Computational Mechanics 23 (1999) 361–388 © Springer-Verlag 1999

A methodology for adaptive finite element analysis: Towards an integrated computational environment

G. H. Paulino, I. F. M. Menezes, J. B. Cavalcante Neto, L. F. Martha

Abstract This work introduces a methodology for self-adaptive numerical procedures, which relies on the various components of an integrated, object-oriented, computational environment involving pre-, analysis, and post-processing modules. A basic platform for numerical experiments and further development is provided, which allows implementation of new elements/error estimators and sensitivity analysis. A general implementation of the Superconvergent Patch Recovery (SPR) and the recently proposed Recovery by Equilibrium in Patches (REP) is presented. Both SPR and REP are compared and used for error estimation and for guiding the adaptive remeshing process. Moreover, the SPR is extended for calculating sensitivity quantities of first and higher orders. The mesh (re-)generation process is accomplished by means of modern methods combining quadtree and Delaunay triangulation techniques. Surface mesh generation in arbitrary domains is performed automatically (i.e. with no user intervention) during the self-adaptive analysis using either quadrilateral or triangular elements. These ideas are implemented in the Finite Element System Technology in Adaptivity (FESTA) software. The effectiveness and versatility of FESTA are demonstrated by representative numerical examples illustrating the interconnections among finite element analysis, recovery procedures, error estimation/adaptivity and automatic mesh generation.

Key words finite element analysis, error estimation, adaptivity, h-refinement, sensitivity, superconvergent patch recovery (SPR), recovery by equilibrium in patches (REP), object oriented programming (OOP), interactive computer graphics.

1 Introduction

This work presents an integrated (object-oriented) computational environment for self-adaptive analyses of generic two-dimensional (2D) problems. This environment includes analysis procedures to insure a given level of accuracy according to certain criteria, and also the procedures to generate and modify the finite element discretization. This computational system, called FESTA (Finite Element System Technology in Adaptivity), involves five main components (see shaded boxes in Figure 1):

- A graphical preprocessor, for defining the geometry of the problem, the initial finite element mesh (together with boundary conditions), and the main parameters used in a self-adaptive analysis. Here the geometrical model is dissociated from the finite element model.
- A finite element module for solving the current boundary value problem. The code has been developed so that it is highly modular, expandable, and user-friendly. Thus, it can be properly maintained and continued. Moreover, other users/developers should be able to modify the basic programming system to fit their specific applications.
- An error estimation and sensitivity module. Discretization errors are estimated according to available recovery procedures, e.g. Zienkiewicz and Zhu (ZZ), superconvergent patch recovery (SPR) and recovery by equilibrium in patches (REP). Sensitivities of various orders (1st., 2nd. or higher) are calculated by means of a procedure analogous to the SPR. The user chooses the desired error estimator and sensitivity order.
- A mesh (re-)generation (rather than mesh enrichment) procedure, based on the combination of quadtree and Delaunay triangulation techniques. According to the magnitude of the error, calculated in the previous module, a new finite element mesh is automatically generated (i.e. with no user intervention), using either triangular or quadrilateral elements (h-refinement).

G.H. Paulino
Department of Civil and Environmental Engineering,
University of Illinois at urbana-champaign
2209 Newmark Laboratory,
205 North Mathews Avenue,
Urbana, IL 61801-2352, U.S.A.

I.F.M. Menezes, J.B. Cavalcante Neto, L.F. Martha
TeCGraf (Computer Graphics Technology Group), PUC-Rio,
Rio de Janeiro, R.J., 22453-900, Brazil

J.B. Cavalcante Neto, L.F. Martha
Department of Civil Engineering, PUC-Rio,
Rio de Janeiro, R.J., 22453-900, Brazil

Correspondence to: G.H. Paulino

G.H. Paulino acknowledges the support from the United States National Science Foundation (NSF) under Grant No. CMS-9713798. I.F.M. Menezes acknowledges the financial support provided by the FAPERJ, which is a Brazilian agency for research and development in the state of Rio de Janeiro. G.H. Paulino and I.F.M. Menezes also acknowledge the Department of Civil and Environmental Engineering at UC-Davis for hospitality while part of this work was performed. J.B. Cavalcante Neto and L.F. Martha acknowledge the financial support provided by the Brazilian agency CNPq. The authors also thank an anonymous reviewer for providing relevant suggestions to this work.

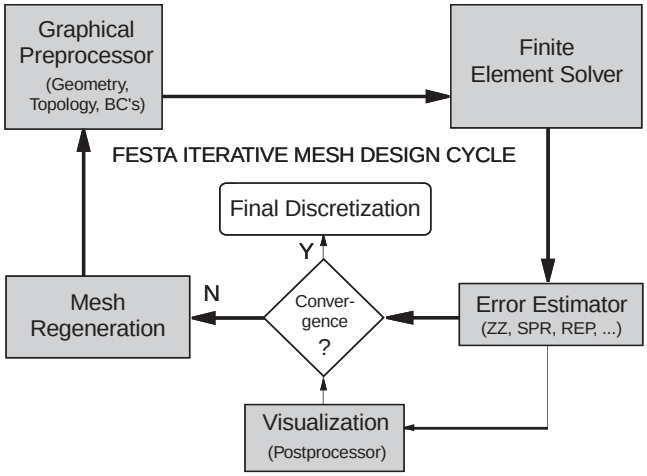

Fig. 1. Simplified diagram of the FESTA interactive meshing

- Finally, a postprocessor module, where all the analysis results (e.g. deformed shape, sensitivity and stress contours) can be visualized.

Essentially, FESTA is a computational laboratory which offers a basic platform for numerical analysis and further development, e.g. implementation of new error estimators, elements, or material models (Cavalcante Neto et al. 1998). Object-oriented programming and integration of pre-, analysis, and post-processing modules make FESTA a software well-suited for both practical engineering applications and further research development.

The remainder of this paper is organized as follows. A motivation to the work and a "brief" literature review are provided in Sect. 2. Afterwards, Sect. 3 presents some theoretical background on self-adaptive simulations and an overview of the graphical interface used in the FESTA software. Section 4 introduces the mathematical formulation of the SPR (using weighted least square systems), the REP, and the sensitivity method. A discussion about the automatic mesh generation techniques used in this work is given in Sect. 5. Relevant information regarding the implementation of FESTA is presented in Sect. 6, especially aspects related to the SPR and REP recoveries. In order to assess the effectiveness of the proposed computational system, representative numerical examples are given in Sect. 7. Finally, in Sect. 8, conclusions are inferred and directions for future research are discussed.

2
Motivation and related work

Normal practice to solve engineering problems by means of the Finite Element Method (FEM) or the Boundary Element Method (BEM) involves increasing the number of discretization points in the computational domain and resolving the resulting system of equations to examine the relative change in the numerical solution. In general, this procedure is time consuming, it depends on the experience of the analyst, and it can be misleading if the solution has not entered an asymptotic range.

Ideally, with a robust and reliable self-adaptive scheme, one would be able to specify an initial discrete model which is sufficient to describe the geometry/topology of the domain and the boundary conditions (BCs), and to specify a desired error tolerance, according to an appropriate criterion. Then, the system would automatically refine the model until the error measure falls below the prescribed tolerance. The process should be fully automatic and without any user intervention. This is the main goal which motivated the development of FESTA. This approach increases the overall reliability of the analysis procedure since it does not depend on the experience, or inexperience, of the analyst.

The need for developing better pre-processing techniques for the FEM, for performing automated analysis, and for obtaining self-adaptive solutions (which is becoming a trend for commercial FEM software) have driven the development of automatic mesh generation algorithms, i.e. algorithms which are capable of discretizing an arbitrary geometry into a consistent finite element mesh without any user intervention. Several algorithms for 2D geometries have been developed (e.g. Baehmann et al. 1987; Blacker and Stephenson 1991; Zhu et al. 1991; Potyondy et al. 1995b; Borouchaki and Frey 1998), and approaches for three-dimensional (3D) geometries have appeared more recently (e.g. Cass et al. 1996; Escobar and Montenegro 1996; Beall et al. 1997; Lo 1998). The present work focus on automatic 2D mesh generation in connection with adaptive solutions. Efficient techniques for generating all-quadrilateral and all-triangular meshes are considered in detail. Although the algorithms presented herein could be extended to mixed meshes, i.e. meshes with both triangular and quadrilateral elements (see, for example, Borouchaki and Frey 1998), this topic is not within the scope of this work.

There exist a vast literature on error estimation and adaptivity, and the reader is directed to the appropriate references[1]. The volumes edited by Brebbia and Aliabadi (1993) and Babuška et al. (1986) review adaptive techniques for the FEM and the BEM. The book edited by Ladevèze and Oden (1998) presents a compilation of papers from the workshop of "New Advances in Adaptive Computational Mechanics," held at Cachan, France, 17–19 September 1997, which dealt with the latest advances in adaptive methods in mechanics and their impact on solving engineering problems. Several issues of journals have also been dedicated to adaptivity, e.g. volume 12 (1996), number 2 of Engineering with Computers, volume 15 (1992), numbers 3/4 of Advances in Engineering Software, and volume 36 (1991), number 1 of the Journal of Computational and Applied Mathematics. Surveys of the literature in FEM include articles by Noor and Babuška (1987), Oden and Demkovicz (1989), Strouboulis and Haque (1992a, b), Babuška and Suri (1994), and Ainsworth and Oden (1997). Mackerle (1993, 1994) has compiled a long list of references on mesh generation, refinement, error analysis and adaptive techniques for FEM and BEM that were published from 1990 to 1993. The

[1] The list of papers referred here is just a small sampling of the literature, considering articles of particular interest to the present work, and is not intended to be a representative survey of the literature in the field.

volume edited by Babuška et al. (1983) presents adaptive techniques for the FEM and the Finite Difference Method (FDM). Relatively recent textbooks in the FEM emphasize the field of adaptive solution techniques. For example, the book by Zienkiewicz and Taylor (1989) includes a Chapter on "Error Estimation and Adaptivity" (Chapter 14), which is supplemented by the papers by Zienkiewicz and Zhu (1992a, b, 1994). Moreover, the book by Szabó and Babuška (1991) is primarily dedicated to this subject.

The first papers on adaptive finite elements appeared in the early seventies. Since then, an explosive number of papers on the subject have been published in the technical literature. Babuška and Rheinboldt (1978) presented a pioneering paper about error estimates by evaluating the residuals of the approximate solution and using them to obtain local, more accurate answers. They developed the mathematical basis of self-adaptive techniques. With the concept of a posteriori error estimates, one can develop a self-adaptive strategy for the FEM such that only certain elements should be refined. Zienkiewicz et al. (1982) presented a hierarchical approach for self-adaptive methods. In the early 1980s, computer graphics techniques started to be used as standard tools by mesh generation programs. Shephard (1986) published a paper where geometric modeling and automatic mesh generation techniques were used in conjunction with self-adaptive methods. Zienkiewicz and Zhu (1987) introduced an error estimator based on obtaining improved values of gradients (stresses) using some available recovery processes. Easy to be implemented in any finite element code, this type of technique, based on averaging and on the so called L_2 projection, has been used to recover the gradients, and reasonable estimators were achieved. In 1992, this technique was corrected/improved by the same authors, leading to the so called Superconvergent Patch Recovery: SPR (Zienkiewicz and Zhu 1992a, b, 1994). This method is a stress-smoothing procedure over local patches of elements and is based on a discrete least-squares fit of a higher-order local polynomial stress field to the stresses at the superconvergent sampling points obtained from the finite element calculation. Attempts to improve further the recovery process can be found in various references, e.g. Wiberg et al. (1994), Wiberg and Abdulwahab (1993), Blacker and Belytschko (1994), Tabbara et al. (1994), and Lee et al. (1997). Essentially these improved techniques incorporate equilibrium and boundary conditions on the recovery process. An exhaustive study by Babuška et al. (1994a, b) showed, through numerous examples, the excellent performance and superiority of the SPR over residual-type approaches. Recently, Boroomand and Zienkiewicz (1997) have presented a new super-convergent method satisfying the equilibrium condition in a weak form, which does not require any knowledge of superconvergent points. The new recovery technique has been called Recovery by Equilibrium in Patches: REP. Both SPR and REP are of particular interest to the present work.

As indicated above, the general field of adaptivity is broad and has advanced significantly in recent years. For instance, Paulino et al. (1997) have proposed a new class of error estimates based on the concept of nodal sensitivities, which can be used in conjunction with general purpose computational methods such as FEM, BEM or FDM. Rannacher and Suttmeier (1997) have suggested a feedback approach for error control in the FEM. Mahomed and Kekana (1998) have presented an adaptive procedure based on strain energy equalisation. Moreover, a summary of recent advances in adaptive computational mechanics can be found in the book edited by Ladevèze and Oden (1998).

Quantification of the quality of a model with respect to another one, taken as the reference, is of primary importance in numerical analysis. This is the case with well established methods, such as the FEM, or emerging methods, such as the element free Galerkin: EFG (Chung and Belytschko 1998), the symmetric Galerkin BEM (Paulino and Gray 1999), or the boundary node method (Mukherjee and Mukherjee 1997). Integration of concepts regarding error estimation and adaptivity in the FEM, within a modern computational environment, is the focus of the present work.

3
Theoretical and computational aspects

Whenever a numerical method is used to solve the governing differential equations of a boundary value problem, error is introduced by the discretization process which reduces the continuous mathematical model to one having a finite number of degrees of freedom. The discretization errors are defined as the difference between the actual solution and its numerical approximation. By definition, the local error

$$e = \phi - \hat{\phi} \tag{1}$$

is a measure of the difference between the exact (ϕ) and an approximate solution ($\hat{\phi}$). Here, ϕ is analogous to a response quantity (e.g. displacements) in a typical numerical solution procedure.

Self-adaptive methods are numerical schemes which automatically adjust themselves in order to improve the solution of the problem to a specified accuracy. The two basic components in adaptive methods are error estimation and adaptive strategy. These components are discussed below.

In general, there are two types of discretization error estimates: a priori and a posteriori. Although a priori estimates are accurate for the worst case in a particular class of solutions of a problem, they usually do not provide information about the actual error for a given model. A posteriori estimates use information obtained during the solution process, in addition to some a priori assumptions about the solution. A posteriori estimates, which can provide quantitatively accurate measures of the discretization error, have been adopted here.

In the context of adaptive strategies, extension methods have been preferred over others approaches (e.g. dual or complementary methods) and are the focus of this work. These methods include *h*-, *p*-, and *r*-extensions. The computer implementation is referred to as the *h*-, *p*-, and *r*-versions, respectively. In the *h*-extension, the mesh is automatically refined when the local error indicator ex-

ceeds a preassigned tolerance. The *p*-extension generally employs a fixed mesh. If the error in an element exceeds a preassigned tolerance, the local order of the approximation is increased to reduce the error. The *r*-extension (node-redistribution) employs a fixed number of nodes and attempts to dynamically move the grid points to areas of high error in the mesh. Any of these extensions can also be combined in a special strategy, for example, *h*-*p*-, *r*-*h*-, among others.

