] J. Penman and M.D. Grieve, "An approach to self adaptive mesh generation", *IEEE Trans. Magnetics* **21** (6), pp. 2567–2570, 1985.

[12] D.T. Lee and B.J. Schachter, "Two algorithms for constructing Delaunay triangulation", *Int. J. Comput. Inform. Sci.* **9**, pp. 219–241, 1980.

[13] D.C. Arney and J.E. Flaherty, "An adaptive mesh-moving and local refinement method for time dependent partial differential equations", *ACM Trans. Mathematical Software* **16** (1), pp. 48–71, 1990.
[14] M.K. Georges and M.S. Shephard, "Automated adaptive two dimensional system for *hp* version of the finite element Method", *Int. J. Numer. Methods Eng.* **32**, pp. 783–810, 1991.
[15] M. Okabe, "One-dimensional self-adaptive interpolations in the *p*-convergence procedure", *Comput. Methods Appl. Mech. Eng.* **77**, pp. 69–89, 1983.
[16] J.J. Rencis and K.Y. Jong, "A self-adaptive *h*-refinement technique for the boundary element method", *Comput. Methods Appl. Mech. Eng.* **73**, pp. 295–316, 1989.
[17] L. Demkowicz, Ph. Devloo and J.T. Oden, "On an *h*-type mesh refinement strategy based on minimization of interpolation errors", *Comput. Methods Appl. Mech. Eng.* pp 67–89, 1985.
[18] W. Rachowicz, J.T. Oden and L. Demkowicz, "Toward a universal *h–p* adaptive finite element strategy part 3, Design of *h–p* meshes", *Comput. Methods Appl. Mech. Eng.* **77**, pp. 181–212, 1989.
[19] J.T. Oden, L. Demkowicz, W. Rachowicz and T.A. Westermann, "Toward a universal *h–p* adaptive finite element strategy Part 2. A posteriori error estimation", *Comput. Methods Appl. Mech. Eng.* **77** , pp. 113–180, 1989.
[20] M.A. Yerry and M.S. Shephard, "Automatic three-dimensional mesh generation by the modified -octree technique", *Int. J. Numer Methods Eng.* **20**, pp. 1965–1990, 1984.
[21] I. Babuska and D. Yu, "Asymptotically exact a posteriori error estimator for biquadratic elements", *Finite Elements in Analysis and Design* **3**, pp. 341–354, 1987.
[22] I. Babuska and M. Suri, "The optimal convergence rate of the *p*-version of the finite element method", *SIAM J. Numer. Anal.* **24** (4), pp. 750–776, 1987.
[23] P. Fernandes, P. Girdinio, P. Molfino and M. Repetto, "A comparison of adaptive strategies for mesh refinement based on a posteriori local error estimation procedures", *IEEE Trans. Magnetics* **26**, pp. 795–798, 1990.
[24] S.Y. Hahn, C. Calmels, G. Meunier and J.L. Coulomb, "A posteriori error estimate for adaptive finite element mesh generation", *IEEE Trans. Magnetics* **24** (1), pp. 315–317, 1988.
[25] M.S. Shephard, "Automatic and adaptive mesh generation", *IEEE Trans. Magnetics* **21** (6), pp. 2484–2489, 1985.
[26] B.N. Jiang and G.F. Carey, "Adaptive refinement for least squares finite elements with element-by-element conjugate gradient solution", *Int. J. Numer. Methods Eng.* **24**, pp. 569–580, 1987.
[27] O.C. Zienkiewicz and J.Z. Zhu, "Adaptivity and mesh generation", *Int. J. Numer. Methods. Eng.* **32**, pp. 783–810, 1991.
[28] J.P. De, S.R. Gago, D.W. Kelly and O.C. Zienkiewicz, "A-posteriori error analysis and adaptive processes in the finite element method: Part II – Adaptive mesh refinement", *Int. Numer. Methods Eng.* **19**, pp. 1621–1656, 1983.
[29] D.W. Kelly, J.P. De, S.R. Gago and O.C. Zienkiewicz, "A-posteriori error analysis and adaptive processes in the finite element method: part I – Error analysis", *Int. J. Numer. Methods Eng.* **19**, pp. 1593–1619, 1983.
[30] A.K. Noor and I. Babuska, "Quality assessment and control of finite element solutions", *Finite Elements in Analysis and Design* **3**, pp. 1–26, 1987.
[31] F. Schmoellebeck and H. Hass, "3-dimensional, adaptive FE-preprocessing using a programmable CAD-system", *IEEE Trans. Magnetics* **24** (1), pp. 370–373, 1988.
[32] D.N. Shenton and Z.J. Cendes, "Max–an expert system for automatic adaptive magnetics modelling", *IEEE Trans. Magnetics* **22** (5), pp. 805–807, 1986.
[33] D.N. Shenton and Z.J. Cendes, "Three dimensional finite element mesh generation using Delaunay tessellation", *IEEE Trans. Magnetics* **21** (5), pp. 2535–2538, 1985.
[34] A.R. Pinchuk and P.P. Silvester, "Error estimation for automatic adaptive finite element mesh generation", *IEEE Trans. Magnetics* **21** (6), pp. 2551–2554, 1985.
[35] Z.J. Cendes and D.N. Shenton, "Adaptive mesh refinement in the finite element computation of magnetic fields", *IEEE Trans. Magnetics* **21** (5), pp. 1811–1815, 1985.
[36] S. Biddlecombe, J. Simkin and C.W. Trowbridge, "Error analysis in finite element models of electromagnetic fields", *IEEE Trans. Magentics* **22** (5), pp. 811–813, 1986.
[37] C.S. Koh, K. Choi, S.Y. Hahn, H.K. Jung and K.S. Lee, "An adaptive finite element scheme using multi-grid method for magnetostatic problems", *IEEE Trans. Magnetics* **25** (4), pp. 2959–2961, 1989.
[38] W. Nehl and D.A. Field, "Adaptive refinement of first order tetrahedral meshes for magnetostatics using local Delaunay subdivisions", *IEEE Trans. Magnetics* **27** (5), pp. 4193–4196, 1991.

[39] K. Ang and S. Valliappan, "Mesh grading technique using modified isoparametric shape functions and its application to wave propagation problems", *Int. J. Numer. Methods Eng.* **23**, pp. 331–348, 1986.
[40] O.C. Zienkiewicz and J.Z. Zhu, "Simple error estimator and adpative procedure for practical engineering analysis", *Int. J. Numer. Methods Eng.* **24**, pp. 337–357, 1987.
[41] M.A. Yerry and M.S. Shephard, "A modified quadtree approach to finite element mesh generation", *IEEE Comput. Graphics and Appl.* **3**, pp. 39–46, 1983.
[42] M.S. Shephard, M.A. Yerry and P.L. Baehmann, "Automatic mesh generation allowing for efficient a priori and a posteriori mesh refinement", *Comput. Methods Appl. Mech. Eng.* pp. 161–180, 1986.
[43] R. Perucchio, M. Saxena and A. Kela, "Automatic mesh generation from solid models based on recursive spatial decompositions", *Int. J. Numer. Methods Eng.* **28**, pp. 2469–2501, 1989.
[44] M. Shpitalni, P.B. Yoseph and Y. Krimberg, "Finite element mesh generation via switching function representation", *Finite Elements in Analysis and Design* **5**, pp. 119–130, 1989.
[45] H. Seok, S.P. Hong, K. Choi, H.K. Jung and S.Y. Hahn, "A three dimensional adaptive finite element for magnetostatic problems", *IEEE Trans. Magnetics* **27** (5), pp. 4081–4084, 1991