3.1
Error estimation and adaptive refinement

As pointed out by several authors (e.g. Zienkiewicz and Taylor 1989), the specification of local error in the manner given in Eq. (1) is generally not convenient and occasionally misleading. Thus mathematical norms are introduced to measure the discretization error. The exact discretization error in the finite element solution is often quantified on the basis of the energy norm for the displacement error, $||e||$, which can be expressed, in terms of stresses, as

$$||e||^2 = \int_\Omega (\boldsymbol{\sigma}_{ex} - \hat{\boldsymbol{\sigma}})^T \mathbf{D}^{-1} (\boldsymbol{\sigma}_{ex} - \hat{\boldsymbol{\sigma}})\, d\Omega \tag{2}$$

where $\boldsymbol{\sigma}_{ex}$ and $\hat{\boldsymbol{\sigma}}$ are the exact and the finite element stress fields, $\mathbf{D}$ is the constitutive matrix, and Ω is the problem definition domain.

The basic idea of error estimators is to substitute the field $\boldsymbol{\sigma}_{ex}$, which is generally unknown, by the field $\bar{\boldsymbol{\sigma}}$, obtained by means of recovery procedures (e.g. ZZ, SPR or REP). Therefore, the expression for computing the approximate (estimated) relative error distribution $||e||_{es}$ can be expressed as

$$||e||_{es}^2 = \int_\Omega (\bar{\boldsymbol{\sigma}} - \hat{\boldsymbol{\sigma}})^T \mathbf{D}^{-1} (\bar{\boldsymbol{\sigma}} - \hat{\boldsymbol{\sigma}})\, d\Omega \tag{3}$$

Taking into account the finite element discretization and considering a specific finite element i, Eq. (3) can be rewritten as

$$\left(||e||_{es}^i\right)^2 = \int_{\Omega_i} (\bar{\boldsymbol{\sigma}} - \hat{\boldsymbol{\sigma}})^T \mathbf{D}^{-1} (\bar{\boldsymbol{\sigma}} - \hat{\boldsymbol{\sigma}}) |\mathbf{J}|\, d\Omega_i \tag{4}$$

where standard isoparametric elements have been assumed; $\bar{\boldsymbol{\sigma}}$ denotes the recovered stress field, $|\mathbf{J}|$ is the determinant of the Jacobian transformation matrix, and Ω_i is the element domain.

The energy norm for the error can be evaluated over the whole domain or part of it. The contribution of all the elements in the mesh is given by

$$||e||^2 = \sum_{i=1}^{m} \left(||e||^i\right)^2 \tag{5}$$

where m is the total number of elements, i refers to the element with domain Ω_i, and $\cup_{i=1}^m \Omega_i = \Omega$.

The relative percentage error in the energy norm (η_{ex}) for the whole domain or part of it can be obtained as

$$\eta_{ex} = \frac{||e||_{ex}}{||\mathscr{U}||_{ex}} \tag{6}$$

where $||\mathscr{U}||_{ex}$ is the square root of twice the strain energy, and it is given by

$$||\mathscr{U}||_{ex} = \left(\int_\Omega \boldsymbol{\sigma}^T \mathbf{D}^{-1} \boldsymbol{\sigma}\, d\Omega\right)^{1/2} \tag{7}$$

The adaptive refinement strategy (*h*-extension) is discussed next. The error estimator will define how the discretization model will be refined or coarsened. A simple criterion to achieve a solution error with an acceptable level for the whole domain can be stated as

$$\eta_{es} \leq \eta_{max} \tag{8}$$

where η_{max} is the maximum permissible error, and η_{es} is given by

$$\eta_{es} = \frac{||e||_{es}}{\left(||\mathscr{U}_h||^2 + ||e||_{es}^2\right)^{1/2}} \tag{9}$$

where $||\mathscr{U}_h||$ is the energy norm obtained from the finite element solution.

A sound criterion for an "optimal mesh" consists of requiring that the energy norm error be equidistributed among elements because it leads to meshes with high convergence rates. Thus, for each element i,

$$||e||_{es}^i < \eta_{max} \left(\frac{||\mathscr{U}_h||^2 + ||e||_{es}^2}{m}\right)^{1/2} = \bar{e}_m \tag{10}$$

By defining the ratio

$$\xi_i = \frac{||e||_{es}^i}{\bar{e}_m} \tag{11}$$

it is obvious that refinement is needed if

$$\xi_i > 1.0 \tag{12}$$

A more efficient procedure, which is adopted here, consists of designing a completely new mesh (re-generation) which satisfies the requirement

$$\xi_i \leq 1.0 \tag{13}$$

in the limit of mesh refinement. By assuming a certain rate of convergence (Zienkiewicz and Taylor 1989), the value of the error ratio ξ_i can be used to decide the new size of the element. Thus

$$h = \frac{h_i}{\xi_i^{1/p}} \tag{14}$$

where h_i is the initial size of the element, h is the final size, and p is the polynomial order of the approximation. This procedure does not account for singularity effects, such as those generated by the sharp corners and cracks. A treatment of singular behavior, with varying degree of sophistication, can be found in Zienkiewicz and Taylor (1989), Szabó and Babuška (1991), Babuška et al. (1994), and Coorevits et al. (1994).

The ratio error (Eq. (11)) is implemented in the numerical analysis module, and it is exported to the self-adaptive module of FESTA, where new element sizes are calculated to satisfy the error criterion. This is done

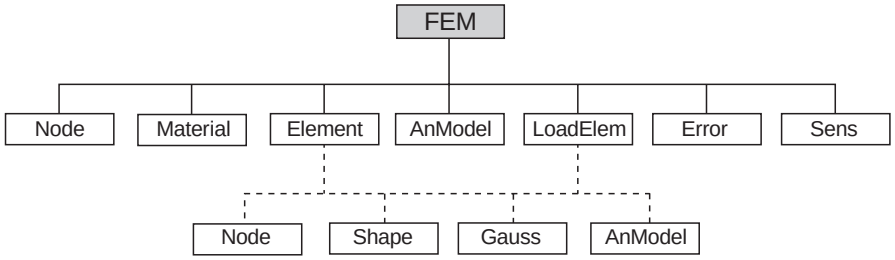

Fig. 2. FESTA OOP class organization

through the *Element class* of the self-adaptive module, which by means of OOP, permits various element types to be treated in the same framework of analysis, i.e. the method is not restricted to a specific element. The determination of the new element size is used either to refine or to coarsen the mesh where necessary (mesh re-generation).

3.2 FESTA computational interface

In order to obtain an efficient and robust computational system for self-adaptive simulations, a generic treatment for dealing with different finite elements (therefore, different shapes and different degrees of interpolation functions) has to be taken into account. For this purpose, the FESTA software has been developed in C++ language, using an object-oriented framework[2]. It allows an easier code maintenance and expansion with reduced probability of introducing errors (Lages et al. 1999).

The most important task in an object-oriented program is the definition of the class structure. The capability of code reuse and extension depends heavily on the quality of this organization. With respect to the implementation of the finite element code and error estimator algorithms, several classes have been defined here, such as FEM, Node, Element, Material, AnModel, Gauss, Shape, LoadElem, Error, and Sens (Martha et al. 1996). The communication among these classes is performed through instances (objects) of the classes. These objects contain, in their local records, abstract data pointers to objects of other classes. Figure 2 shows the class organization of FESTA.

The *FEM class* represents the numerical discretization of the model into finite elements. It contains references to objects of several classes in the organization: to a list of objects of *Node class* (the nodes of the mesh), to a list of objects of *Element class* (the elements of the mesh), to a list of objects of *Material class* (the groups of distinct materials used), and a reference to an object of the *AnModel class*. The latter class is responsible for the specific features of the type of analysis being performed (e.g. plane-stress, plane-strain, plate-bending, solid, shell). The *Gauss class* holds information about the numerical integration process. An object of this class is associated to an element integration point. The *Shape class* holds the geometric and field interpolation aspects of an element. As explained later in this paper, the *Shape class* is also responsible for the recovery techniques, once the terms of the polynomial expansion are specified. The *LoadElem class* consists of fictitious elements that transfer natural boundary conditions from the elements to the nodes. The *Error class* is responsible for computing the relative error distribution $||e||$, as shown in Eqs. (2) to (14). Finally, the *Sens class* is responsible for computing sensitivities using an extension of the SPR technique.

The development of the FESTA interface is based on a portable user interface toolkit, called IUP/LED (Levy et al. 1996). The portability achieved by using this graphical package allows the computational system to be implemented in various computational environments, e.g. SUN SparcStation, IMB-RS6000, PC/Linux, and Microsoft Windows-95.

In order to illustrate the actual user-interface, Fig. 3 shows different features of the FESTA software available during a self-adaptive simulation. All the windows display different instances of a frame under uniform compressive loading on the top. The upper left window shows the initial finite element mesh (obtained by means of transfinite mapping) and boundary conditions. The upper right and center right windows show the refined finite element meshes, using quads and triangles, respectively, after one step of self-adaptive analysis. The center left window displays both the original and displaced shape of the structure. Both lower windows show contour plots: the left one illustrates the distribution of the ratio of the error (with respect to an average error) over the problem domain, while the right one shows the distribution of sensitivity of the stress field σ_{xx}, i.e. $\partial\sigma_{xx}/\partial x$.

4 Patch recoveries, revisited

Initially, this section describes the SPR based on weighted least square systems. Afterwards, the REP is described and the connections between the two recovery procedures are pointed out. Next, an extension of the SPR technique for sensitivity analysis using the FEM is presented.

4.1 Super-convergent Patch Recovery (SPR)

A generic field (e.g. stress) can be approximated by the polynomial expansion

$$\bar{\sigma} = \mathscr{P}\mathbf{a} \tag{15}$$

where $\mathscr{P}$ contains the appropriate polynomial terms, and $\mathbf{a}$ is a set of unknown parameters. Note that this expansion is used for each component of the stress tensor. For example, for 2D problems and quadratic (8 noded isoparametric) finite elements (see Fig. 4), the following

[2] For further details about object-oriented programming, see the book by Stroustrup (1991).

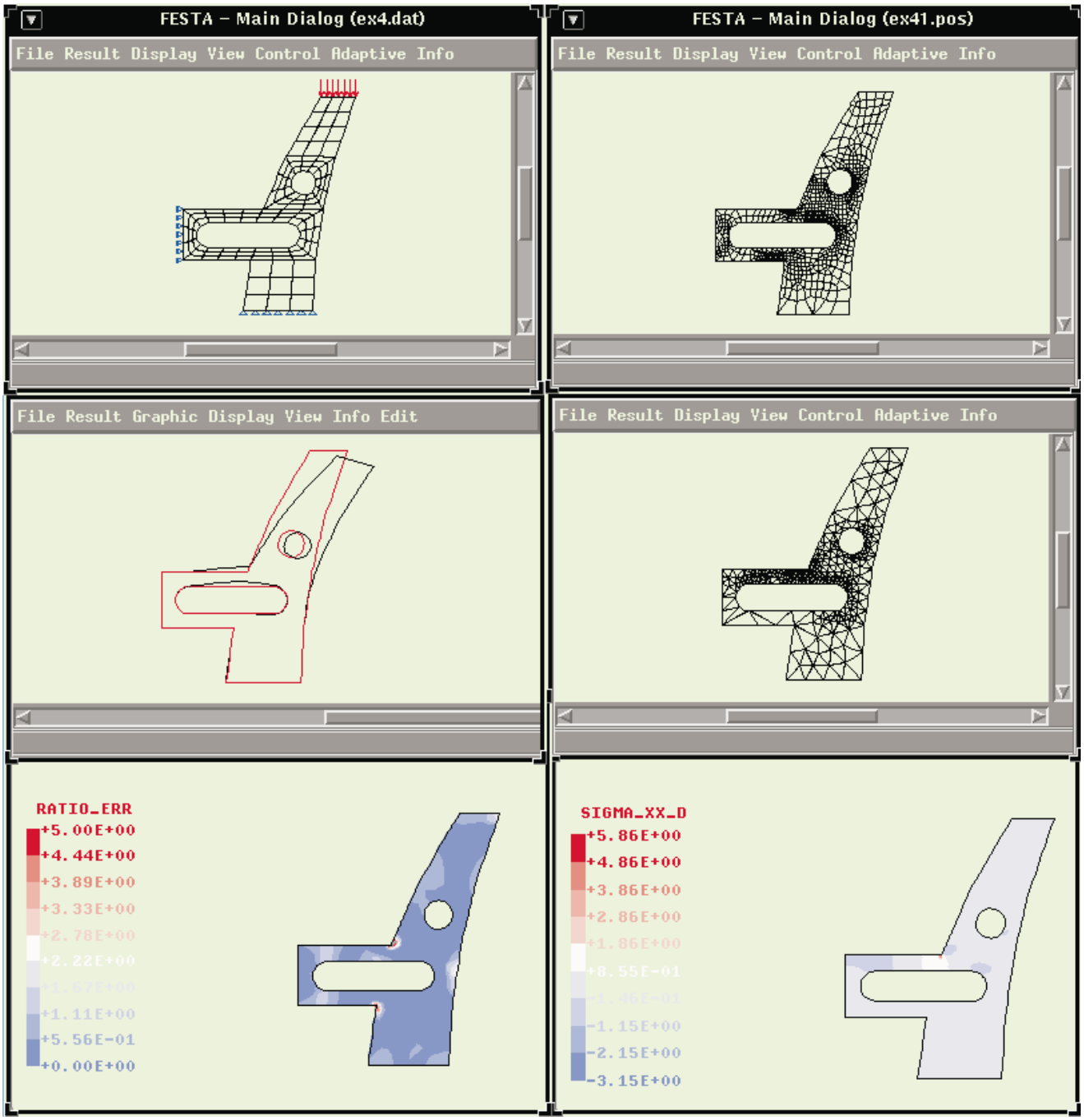

Fig. 3. Overview of the FESTA Interface

approximation is recommended (Zienkiewicz and Zhu 1992a)

$$\mathscr{P} = [1,\ x,\ y,\ x^2,\ xy,\ y^2] \tag{16}$$

$$\mathbf{a} = [a_0,\ a_1,\ a_2,\ a_3,\ a_4,\ a_5]^{\mathrm{T}} \tag{17}$$

The unknown coefficient **a** can be obtained through a weighted least square fit of the polynomial expansion (15) to the values of σ obtained from the finite element solution at the sampling points, i.e. $\hat{\sigma}$. Small patches of elements are used to perform local least square fits, and a weighting parameter (w_i) is considered here to emphasize the influence of the sampling points which are closer to the patch assembly node (see Fig. 4). Thus,

$$w_i = 1/\rho_i^p \tag{18}$$

where ρ_i is the Euclidean distance between the sampling point i and the patch assembly node, and p is an integer. In practical applications p is generally in the range between 0 and 4. The case $p = 0$ corresponds to the original SPR (Zienkiewicz and Zhu 1992a), i.e. with uniform weighting. It is important to mention that the weighting function is effective for solving problems with steep gra-

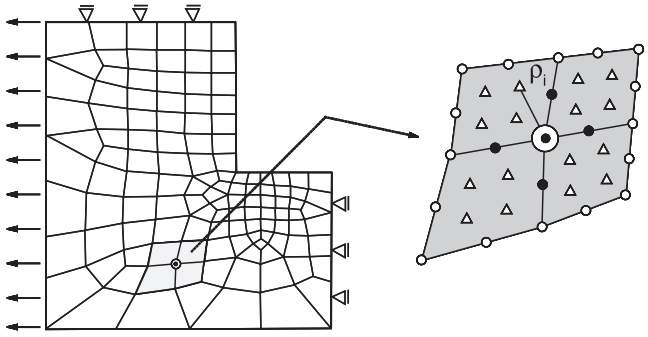

Fig. 4. Patch Recovery Notation: Δ Sampling points; • Nodal values determined by recovery procedure; ○ Nodal Points; ⊙ Patch assembly node; ρ_i distance between the sampling point i and the patch assembly node

dients. It also alleviates ill-conditioning problems associated with the standard SPR.

To illustrate the discrete superconvergent recovery, consider a patch of elements containing m sampling points, as shown in Fig. 4. For a generic sampling point i in this patch, let (x_i,y_i) be the Cartesian coordinates in the global axes. Thus, the weighted least square problem is reduced to the minimization of the following functional

$$\mathscr{G} = \sum_{i=1}^{m} w_i^2 [\hat{\sigma}(x_i, y_i) - \bar{\sigma}(x_i, y_i)]^2 \tag{19}$$

Substituting Eq. (15) in Eq. (19), one obtains

$$\mathscr{G} = \sum_{i=1}^{m} w_i^2 [\hat{\sigma}(x_i, y_i) - \mathscr{P}(x_i, y_i)\mathbf{a}]^2 \tag{20}$$

The minimization problem is solved by setting $\partial\mathscr{G}/\partial\mathbf{a} = \mathbf{0}$, which leads to the following set of linear algebraic equations

$$\mathbf{A}\mathbf{a} = \mathbf{b} \tag{21}$$

where

$$\mathbf{A} = \sum_{i=1}^{m} w_i^2 \mathscr{P}^{\mathrm{T}}(x_i, y_i)\mathscr{P}(x_i, y_i) \tag{22}$$

$$\mathbf{b} = \sum_{i=1}^{m} w_i^2 \mathscr{P}^{\mathrm{T}}(x_i, y_i)\hat{\sigma}(x_i, y_i) \tag{23}$$

Finally, the set of simultaneous equations given by (21) can be solved for the unknown vector $\mathbf{a}$.

4.2
Recovery by Equilibrium in Patches (REP)

In the displacement-based FEM, once the equilibrium equation is solved, the discrete equilibrium of all elements is assured. Thus, every isolated patch Ω_P will also be in equilibrium. Using this argument, Boroomand and Zienkiewicz (1997) have shown that

$$\int_{\Omega_P} \mathbf{B}^{\mathrm{T}}\boldsymbol{\sigma}\, \mathrm{d}\Omega = \int_{\Omega_P} \mathbf{B}^{\mathrm{T}}\hat{\boldsymbol{\sigma}}\, \mathrm{d}\Omega \tag{24}$$

where $\mathbf{B}$ is the strain-displacement matrix and, as before, $\hat{\boldsymbol{\sigma}}$ denotes the finite element solution for stresses. For the elastic case,

$$\hat{\boldsymbol{\sigma}} = \mathbf{D}\mathbf{B}\hat{\mathbf{u}} \tag{25}$$

where $\hat{\mathbf{u}}$ represents nodal displacement values and $\mathbf{D}$ is the elasticity matrix. Substitution of Eq. (25) in the right-hand-side of Eq. (24) leads to

$$\int_{\Omega_P} \mathbf{B}^{\mathrm{T}}\boldsymbol{\sigma}\, \mathrm{d}\Omega = \int_{\Omega_P} \mathbf{B}^{\mathrm{T}}\mathbf{D}\mathbf{B}\hat{\mathbf{u}}\, \mathrm{d}\Omega \tag{26}$$

Now, consider a smooth representation of the stress field over the patch Ω_P, as given by Eq. (15). Using this representation on the left-hand-side of Eq. (26), one obtains

$$\left(\int_{\Omega_P} \mathbf{B}^{\mathrm{T}}\mathscr{P}\, \mathrm{d}\Omega\right)\mathbf{a} = \left(\int_{\Omega_P} \mathbf{B}^{\mathrm{T}}\mathbf{D}\mathbf{B}\, \mathrm{d}\Omega\right)\hat{\mathbf{u}} \tag{27}$$

This set of equations can be rewritten in matrix form as

$$\mathbf{H}\mathbf{a} = \mathbf{F}_P \tag{28}$$

where

$$\mathbf{H} = \int_{\Omega_P} \mathbf{B}^{\mathrm{T}}\mathscr{P}\, \mathrm{d}\Omega \tag{29}$$

and

$$\mathbf{F}_P = \left(\int_{\Omega_P} \mathbf{B}^{\mathrm{T}}\mathbf{D}\mathbf{B}\, \mathrm{d}\Omega\right)\hat{\mathbf{u}} \tag{30}$$

The computation of the polynomial coefficients can be obtained by means of a least square scheme. Let the error $\boldsymbol{\delta}$ be

$$\boldsymbol{\delta} = \mathbf{H}\mathbf{a} - \mathbf{F}_P \tag{31}$$

Thus the function to be minimized is

$$\mathscr{F} = \boldsymbol{\delta}^{\mathrm{T}}\boldsymbol{\delta} = (\mathbf{H}\mathbf{a} - \mathbf{F}_P)^{\mathrm{T}}(\mathbf{H}\mathbf{a} - \mathbf{F}_P) \tag{32}$$

The minimization problem is solved by setting $\partial\mathscr{F}/\partial\mathbf{a} = \mathbf{0}$, which leads to

$$\mathbf{a} = \left[\mathbf{H}^{\mathrm{T}}\mathbf{H}\right]^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{F}_P \tag{33}$$

where $\mathbf{H}$ and $\mathbf{F}_P$ are given by Eqs. (29) and (30), respectively.

An important factor affecting the accuracy of the method is the number of elements and configuration of the patch. For the sake of simplicity, the same procedure adopted for constructing the patches in the SPR is also adopted in the REP process.

4.3
Sensitivity calculations

In this Section, a natural extension of the SPR is presented, which consists of its use for calculation of sensitivity quantities. The procedure is similar to the one proposed by Liszka and Orkisz (1980) for the FDM. The technique is a simple and effective means of directly computing derivatives in finite element analysis. This offers the possibility of using this method in conjunction with modern techniques for sensitivity analysis and optimization in computational mechanics.

Again, consider the patch of Fig. 4, and develop Taylor expansions of a generic (differentiable) function f (e.g. stress, strain, displacement, or related quantities) around each sampling point of the patch. For instance, let (x_o, y_o) denote the coordinates of the patch assembly node P_o. Thus

$$f_i = f_o + h_i \frac{\partial f_o}{\partial x} + k_i \frac{\partial f_o}{\partial y} + \frac{h_i^2}{2} \frac{\partial^2 f_o}{\partial x^2} + \frac{k_i^2}{2} \frac{\partial^2 f_o}{\partial y^2} + \cdots + \mathcal{O}(\rho_i^3) \tag{34}$$

where

$$f_o = f(x_o, y_o), \quad f_i = f(x_i, y_i), \quad h_i = x_i - x_o, \quad k_i = y_i - y_o, \quad \text{and} \quad \rho_i = \sqrt{h_i^2 + k_i^2}. \tag{35}$$

Note that the expansion (34) must be performed for every sampling point i in the patch. Similarly to the previous subsection, the weighted least square problem is reduced to the minimization of the following functional

$$\mathscr{F} = \sum_{i=1}^{m} \left[w_i^2 \left(f_i - f_o - h_i \frac{\partial f_o}{\partial x} - k_i \frac{\partial f_o}{\partial y} - \cdots \right)^2 \right] \tag{36}$$

The minimization problem can be solved by setting $\partial \mathscr{F} / \partial \mathbf{c} = 0$, which leads to a linear system of the form

$$\mathbf{A}\mathbf{c} = \mathbf{d} \tag{37}$$

where $\mathbf{A}$ is given by Eq. (22) with

$$\mathscr{P} = [1, h_i, k_i, \ldots]. \tag{38}$$

The right-hand-side of Eq. (37) is

$$\mathbf{d} = \sum_{i=1}^{m} w_i^2 \mathscr{P}^{\mathrm{T}} f_i \tag{39}$$

and the linear system of equations is solved with respect to the unknowns

$$\mathbf{c} = \left[f_o, \frac{\partial f_o}{\partial x}, \frac{\partial f_o}{\partial y}, \cdots \right]^{\mathrm{T}} \tag{40}$$

The advantage of this formulation is that not only the function, but also its sensitivities can be easily computed. Moreover, the order of sensitivity to be calculated depends on the number of terms used in the Taylor expansion (see Eq. 34).

5
Mesh generation techniques

The required meshing characteristics of an integrated system for interactive, self-adaptive finite element analysis are robustness, versatility, and computational efficiency. For 2D simulations, current meshing technology possesses these characteristics. This section summarizes the developed methodology for 2D self-adaptive mesh generation aimed on these characteristics. The meshing strategy is based on four major components:

- **Geometry-based mesh generation** – The simulation model is defined by its geometry and topology. The simulation attributes are linked to topological entities. A simple topological description is adopted: a list of regions, defined by a list of boundaries, which in turn are defined by a list of curves. The geometrical properties are attached to the curves.
- **Independent boundary and domain recursive decomposition** – Boundary curves are discretized "a priori" using a recursive binary decomposition algorithm, which is based on the error estimation analysis. Each region domain is recursively enumerated using a quadtree decomposition, which is based on the boundary discretization and on the error estimation analysis.
- **Mesh generation combining quadtree and Delaunay techniques** – The mesh is generated in two stages. Initially, the interior of the region is meshed using templates based on quadtree decomposition. In a second stage, the areas between the interior mesh and the boundary of the region are meshed using a "boundary contraction technique" based on a Delaunay triangulation property.
- **Automatic analysis attribute redistribution** – Because attributes are attached to topological and geometrical entities, simulation attributes are automatically reapplied to the generated elements, element boundaries, and nodes of all meshes generated in the self-adaptive process.

5.1
Mesh generation using triangular elements

The present strategy for adaptive meshing is described by means of the example shown in Fig. 5. This figure shows an "L" shape in-plane stress, with fixed vertical displacements at the upper horizontal border, fixed horizontal displacements at the right vertical border, a distributed load along the left vertical border, and free tractions elsewhere on the boundary. An initial finite element mesh for this model is shown in this figure. This mesh is used as the starting mesh for the self-adaptive process: the first error estimation analysis is performed based on this mesh.

5.1.1
Recursive boundary decomposition

One of the most important characteristics of the present self-adaptive meshing strategy is that the boundary refinement is enforced independently of the domain refinement. In fact, the domain discretization requires an "a priori"

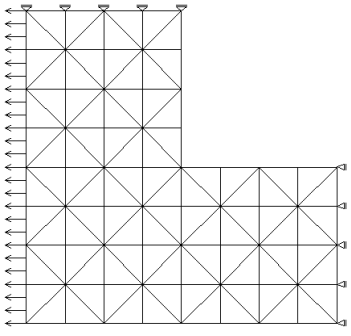

Fig. 5. Initial mesh and BCs

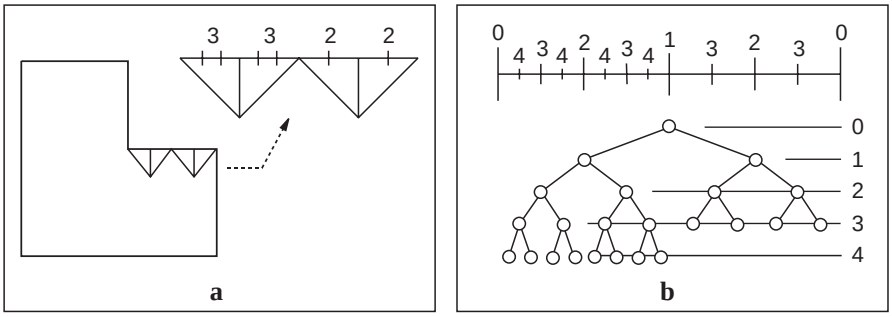

Fig. 6. **a** Current curve discretization; **b** Corresponding binary tree

boundary discretization. This enforces a better mesh gradation along the generated mesh boundary. In this boundary refinement, each boundary curve is discretized in its own parametric space. Therefore, the algorithm is general for all classes of geometric curves.

The algorithm used to refine each boundary curve is a one-dimensional version of the algorithm that is used to refine the domain, which is based on a quadtree technique. Each curve is decomposed using a binary tree technique. The idea consists of recursively subdividing the curve into segments whose sizes are computed based on the characteristic sizes of finite elements adjacent to the curve. These sizes are dictated by the error estimation analysis of the previous step in the adaptive mesh refinement.

The recursive binary refinement algorithm is simple. Each test point is located with respect to the current tree. The cell where the point is located is recursively divided by two until the size of the resulting cell is less than the characteristic size of the element associated with the test point. This process is repeated for each element edge segment along the curve. Each segment mid point is located in the tree resulting from the refinement due to the previous point.

Figure 6(a) shows an example of the binary refinement of a boundary curve. This straight line boundary curve corresponds to one border of the model of Fig. 5. The current curve discretization is shown in Fig. 6(a), where the number of test points (in the middle of predicted refinement segments) are indicated for each element edge. The resulting curve discretization and binary tree are shown in Fig. 6(b). Note that the binary tree is also influenced by the characteristic size (h_i) of the element.

Figure 7 shows the refinement of all boundary curves of the example model. As mentioned previously, in the present meshing strategy, simulation attributes are attached to the curves themselves instead of the element sides or nodes. These attributes, after the boundary refinement, are redistributed to the new element sides and nodes.

5.1.2 Recursive domain decomposition

Several algorithms with different strategies have been published in the literature for automatic meshing of 2D regions. Among them, the algorithms based on recursive spatial enumeration using quadtree (Yerry and Shephard 1984; Baehmann et al. 1987) and the algorithms based on Delaunay triangulation (Joe 1986; Chew 1989; Lo 1989; Florani and Puppo 1992; Potyondy et al. 1995b) are the most robust ones.

The quadtree technique has been successfully used for 2D finite element meshes for several years (Yerry and Shephard 1984; Baehmann et al. 1987). Due to properties of the quaternary tree, the algorithm is fast, efficient, and produces a good transition between regions of different degrees of mesh refinement. One of the problems with this technique is that the boundary refinement is dictated by the domain cell decomposition. This produces an irregular generation of boundary nodes with little positioning control. The algorithm, in its original version, cannot conform with a given boundary refinement, which is a very useful property for the combination of different meshing algorithms and is essential for local mesh modifications.

The present adaptive meshing strategy combines the advantages of the quadtree and boundary contraction

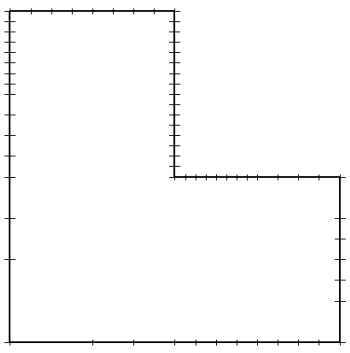

Fig. 7. Boundary refinement

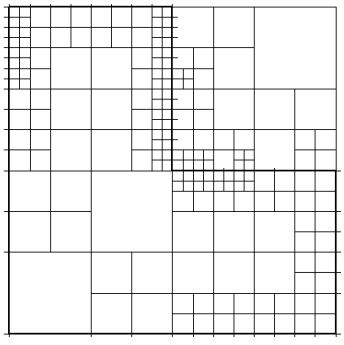

Fig. 8. Initial quadtree

techniques. The present technique can be considered as a hybrid algorithm between the traditional Shephard's quadtree approach (Baehmann et al. 1987) and Potyondy's boundary contraction technique with a quadtree interior point generation (Potyondy et al. 1995b). The domain discretization conforms with the "a priori" boundary refinement (described previously). No additional nodes are created on the boundary curves.

Figures 8 to 14 illustrate the current meshing algorithm corresponding to the adopted "L" shape example. Figure 8 shows the initial quadtree, which is defined solely on the predefined boundary refinement. This tree is further refined based on the error analysis, i.e. the quadtree cell sizes are defined considering the characteristic finite element sizes predicted by the error estimator. This is shown in Fig. 9.

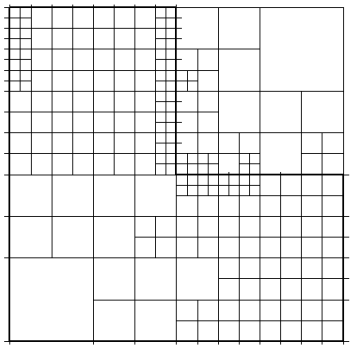

Fig. 9. Quadtree considering error analysis

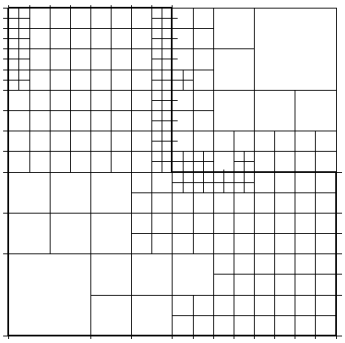

Fig. 10. Quadtree for one depth level

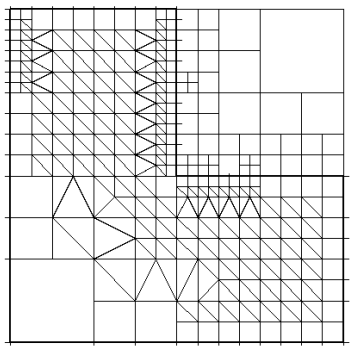

Fig. 11. Quadtree and interior mesh

Following the traditional quadtree meshing approach (Baehmann et al. 1987), finite elements are generated in the interior cells using templates. These templates require the maximum difference in depth level for two adjacent cells to be one. For the model example, Fig. 10 shows the cell decomposition after refining the interior cells to conform with this requirement. Figure 11 shows generated elements in the interior cells using templates for triangles, together with the associated quadtree, and Fig. 12 shows the final interior mesh.

A major difference between the present algorithm and the traditional quadtree meshing is that only interior cells are considered for generating elements based on the quadtree decomposition. The narrow areas between the interior cells and the domain boundary are meshed in a unique process. In the present case, a boundary contraction procedure generates the mesh in the remaining area. No new interior node is generated in this process. A property of the Delaunay triangulation is used for the creation of triangular elements. Given a segment of the current boundary, the selection of a boundary node for the creation of a triangle is based on the maximum included angle. Because the boundary is not necessarily convex, additional checks are needed to avoid triangle overlapping (Shaw and Pitchen 1978). Figure 13 shows the generated triangular elements along the model boundary.

A problem with the boundary contraction technique is the quadratic complexity with respect of the number of

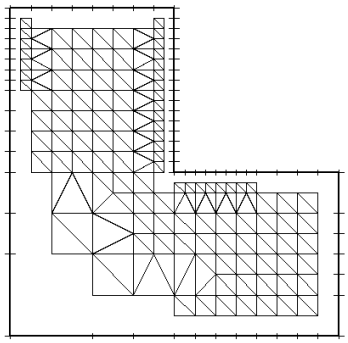

Fig. 12. Interior mesh

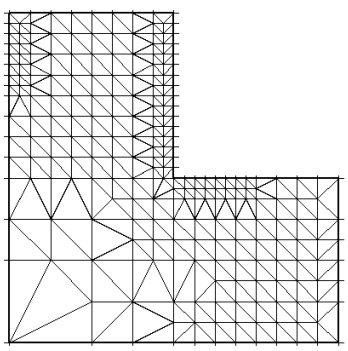

Fig. 13. Resulting mesh

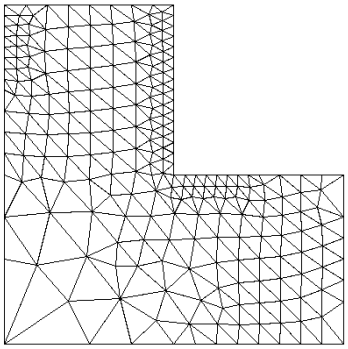

Fig. 14. Final mesh (after smoothing)

mesh nodes. In the present case, the algorithm is enhanced by taking into account the existing quadtree domain decomposition. The selection of nodes for triangle or quadrilateral creation exploits the quadtree data structure to avoid testing of nodes that are not in the vicinity of the boundary segment in consideration. This is certainly an important efficiency gain of the present algorithm when compared to previous boundary contraction procedures.

The final step of the mesh generation is a node coordinate smoothing by averaging the coordinates of adjacent nodes. For the present example, the final mesh is shown in Fig. 14.

5.2
Mesh generation using quadrilateral elements

For the generation of quadrilateral elements, a combination of the procedure devised by Potyondy et al. (1995b) and the templates of the quadtree technique is adopted here. An example, parallel to the one in the previous section, is given in Figs. 15 to 26. The boundary contraction algorithm generates quadrilaterals (preferentially) and triangles using pairs of boundary segments (cf Fig. 17). To this effect, pairs of boundary segments and double interior cells (as mentioned previously) in the quadtree are considered. After the generation of quadrilaterals and triangles by the boundary contraction procedure, templates are used for the creation of quadrilateral finite elements. This algorithm requires an even number of boundary segments. This is achieved in the boundary refinement by subdividing the smallest boundary segment of each curve into two if the final number of segments of the

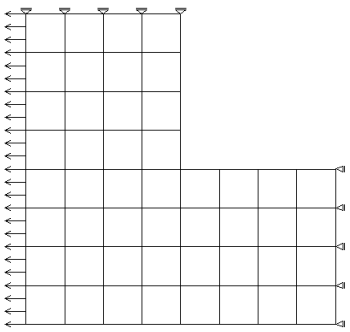

Fig. 15. Initial mesh and BCs

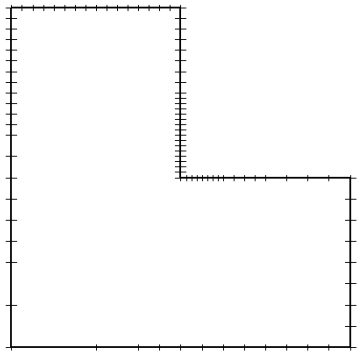

Fig. 16. Boundary refinement

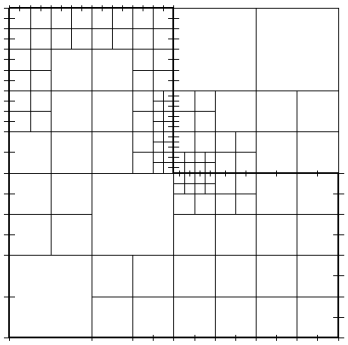

Fig. 17. Initial quadtree

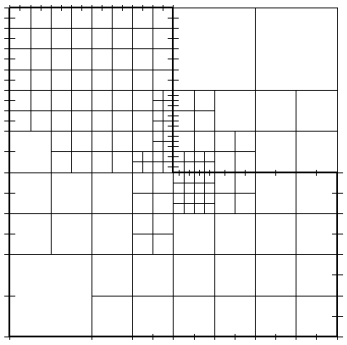

Fig. 18. Quadtree considering error analysis

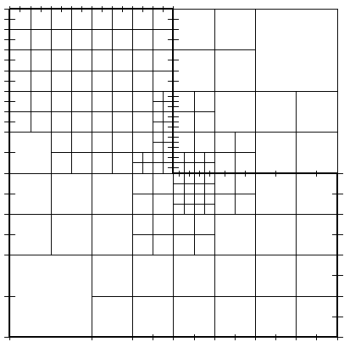

Fig. 19. Quadtree for one depth level

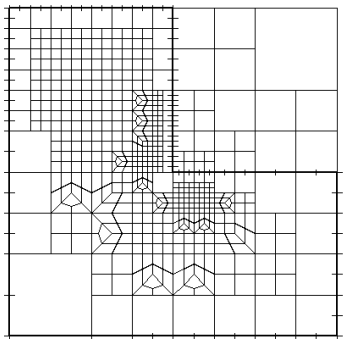

Fig. 20. Quadtree and interior mesh

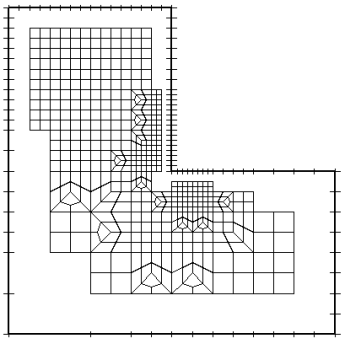

Fig. 21. Interior mesh

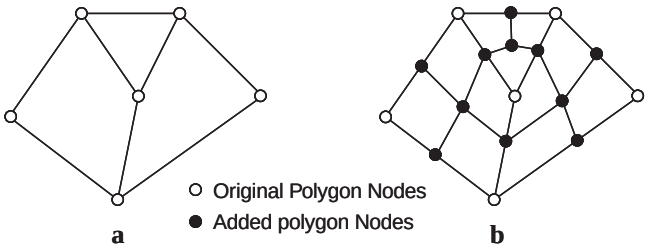

Fig. 22 a, b. Transforming a collection of polygons into a collection of strictly 4-sided polygons. The original collection of 3 and 4 sided polygons is shown in **a**, while the resulting collection of 4-sided polygons is shown in **b**

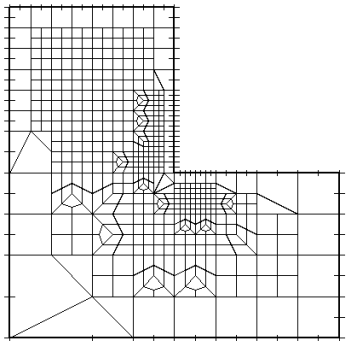

Fig. 23. Mesh with quads and triangles

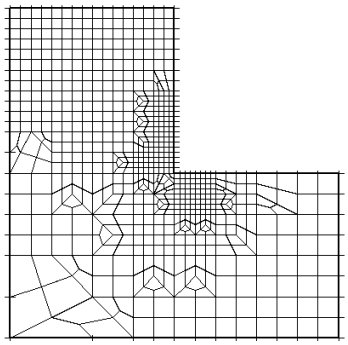

Fig. 24. Resulting mesh

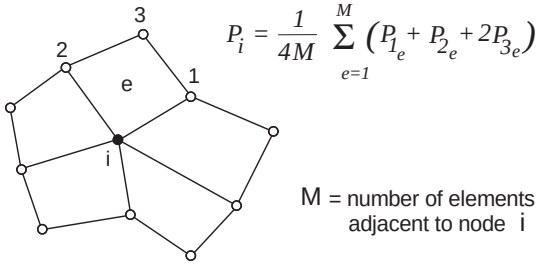

Fig. 25. Quadrilateral element mesh smoothing procedure. The position of each internal node is adjusted to lie at the centroid of the polygon formed by its adjacent elements

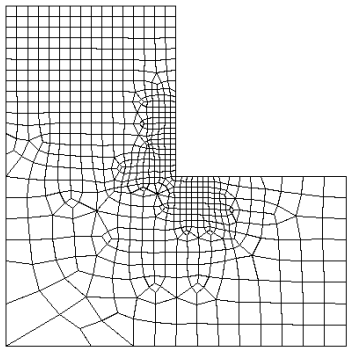

Fig. 26. Final mesh (after smoothing)

curve is odd. The subdivision of the smallest segment guarantees the boundary gradation and provides a more refined boundary where the error is larger.

Consider the initial finite element mesh, made up of linear quadrilateral elements (Q4), as illustrated in Fig. 15 (cf Fig. 5). Figure 16 shows the boundary discretization, as a result of the recursive decomposition technique, from the error analysis obtained using the initial mesh.

The use of pairs of segments leads to some modifications in the treatment of the quadtree, with respect to the recursive domain decomposition. The generation of the initial quadtree is performed considering these pairs of segments, which results in cells with double the size when compared to the resulting sizes if the boundary segments were considered one by one. Figure 17 illustrates this point for the current example.

The refinement of the quadtree due to the error analysis calculated in the previous mesh also takes into ac-

count (for the same reason) double the characteristic size of the element dictated by the error analysis, to be consistent with the generation of the initial quadtree. Figure 18 shows the quadtree after consideration of the error analysis.

Accordingly, the segment pairs are used to classify the cells as "boundary", with respect to the distance to the boundary contour. The vertex cells consider only the vertices of the segment pairs, i.e. they do not consider the vertex between the segments, although this vertex belongs to the boundary. These particular vertices are considered at the final stage of the algorithm. This classification is also used for mesh generation using triangles. However, in this case, each boundary segment is considered independently, while here, for quadrilateral mesh generation, pairs of segments are adopted.

The treatment of the quadtree in order to ensure only one depth level between adjacent cells is the same one used for triangular elements, as explained in the previous Section. Figure 19 illustrates the quadtree with only one depth level.

The templates for quadrilateral elements were devised in such a way that the cells have twice the size of the cells for generating triangular elements. In the present case, this is considered to refine the quadtree cells taking into account the boundary discretization and the error estimation analysis. These templates are then employed in the interior cells, according to the observations described above. The finite elements generated by the templates, together with the associated quadtree, are given in Fig. 20. The interior mesh is shown in Fig. 21.

The boundary contraction algorithm also has to be modified in order to consider pairs of boundary segments. It initially generates a mesh with quadrilateral (preferably) and triangular polygons that will be treated in order to generate the final quadrilateral elements. The criterion for the quadrilateral element generation (Potyondy et al. 1995b) is similar to the one adopted for the triangular meshing, i.e. it adopts a "best angle criterion," according to the Delaunay triangulation. Each polygon of the current mesh is subdivided into a set of quadrilateral elements by inserting a new vertex in its centroid. Figure 22 illustrates this idea.

Note that the new vertices obtained, when each edge of the polygon is subdivided, may already exist (it occurs, for example, in a boundary edge). Thus the algorithm should verify if the vertex to be created already exists. An optimized algorithm, which uses information from the tree and avoids exhaustive searches, is adopted here. Figure 23 shows the finite element mesh with both triangular and quadrilateral elements at the boundary region, and Fig. 24 presents the resulting mesh with only quadrilateral elements.

Finally, a smoothing procedure is applied based on an algorithm proposed by Zhu et al. (1991). Each internal vertex is re-located towards the centroid of the polygon of its vicinity. This procedure tends to generate quadrilaterals of optimum shape and is illustrated in Fig. 25. Fig. 26 shows the final mesh after this smoothing procedure.

6 Numerical implementation aspects

This Section presents some implementation aspects of the FESTA software, illustrating the interconnections among different modules of the whole system. Specific pseudo-codes for the SPR and REP techniques are provided, which show how to transform the governing formulae to matrix form using an OOP philosophy.

The notation adopted for the pseudo-codes is explained next.

- The initial capital letters of each pseudo-code name indicates the OOP class which is responsible for implementing the corresponding method, e.g. **FEM_StressRecoverySPR** means that this method belongs to the FEM class;
- Each pseudo-code has a definition line on the top that defines the input and output explicitly, which are indicated as **in** or **out**, respectively;
- Algorithmic key words such as **if**, **begin**, **end**, **foreach**, among others, are indicated in bold;
- Vectors and matrices are denoted by $\{\cdot\}$ and $[\cdot]$, respectively, while components of vectors or matrices are denoted by means of subscripts (e.g. $\{\cdot\}_{(i)}$, $[\cdot]_{(i,j)}$);
- Comments are indicated in italic (after symbol //).

Only the most important methods, which are necessary for understanding the SPR techniques, are presented here, namely **FEM_StressRecoverySPR**, **NODE_ComputeStressSPR**, **ELEM_ComputeAbMatricesSPR**, and **ELEM_RecoverStressSPR**. These are presented in Pseudo-codes 1 to 4, respectively. It is worth mentioning that, in order to implement the REP technique, relatively small modifications in the SPR pseudo-codes are needed. These modifications are mostly related to the implementation of Eqs. (24) to (33), i.e. computation of the matrices **H** and **F**. Therefore, only the pseudo-code responsible for computing the matrices **A** and **b** (Eqs. (15) to (23)) needs to be changed in order to implement the REP technique. Thus, according to the notation adopted above, this pseudo-code is renamed **ELEM_ComputeHFMatricesREP**, and it is presented in Pseudo-code 5.

Pseudo-code 1: *FEM_StressRecoverySPR*

```
FEM_StressRecoverySPR(out: [rec_stress], out: {counter})
begin
   // Get total number of nodes in the mesh
   num_mesh_node ⟵ NODE_GetNumMeshNodes();
   // Get number of stress components
   num_stress_comp ⟵ ANMODEL_GetNumStressComp();
```

```
    // Initialize recovered stress data structure
    foreach nodal point (i), i = 1, ..., num_mesh_node
    begin
        {counter}_(i) ⟵ 0
        foreach stress component (j), j = 1, ..., num_stress_comp
        begin
            [rec_stress]_(i,j) ⟵ 0.0
        end
    end
    // Loop over the nodes to recover nodal stresses by means of the SPR procedure
    foreach nodal point (i), i = 1, ..., num_mesh_node
    begin
        NODE_ComputeStresSPR (i, [rec_stress], {counter})
    end
    // Update nodal recovered stresses
    foreach nodal point (i), i = 1, ..., num_mesh_node
    begin
        foreach stress component (j), j = 1, ..., num_stress_comp
        begin
            [rec_stress]_(i,j) ⟵ [rec_stress]_(i,j) / {counter}_(i);
        end
    end
end
```

374

Pseudo-code 2: *NODE_ComputeStressSPR*

```
NODE_ComputeStressSPR(in: node, out: [rec_stress], out: {counter})
 begin
    // Check to see if current node can be considered as a PATCH NODE:
    // Get list of adjacent elements
    num_adj_elem ⟵ ELEM_GetAdjElem(node, {adj_elem});
    if(num_adj_elem ≤ 2) return;
    // Check to see if adjacent elements have different materials
    material ⟵ ELEM_GetMaterial({adj_elem}_(1));
    foreach adjacent element (i), i = 2, ..., num_adj_elem
    begin
        current_material ⟵ ELEM_GetMaterial({adj_elem}_(i));
        if (material ≠ current_material ) return;
    end
    // Check to see if current node is a corner node
    if (SHAPE_CornerNode(node ) = false ) return;
    // Get number of nodes per element
    num_node ⟵ SHAPE_GetNumElemNodes({adj_elem}_(1));
    // Get number of polynomial terms
    num_poly_term ⟵ SHAPE_GetNumPolyTerms({adj_elem}_(1));
    // Get number of stress components
    num_stress_comp ⟵ ANMODEL_GetNumStressComp();
    // Get memory for vector of polynomial terms
    {poly_term} ⟵ AllocVector(num_poly_term);
    // Get memory for vector of recovered nodes
    {rec_node} ⟵ AllocVector(num_node * num_adj_elem);
    // Get memory for matrices [A], [a], and [b]
    [A] ⟵ AllocMatrix(num_poly_term, num_poly_term);
    [a] ⟵ AllocMatrix(num_poly_term, num_stress_comp);
    [b] ⟵ AllocMatrix(num_poly_term, num_stress_comp);
    // Loop over the elements of the current PATCH
    foreach adjacent element (i), i = 1, ..., num_adj_elem
    begin
        // Compute element contributions for matrices [A] and [b]
        ELEM_ComputeAbMatricesSPR(node, {adj_elem}_(i), [A], [b]);
    end
```

```
   // Solve linear system of equations with multiple right hand sides: [A] [a] = [b]
   [a] ⟵ [A]^-1 [b]
   // Loop over the elements of the current PATCH for defining the nodes to be recovered
   foreach adjacent element (i), i = 1, ..., num_adj_elem
   begin
      // Select nodes to be recovered
      ELEM_GetRecoveredNodes({adj_elem}_(i), num_rec_node, {rec_node});
   end
   // Recover stress values for each selected node
   foreach recovered node (i), i = 1, ..., num_rec_node
   begin
      // Get Cartesian coordinates of current recovered node
      Cart_coord ⟵ NODE_GetCartCoord({rec_node}_(i));
      // Get polynomial terms for current recovered node
      SHAPE_GetPolyTerms(Cart_coord, {poly_term});
      // Recover stress values for current node
      ELEM_RecoverStressSPR( {rec_node}_(i), num_poly_term, {poly_term}, [a], [rec_stress], {counter});
   end
end
```

Pseudo-code 3: *ELEM_ComputeAbMatricesSPR*

```
ELEM_ComputeAbMatricesSPR(in: node, in: elem, out: [A], out: [b])
begin
   // Get number of nodes per element
   num_node ⟵ SHAPE_GetNumElemNodes(elem);
   // Get number of polynomial terms
   num_poly_term ⟵ SHAPE_GetNumPolyTerms(elem);
   // Get number of stress components
   num_stress_comp ⟵ ANMODEL_GetNumStressComp();
   // Get memory for vector of stresses
   {stress} ⟵ AllocVector(num_stress_comp);
   // Get memory for vector of polynomial terms
   {poly_term} ⟵ AllocVector(num_poly_term);
   // Get memory for vector of shape functions
   {shape} ⟵ AllocVector(num_node);
   // Get memory for vector of nodal coordinates
   {nodal_coord} ⟵ AllocVector(num_node);
   // Get Cartesian coordinates of element nodes
   SHAPE_GetCartCoord(elem, {nodal_coord});
   // Get Cartesian coordinates of the patch node
   patch_coord ⟵ NODE_GetCartCoord(node);
   // Get number of integration points of current element
   num_int_point ⟵ GAUSS_GetNumIntPoints(elem);
   // Loop over the integration points
   foreach integration point (i), i = 1, ..., num_int_point
   begin
      // Get local coordinates of current integration point
      local_int_point_coord ⟵ GAUSS_GetLocalCoord(i );
      // Get shape functions for current integration point
      SHAPE_GetShapeFunction(local_int_point_coord, {shape});
      // Get stress values at integration point (from previous finite element analysis)
      {stress} ⟵ {FiniteElementStressData}
      // Compute Cartesian coordinates of the current integration point
      foreach element node (j), j = 1, ..., num_node
      begin
         Cart_int_point_coord ⟵ Cart_int_point_coord + shape(j)* nodal_coord(j)
      end
      // Compute Euclidean distance between integration point and patch node
      distance ⟵ ComputeDistance(patch_coord, Cart_int_point_coord );
```

```
      // Compute weighting parameter: "p" is given by the user in the range from 0 to 4
      weight ⟵ 1/distance^p;
      // Get polynomial terms for current integration point
      SHAPE_GetPolyTerms(Cart_int_point_coord, {poly_term});
      // Compute contribution for matrices [A] and [b]
      foreach polynomial term (j), j = 1, ..., num_poly_term
      begin
         foreach polynomial term (k), k = 1, ..., num_poly_term
         begin
            [A]_(j,k) ⟵ [A]_(j,k) + weight^2 * poly_term(j) * {poly_term}_(k);
         end
         foreach stress component (k), k = 1, ..., num_stress_comp
         begin
            [b]_(j,k) ⟵ [b]_(j,k) + weight^2 * poly_term(j) * {stress}_(k);
         end
      end
   end
end
```

Pseudo-code 4: *ELEM_RecoverStressSPR*

```
ELEM_RecoverStressSPR(in: node, in: num_poly_term, in: {poly_term}, in: [a], out: [rec_stress], out: {counter})
begin
   // Get number of stress components
   num_stress_comp ⟵ ANMODEL_GetNumStressComp();
   // Update number of times that current node has been recovered
   {counter}_(node) ⟵ {counter}_(node) + 1;
   // Loop over the stresses components
   foreach stress component (i), i = 1, ..., num_stress_comp
   begin
      // Loop over the polynomial terms
      foreach polynomial term (j), j = 1, ..., num_poly_term
      begin
         [rec_stress]_(node,i) ⟵ [rec_stress]_(node,i) + {poly_term}_(j) * [a]_(j,i);
      end
   end
end
```

Pseudo-code 5: *ELEM_ComputeHFMatricesREP*

```
ELEM_ComputeHFMatricesREP(in: node, in: elem, out: [H], out: [F])
begin
   // Get number of nodes per element
   num_node ⟵ SHAPE_GetNumElemNodes(elem);
   // Get number of degrees of freedom per node
   num_dof_node ⟵ ANMODEL_GetNumDofNode(node);
   // Compute number of degrees of freedom of current element
   num_dof_elem ⟵ num_node * num_dof_node;
   // Get number of polynomial terms
   num_poly_term ⟵ SHAPE_GetNumPolyTerms(elem);
   // Get dimension of matrix [B] (strain-displacement matrix)
   dim_B_mat ⟵ ANMODEL_GetDimBMatrix();
   // Get total number of components to be recovered
   num_rec_comp ⟵ num_poly_term * dim_B_mat;
   // Get memory for element stiffness matrix
   [elem_stiff] ⟵ AllocMatrix(num_dof_elem, num_dof_elem);
   // Get memory for element displacement vector
   {elem_disp} ⟵ AllocVector(num_dof_elem);
   // Get memory for vector of shape functions
```

```
{shape} ⟵ AllocVector(num_node);
// Get memory for vector of nodal coordinates
{nodal_coord} ⟵ AllocVector(num_node);
// Get Cartesian coordinates of element nodes
SHAPE_GetCartCoord(elem, {nodal_coord});
// Get memory for matrix [B] (strain-displacement matrix)
[B] ⟵ AllocMatrix(dim_B_mat, num_dof_elem);
// Get memory for matrix [P] (matrix with the polynomial terms)
[P] ⟵ AllocMatrix(dim_B_mat, num_rec_comp);
// Get number of integration points of current element
num_int_point ⟵ GAUSS_GetNumIntPoints(elem);
// Loop over the integration points
foreach integration point (i), i = 1, ..., num_int_point
begin
   // Get local coordinates of current integration point
   local_int_point_coord ⟵ GAUSS_GetLocalCoord(i);
   // Get shape functions for current integration point
   SHAPE_GetShapeFunction(local_int_point_coord, {shape});
   // Compute Cartesian coordinates of the current integration point
   foreach element node (j), j = 1, ..., num_node
   begin
      Cart_int_point_coord ⟵ Cart_int_point_coord + shape(j) * nodal_coord(j)
   end
   // Get [B] matrix at current integration point
   ELEM_GetBMatrix(local_int_point_coord, [B]);
   // Get [P] matrix at current integration point
   ELEM_GetPMatrix(Cart_int_point_coord, [P]);
   // Compute contribution of current element to matrix [H]
   [H] ⟵ [H] + [B]^T * [P];
end
// Compute current element stiffness matrix
[elem_stiff] ⟵ ELEM_ComputeElemStiffMat(elem);
// Get element displacements (from previous finite element analysis)
{elem_disp} ⟵ {FiniteElementDisplacementData}
// Compute contribution of current element to matrix [F]
[F] ⟵ [F] + [elem_stiff] * {elem_disp};
end
```


7 Computational experiments

The effectiveness of the FESTA software is assessed by means of representative numerical examples. The examples illustrate properties associated with the interconnections among finite element analysis, error estimators/adaptive techniques and automatic mesh generation schemes using quadtree and Delaunay triangulation. The following examples are presented:

(1) L-Shaped Plate
(2) Plate with Holes

The first set of examples involves an L-shaped plate. A comparative study involving both adaptive and uniform meshes is conducted. Self-adaptive calculations are carried out using both SPR and REP. In this study, Q8 finite elements are considered. The second set of examples consists of investigating self-adaptive solutions obtained for a plate with holes. Adaptive meshing is considered using the SPR and various element types, e.g. T3, T6, Q4, and Q8. Moreover, sensitivity calculations are also performed for this example.

7.1 L-shaped plate

A square plate of unit thickness with a square hole, under plane stress conditions, is considered here. Due to symmetry, only one quarter of the domain is discretized, as shown in Fig. 5. The uniformly distributed load is $q = 1.0$, the external dimensions are 100.0×100.0, the dimensions of the square hole are 50.0×50.0, and the material has $E = 10^5$ and $\nu = 0.3$. Consistent units are used. The computational experiments are performed with Q8 elements and using an error tolerance $\eta_{\max} = 5\%$. The exact errors are computed in terms of the squared energy norm of the "exact" solution, $||\mathscr{U}||^2_{\text{ex}} = 0.311332$ (see Baehmann and Shephard 1989).

Three adaptive steps are carried out using both the SPR (see Figs. 27 to 30) and REP (see Figs. 31 to 34). Moreover, a sequence of uniform meshes is provided in Figs. 35 to 40. Tables 1, 2 and 3 present some of the results obtained during the analysis, such as: number of the current adaptive step (Step #, for the adaptive meshes) or the identification of the mesh (Mesh #, for the uniform meshes), total number of nodes (#Nodes), total number of

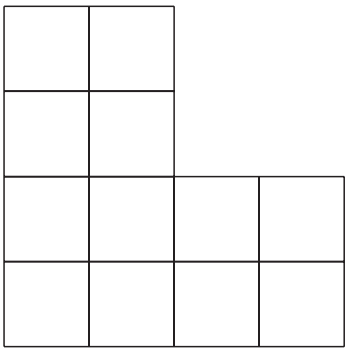

Fig. 27. Initial mesh

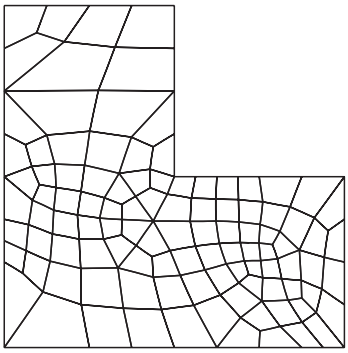

Fig. 28. 1st. Adaptive step – (SPR)

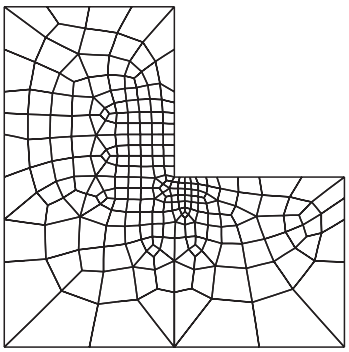

Fig. 29. 2nd. Adaptive step – (SPR)

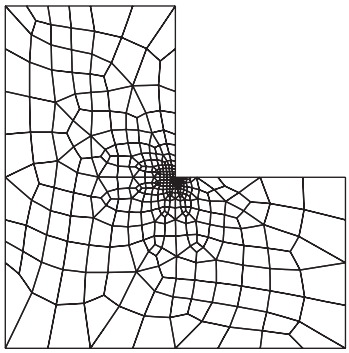

Fig. 30. 3rd. Adaptive step – (SPR)

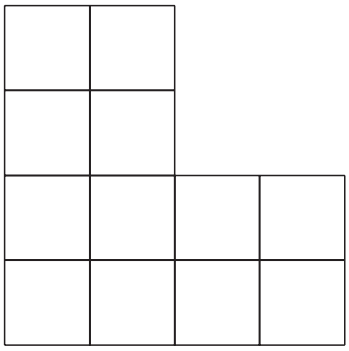

Fig. 31. Initial mesh

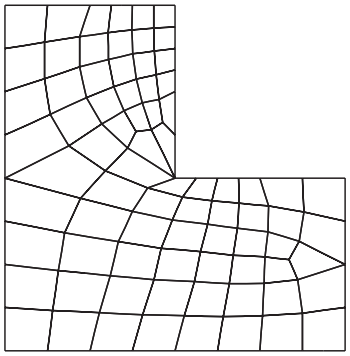

Fig. 32. 1st. Adaptive step – (REP)

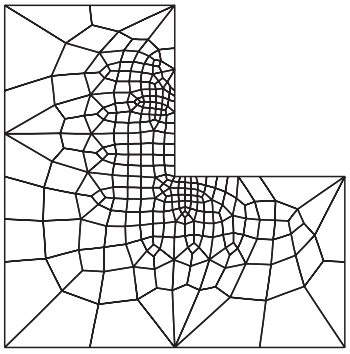

Fig. 33. 2nd. Adaptive step – (REP)

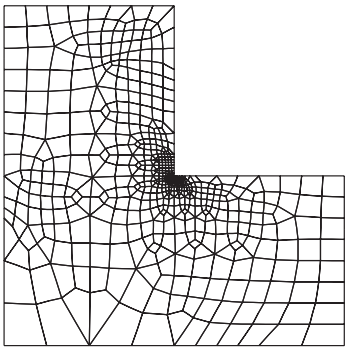

Fig. 34. 3rd. Adaptive step – (REP)

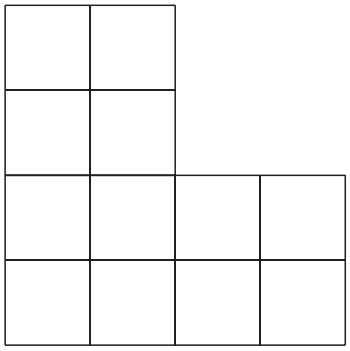

Fig. 35. Uniform Mesh #1

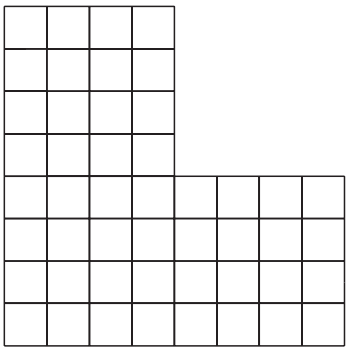

Fig. 36. Uniform Mesh #2

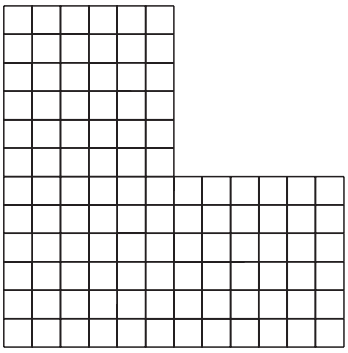

Fig. 37. Uniform Mesh #3

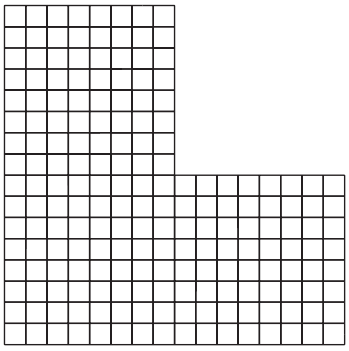

Fig. 38. Uniform Mesh #4

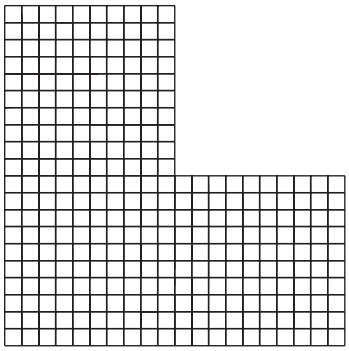

Fig. 39. Uniform Mesh #5

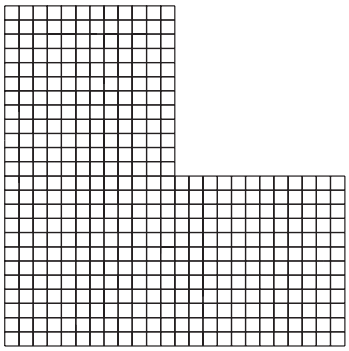

Fig. 40. Uniform Mesh #6

elements (#Elem), total number of degrees of freedom (#DOF), central processing unit (CPU) time, and current relative error. As shown in Tables 1 and 2, the permissible error is achieved in the second iterative step. If the process is continued, the global error drops even more, as expected. For comparable error levels, the REP leads to more

Table 1. Results obtained with the adaptive meshes (using SPR technique)

Step (#)	#Node	#Elem	#DOF	Time (sec)*		Error (%)
				FEA	Meshing	
Initial	53	12	96	0.62	0.58	12.60
1	344	103	670	8.60	1.62	5.28
2	767	242	1516	35.63	3.48	2.83
3	1194	381	2366	92.26	–	1.37

* Using a SUN Sparc Station 20

Table 2. Results obtained with the adaptive meshes (using REP technique)

Step (#)	#Node	#Elem	#DOF	Time (sec)*		Error (%)
				FEA	Meshing	
Initial	53	12	96	0.98	0.72	16.56
1	270	77	518	8.87	3.85	6.67
2	883	282	1756	56.20	12.43	4.29
3	2010	645	3986	235.19	–	1.33

* Using a SUN Sparc Station 20

Table 3. Results obtained with the uniform meshes (using SPR technique)

Mesh (#)	#Node	#Elem	#DOF	Time (sec)* (FEA)	Error (%)
1	53	12	96	0.62	12.60
2	177	48	336	3.10	6.92
3	373	108	720	9.60	5.33
4	641	192	1248	24.93	4.46
5	981	300	1920	61.10	3.89
6	1393	432	2736	120.46	3.49

* Using a SUN Sparc Station 20

DOFs than the SPR for this example. For the uniform meshes, Table 3 shows that the permissible error is achieved after 3 refinement steps, i.e. for Mesh #4. The refinement is continued a couple times leading to a final error of 3.49%. Although the #DOFs is comparable for Step #3 of Table 1 and Mesh #6 of Table 3, the error is smaller in the former case. Thus, selective refinement by means of the SPR leads to a better convergence rate than that of uniform meshes.

In Tables 4, 5 and 6, the effectivity of the error estimates is studied. These Tables provide the Step # for adaptive meshes, or Mesh # for uniform meshes, the energy norm of the finite element solution $||\mathscr{U}_h||^2$, the energy norm of the approximation of the error $||e||^2_{\text{es}}$ and $||e||_{\text{es}}$, the estimated error η_{es} (see Eq. (9)), the energy norm of the exact error $||e||^2_{\text{ex}}$ and $||e||_{ex}$ (where $||e||^2_{\text{ex}} = ||\mathscr{U}||^2_{\text{ex}} - ||\mathscr{U}_h||^2$), the exact error η_{ex} (%), and the effectivity index θ. This index is defined by

$$\theta = \frac{||e||_{\text{es}}}{||e||_{\text{ex}}}. \tag{41}$$

If $\theta > 1.0$, the error is overestimated, and if $\theta < 1.0$, the error is underestimated. The effectivity index for all the meshes using the SPR and REP is very close to 1.0. As expected, the effectivity index for uniform meshes is not as close to 1.0 as the adaptive meshes. This can be seen from the last column of Tables 4 and 6.

Adaptive meshes using the SPR and REP (Tables 4 and 5) converge very fast to the actual value of $||\mathscr{U}_h||^2$ up to 3 decimal digits accuracy. The uniform meshes (Table 6) show a much slower convergence. This can be verified by comparing the 3rd. column of Tables 4, 5 and 6 with the "analytical" solution.

Finally, Fig. 41 shows a graph of $\log(||e||_{\text{es}}) \times \log(\#\text{DOF})$ for the adaptive refinement using SPR and REP, and for the uniform meshes. The adaptive meshes show better

Table 4. Parameters obtained with the adaptive meshes (using SPR technique)

Step	#DOF	$\|\mathscr{U}_h\|^2$	$\|e\|^2_{\text{es}}$	$\|e\|_{\text{es}}$	η_{es} (%)	$\|e\|^2_{\text{ex}}$	$\|e\|_{\text{ex}}$	η_{ex} (%)	θ
Initial	96	0.307074	0.004954	0.070385	12.60	0.004258	0.065253	11.69	1.078642
1	670	0.310038	0.000865	0.029411	5.27	0.001294	0.035972	6.45	0.817603
2	1516	0.311040	0.000249	0.015779	2.83	0.000292	0.017088	3.06	0.923396
3	2366	0.311263	0.000059	0.007681	1.38	0.000069	0.008307	1.49	0.924684

Table 5. Parameters obtained with the adaptive meshes (using REP technique)

Step	#DOF	$\|\mathscr{U}_h\|^2$	$\|e\|^2_{\text{es}}$	$\|e\|_{\text{es}}$	η_{es} (%)	$\|e\|^2_{\text{ex}}$	$\|e\|_{\text{ex}}$	η_{ex} (%)	θ
Initial	96	0.307074	0.008663	0.093075	16.56	0.004258	0.065253	11.69	1.426371
1	518	0.309694	0.001384	0.037202	6.67	0.001638	0.040472	7.25	0.919203
2	1756	0.310920	0.000573	0.023937	4.29	0.000412	0.020297	3.64	1.179337
3	3986	0.311293	0.000055	0.007416	1.33	0.000039	0.006245	1.12	1.187510

Table 6. Parameters obtained with the uniform meshes (using SPR technique)

Mesh (#)	#DOF	$\|\mathscr{U}_h\|^2$	$\|e\|^2_{es}$	$\|e\|_{es}$	η_{es} (%)	$\|e\|^2_{ex}$	$\|e\|_{ex}$	η_{ex} (%)	θ
1	96	0.307074	0.004954	0.070385	12.60	0.004258	0.065253	11.69	1.078642
2	336	0.309580	0.001488	0.038575	6.92	0.001752	0.041857	7.50	0.921592
3	720	0.310241	0.000886	0.029766	5.33	0.001091	0.033030	5.92	0.901173
4	1248	0.310547	0.000619	0.024879	4.46	0.000785	0.028018	5.02	0.887970
5	1920	0.310722	0.000472	0.021726	3.89	0.000610	0.024698	4.43	0.879660
6	2736	0.310836	0.000379	0.019468	3.49	0.000496	0.022271	3.99	0.874139

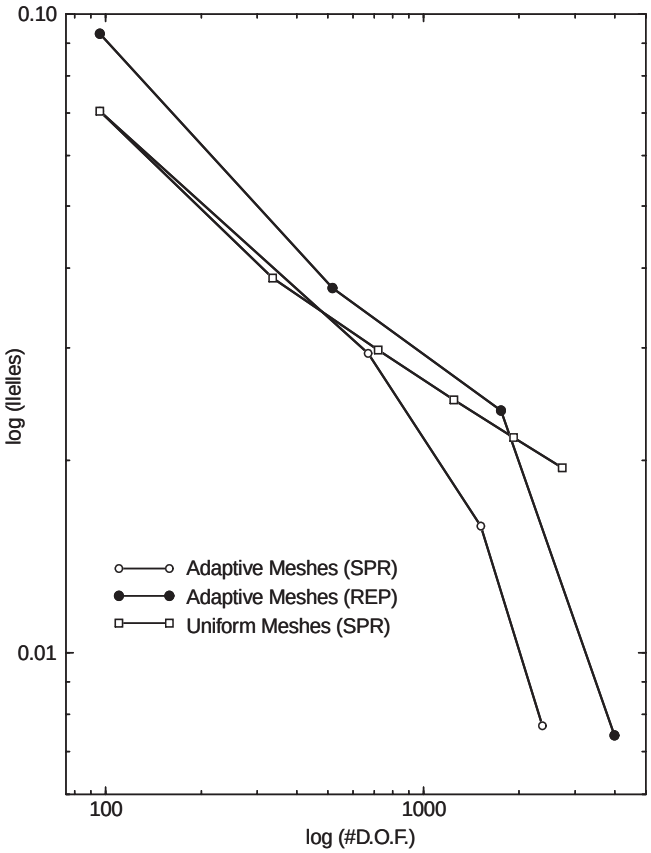

Fig. 41. $\log(\|e\|_{es}) \times \log(\#DOF)$

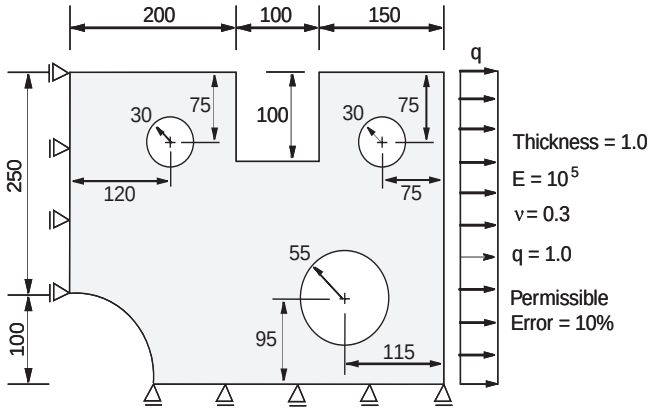

Fig. 42. Plate with holes

convergence than the uniform ones. For this specific example, the behavior of both SPR and REP is in agreement with observations by Boroomand and Zienkiewicz (1997).

7.2
Plate with holes

The numerical model here consists of a three-hole plate under plane-strain, for which a complete self-adaptive analysis is performed. The geometry and boundary conditions, as well as some numerical parameters (in consistent units) used in this example, are shown in Fig. 42. This problem has been studied by Baehmann (1989) and Cavalcante Neto (1994). However, they have used triangular

Table 7. Self-adaptive results for plate under plane-strain

Element	Adaptive Step #	#Node	#Elem	#DOF	Time (sec)*		Error (%)
					FEA	Meshing	
	(Initial Mesh)	81	118	152	2.78	41.52	28.71
T3	1	978	1740	1925	88.51	427.05	13.56
	2	2910	5415	5782	737.49	–	7.45
	(Initial Mesh)	282	118	546	5.15	5.85	15.91
T6	1	935	421	1842	19.55	9.26	7.07
	2	1394	642	2758	34.32	–	4.85
	(Initial Mesh)	81	59	152	1.98	43.10	27.79
Q4	1	1438	1335	2846	105.11	422.47	9.72
	2	5204	4964	10346	1310.53	–	4.66
	(Initial Mesh)	223	59	428	3.60	5.88	16.56
Q8	1	1421	439	2808	38.50	19.68	6.10
	2	2766	872	5494	96.93	–	3.83

* Using a SUN Sparc Station 20

elements, while in the present work, both linear and quadratic, triangular (T3, and T6) and quadrilateral (Q4, and Q8) elements are used. As indicated in Fig. 42, the permissible relative error is $\eta_{\max} = 10\%$ for all elements. Table 7 shows that, for the T6, Q4, and Q8 meshes, the permissible error is achieved in just one iteration of the adaptive procedure. If the process is continued, then the (global) error drops even more – however, as expected, there is a substantial increase in the number of DOFs. The

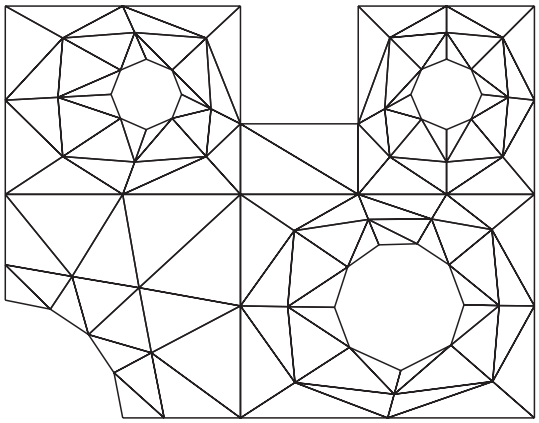

Fig. 43. Initial finite element mesh (T3)

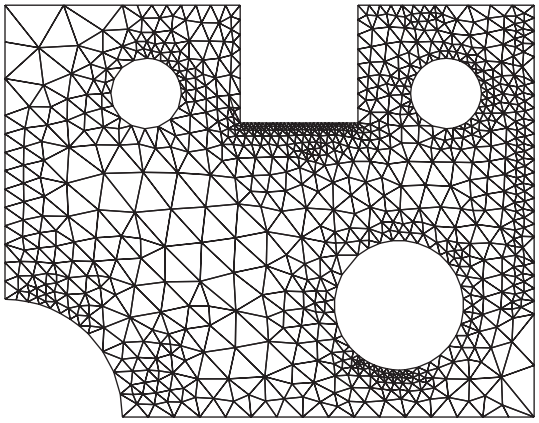

Fig. 44. 1st. Self-adaptive step (T3)

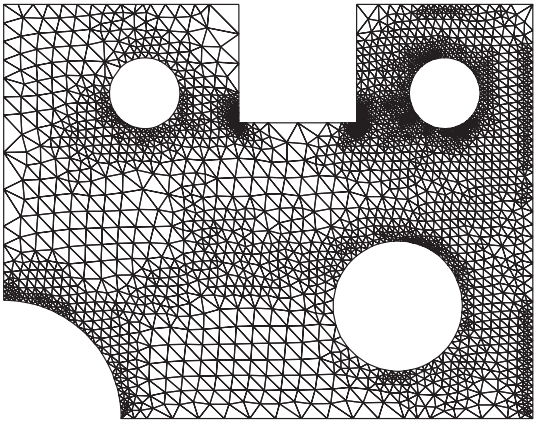

Fig. 45. 2nd. Self-adaptive step (T3)

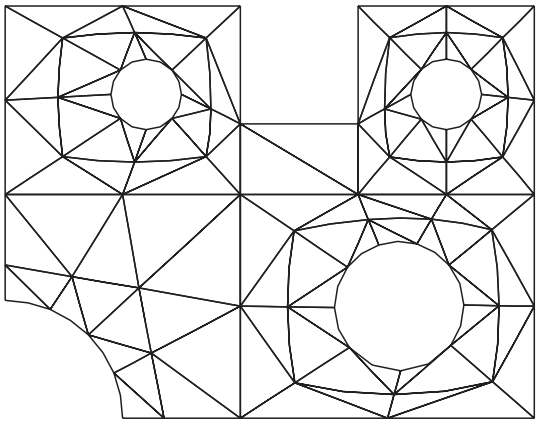

Fig. 46. Initial finite element mesh (T6)

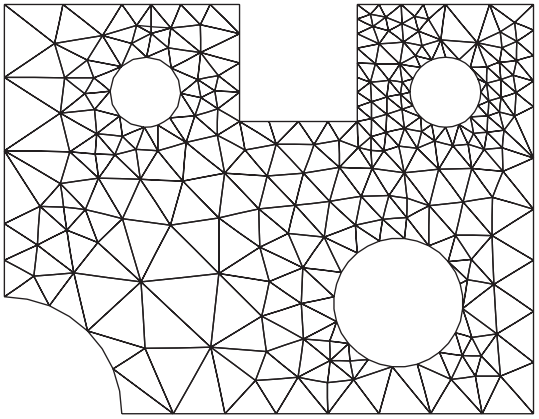

Fig. 47. 1st. Self-adaptive step (T6)

Table presents, for each iterative step, some results obtained during the self-adaptive analysis: number of the current step (Adaptive Step), total number of nodes (#Node), total number of elements (#Elem), total number of degrees-of-freedom (#DOF), FEA and meshing time, and the current relative error. The initial finite element meshes and the two refined meshes corresponding to the self-adaptive steps (for each element type) are shown in Figs. 43 to 45 for the T3 mesh, Figs. 46 to 48 for the T6

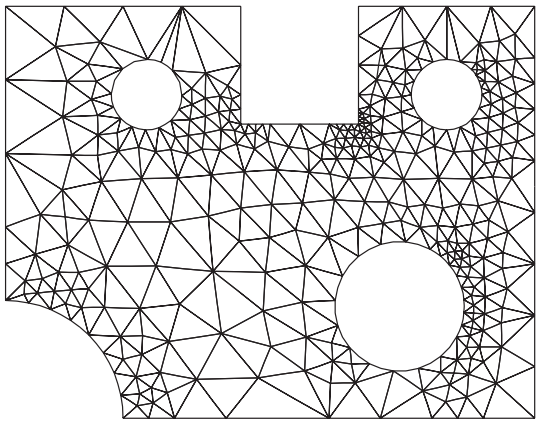

Fig. 48. 2nd. Self-adaptive step (T6)

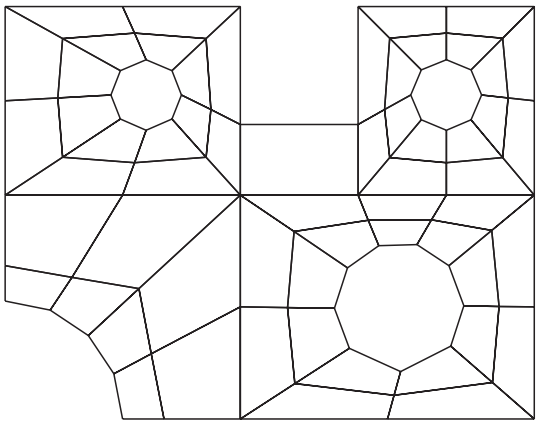

Fig. 49. Initial finite element mesh (Q4)

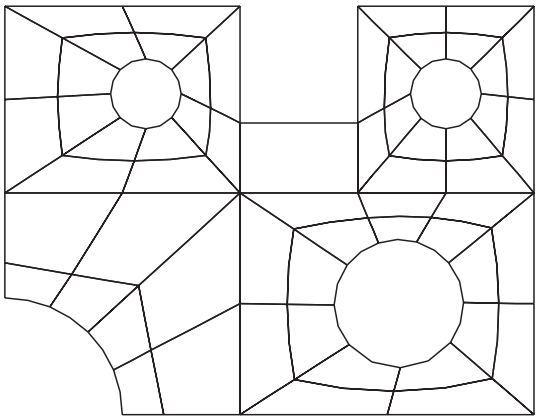

Fig. 52. Initial finite element mesh (Q8)

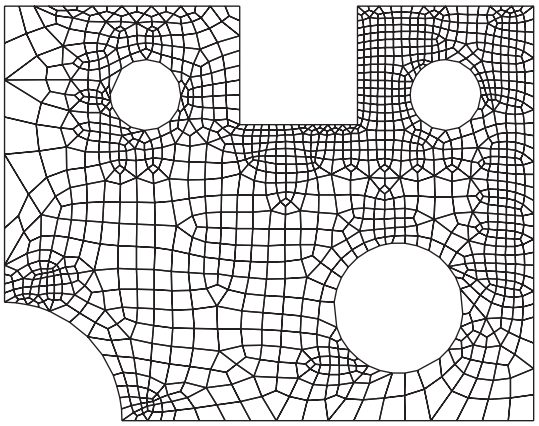

Fig. 50. 1st. Self-adaptive step (Q4)

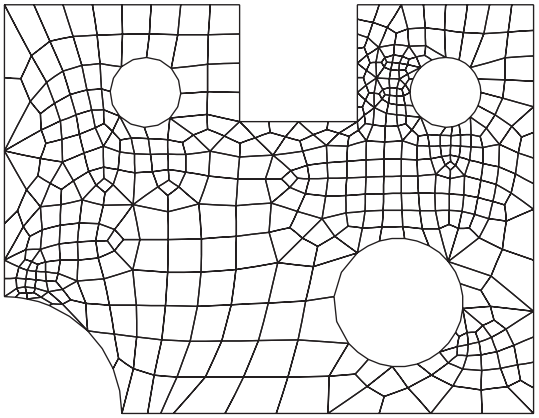

Fig. 53. 1st. Self-adaptive step (Q8)

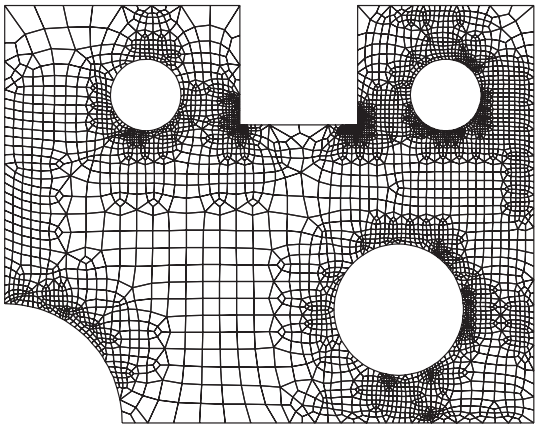

Fig. 51. 2nd. Self-adaptive step (Q4)

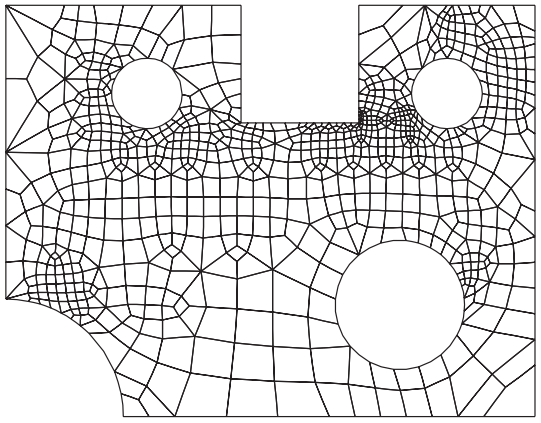

Fig. 54. 2nd. Self-adaptive step (Q8)

mesh, Figs. 49 to 51 for the Q4 mesh, and Figs. 52 to 54 for the Q8 mesh. Note that the adapted mesh with T3 elements, shown in Fig. 45, displays refinement out of the region of stress concentration. This problem might be attributed to the actual error estimator and the type of element employed. In general, all the elements provide a relatively good mesh gradation, however, the aspect ratio obtained with triangles (Figs. 43 to 48) is better than that obtained with quads (Figs. 49 to 54).

7.2.1 Sensitivity calculation

The performance of the FESTA software is evaluated by means of sensitivity calculations. For this purpose, the

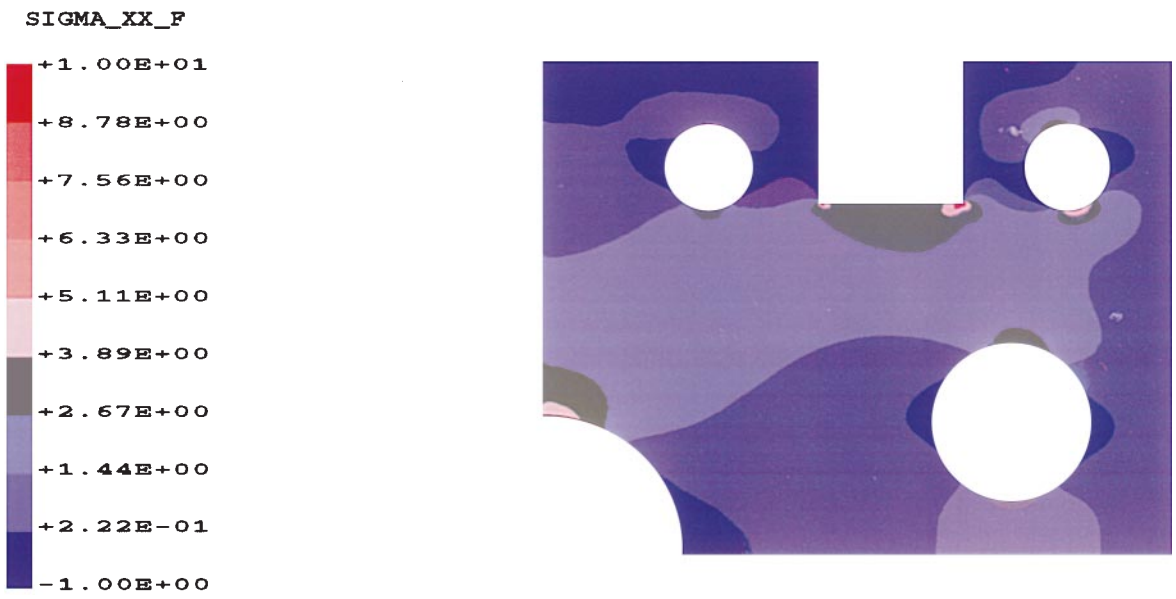

Fig. 55. Stress field σ_{xx} calculated by means of sensitivity analysis

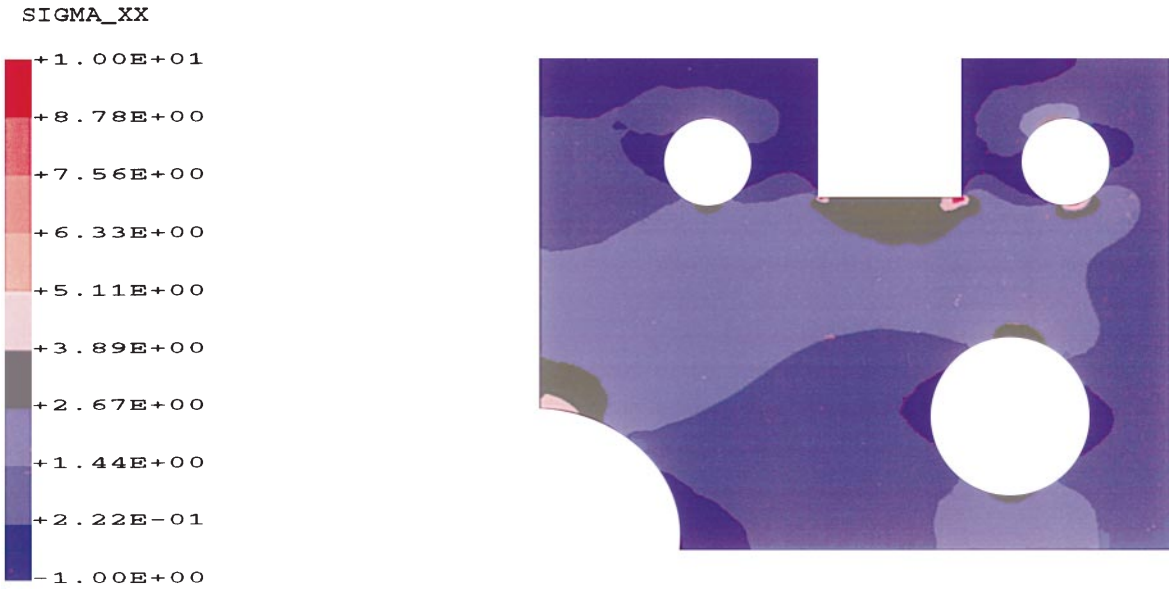

Fig. 56. Nodal smoothing finite element values of σ_{xx} stress field

function corresponding to the σ_{xx} stress field (see the example given in the previous section) is recovered using the sensitivity formulation proposed in Sect. 3.3. The sensitivity results are comparable to the ones obtained with the finite element analysis, as illustrated by Figs. 55 and 56. The sensitivities $\partial\sigma_{xx}/\partial x$ and $\partial\sigma_{xx}/\partial y$ are given in Figs. 57 and 58.

8
Concluding remarks and extensions

A methodology for self-adaptive numerical analysis using the FEM has been presented. Based on five major modules – namely preprocessor, finite element code, error/sensitivity analysis, mesh (re-)generator, and postprocessor – the self-adaptive simulations are performed in a robust, versatile, flexible, and integrated (object oriented) computational environment called FESTA (Finite Element System Technology in Adaptivity). It provides a basic platform for numerical analyses and further development, e.g. implementation of new elements, analysis procedures, error estimators, and sensitivity methods. The effectiveness of the FESTA software is demonstrated by representative numerical examples illustrating features of automatic mesh generators and the interconnections among finite element analysis, recovery procedures, error estimator/adaptivity and mesh generation.

A general implementation of the SPR and the recently proposed REP is presented. Both SPR and REP are compared and used for error estimation and for guiding several adaptive remeshing processes. Moreover, the SPR is also extended for calculating sensitivities of first, second, and higher orders. The mesh regeneration (rather than

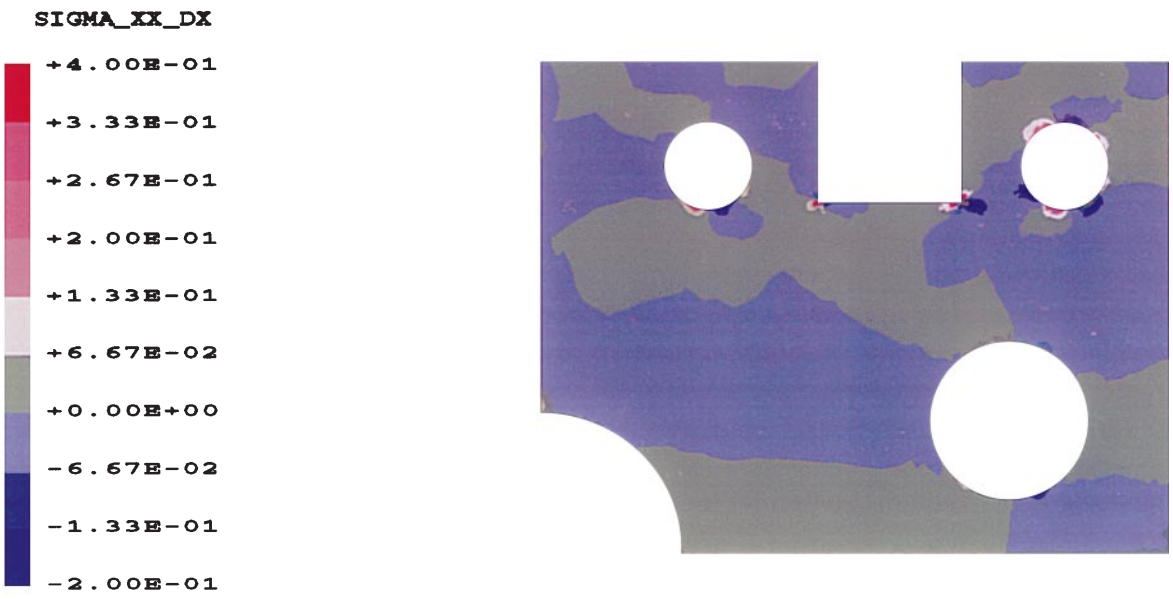

Fig. 57. Sensitivity $\partial\sigma_{xx}/\partial x$

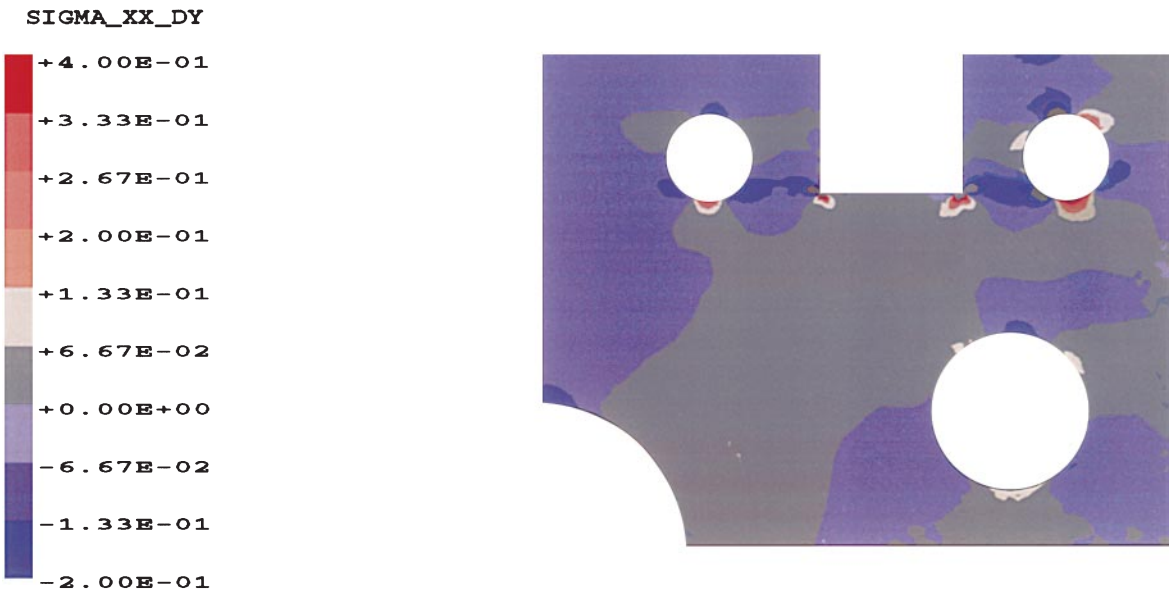

Fig. 58. Sensitivity $\partial\sigma_{xx}/\partial y$

enrichment) procedure is accomplished by combining quadtree and Delaunay triangulation techniques. Surface mesh generation in arbitrary domains is performed automatically during the self-adaptive analysis using either quadrilateral or triangular elements. Currently, this technology is being used for 2D fracture propagation simulation by Potyondy et al. (1995a, b) and Araújo et al. (1997).

A natural subject for future research is the extension of FESTA to three-dimensional (3D) problems. The main ingredient for 3D self-adaptive analysis is an automatic 3D mesh generator. Extension of the 2D mesh generation techniques presented here (e.g. quadtree-based) to 3D (octree-based) is currently being investigated. The 3D mesh generator provides the essential feature for 3D adaptive calculations. It would be worth studying error estimators based on SPR, REP, or the nodal sensitivity-based error estimator recently proposed by Paulino et al. (1997), and their use as drivers for a 3D adaptive mesh refinement scheme. Another area of major interest is adaptive analysis in conjunction with multigrid techniques (e.g. Paulino 1997; Bank 1998).

Other relevant areas for future investigation include fracture mechanics and no linear problems. The solution of general crack problems in a linear elastic fracture mechanics (LEFM) setting involve issues such as development of special crack tip singular elements (e.g. Gray and Paulino 1998), corresponding meshing criteria using a rosette of elements around the crack tip (e.g. Gerstle and

Abdalla Jr. 1990), and error estimation/adaptivity considering singularity of the stress/strain fields (e.g. Lo and Lee 1992; Jayaswal and Grosse 1992; Coorevits et al. 1994). This procedure would be especially relevant for an automated crack propagation procedure, which could benefit from the basic capabilities of FESTA.

The environment proposed herein can also be extended for nonlinear problems. Although adaptive procedures for nonlinear problems are not as well understood as those for linear problems, several techniques have been presented by, among others, Wiberg et al. (1996), Barthold et al. (1998), and Rannacher and Suttmeier (1998) for plasticity problems, Wriggers and Scherf (1998) for contact problems in plasticity, Yang et al. (1989), Zienkiewicz et al. (1990), and Owen et al. (1998) for forming processes, and Ortiz and Quigley (1991), Batra and Ko (1992), Perić et al. (1994), and Zienkiewicz et al. (1995) for strain localization problems. The book edited by Ladevèze and Oden (1998) presents error estimators for nonlinear time dependent problems and corresponding adaptive computational methods. Recently, Mosalam and Paulino (1999) have used mesh enrichment procedures for nonlinear problems involving damage evolution in continuum solid mechanics, where the error is accounted for in an increment-wise manner. Mesh regeneration (rather than mesh enrichment) procedures would be specially advantageous for this type of problems since areas of mesh refinement/derefinement can be effectively treated. Such problems are currently under investigation by the authors.

As discussed above, the range of application of FESTA is potentially broad. Further use of the techniques presented in this paper could contribute towards a reliable and automated environment in computational mechanics (e.g. see Tworzydlo and Oden 1993) employing finite element techniques.

References

Ainsworth M, Oden JT (1997) A posteriori error estimation in finite element analysis. Comp. Meth. Appl. Mech. Eng. 142(1–2):1–88

Araújo TDP, Cavalcante Neto JB, Carvalho MTM, Bittencourt TN, Martha LF (1997) Adaptive simulation of fracture processes based on spatial enumeration techniques. Int. J. Rock Mechanics and Mining Sciences & Geomechanics Abstracts 34(3–4): 551

Babuška I, Chandra J, Flaherty JE (Eds.) (1983) Adaptive Computational Methods for Partial Differential Equations. SIAM (Society for Industrial and Applied Mathematics), Philadelphia

Babuška I, Von Petersdorff T, Andersson B (1994) Numerical treatment of vertex singularities and intensity factors for mixed boundary value problems for the Laplace equation in $\mathbf{R}^3$. SIAM Journal on Numerical Analysis 31(5):1265–1288

Babuška I, Rheinboldt WC (1978) *A posteriori* error estimates for the finite element method. Int. J. Num. Meth. Eng. 12(10):1597–1615

Babuška I, Strouboulis T, Upadhyay CS, Gangaraj SK (1994a) A model study of the quality of *a posteriori* estimators for linear elliptic problems error estimation in the interior of patchwise uniform grids of triangles. Comp. Meth. Appl. Mech. Eng. 114:307–378

Babuška I, Strouboulis T, Upadhyay CS, Gangaraj SK, Copps K (1994b) Validation of *a posteriori* error estimators by a numerical approach. Int. J. Num. Meth. Eng. 37(7):1073–1123

Babuška I, Suri M (1994) The P and H-P versions of the finite element method, basic principles and properties. SIAM Review 36(4):578–632

Babuška I, Zienkiewicz OC, Gago J, Oliveira ER de A (Eds.) (1986) Accuracy Estimates and Adaptive Refinements in Finite Element Computations. John Wiley & Sons, Chichester

Baehmann PL (1989) Automated finite element modeling and simulation. Ph.D. thesis, Rensselaer Polytechnic Institute, Troy, N.Y. – U.S.A.

Baehmann PL, Shephard MS (1989) Adaptive multiple-level h-refinement in automated finite element analysis. Engineering with Computers 5:235–247

Baehmann PL, Wittchen SL, Shephard MS, Grice KR, Yerry MA (1987) Robust geometrically based, automatic two-dimensional mesh generation. Int. J. Num. Meth. Eng. 24(6):1043–1078

Bank RE (1988) PLTMG: A Software Package for Solving Elliptic Partial Differential Equations: Users' Guide 8.0. SIAM (Society for Industrial and Applied Mathematics), PA

Barthold F-J, Schmidt M, Stein E (1998) Error indicators and mesh refinements for finite-element-computations of elastoplastic deformations. Comput. Mech. 22(3):225–238

Batra RC, Ko K-I (1992) An adaptive mesh refinement technique for the analysis of shear bands in plane strain compression of a thermoviscoplastic solid. Comput. Mech. 10:369–379

Beall MW, Dey S, Klaas O, Shephard MS (1997) Adaptive analysis within a geometry-based analysis framework. In: Fourth U.S. National Congress on Computational Mechanics, p. 16, San Francisco, CA

Blacker TD, Belytschko T (1994): Superconvergent patch recovery with equilibrium and conjoint interpolant enhancements. Int. J. Num. Meth. Eng. 37(3):517–536

Blacker TD, Stephenson MB (1991): Paving: A new approach to automated quadrilateral mesh generation. Int. J. Num. Meth. Eng. 32(4):811-847

Boroomand B, Zienkiewicz OC (1997) Recovery by equilibrium in patches (REP). Int. J. Num. Meth. Eng. 40(1):137–164

Borouchaki H, Frey PJ (1998) Adaptive triangular-quadrilateral mesh generation. Int. J. Num. Meth. Eng. 41(5):915–934

Brebbia CA, Aliabadi MH (Eds.) (1993) Adaptive Finite and Boundary Element Methods. Computational Mechanics Publications, Southampton; Elsevier Applied Science, London

Cass RJ, Benzley SE, Meyers RJ, Blacker TD (1996) Generalized 3-D paving: An automated quadrilateral surface mesh generation algorithm. Int. J. Num. Meth. Eng. 39(9):1475–1489

Cavalcante Neto JB (1994) Self-adaptive simulation based on recursive spatial enumeration of bidimensional finite element models. M.S. Thesis, Department of Civil Engineering, PUC-Rio, Rio de Janeiro, RJ – Brazil (in Portuguese)

Cavalcante Neto JB, Martha LF, Menezes IFM, Paulino GH (1998) A methodology for self-adaptive finite element analysis using an object oriented approach. In: IV-WCCM: Fourth World Congress on Computational Mechanics, Buenos Aires, Argentina, 20 pages (available in CD ROM)

Chew LP (1989) Constrained Delaunay triangulation. Algorithmica 4:97–108

Chung H-J, Belytschko T (1998) An error estimate in the EFG method. Comput. Mech. 21(2):91–100

Coorevits P, Ladevèze P, Pelle J-P (1994) Mesh optimization for problems with steep gradients. Engineering Computations 11(2):129–144

Escobar JM, Montenegro R (1996) Several aspects of three-dimensional Delaunay triangulation. Advances in Engineering Software 27(1–2):27–39

Florani L De, Puppo E (1992) An on-line algorithm for constrained Delaunay triangulation. Computer Vision, Graphics, and Image Processing 54:290–300

Gerstle WH, Abdalla Jr JE (1990) Finite element meshing criteria for crack problems. In: Gudas JP, Joyce JA, Hackett EM

(Eds.), Fracture Mechanics: Twenty-First Symposium, ASTM, STP 1074, pp. 509–521, Philadelphia PA
Gray LJ, Paulino GH (1998) Crack tip interpolation, revisited. SIAM J. Appl. Mathematics 58(2):428–455
Jayaswal K, Grosse IR (1993) Finite element error estimation for crack tip singular elements. Finite Elements in Analysis and Design 14(1):17–35
Joe B (1986) Delaunay triangular meshes in convex polygons. SIAM J. Sci. Stat. Comput. 7:514–539
Ladevèze P, Oden JT (Eds.) (1998) Advances in Adaptive Computational Methods. Studies in Applied Mechanics, Vol. 47, Elsevier
Lages EN, Paulino GH, Menezes IFM, Silva RR (1999) Nonlinear finite element analysis using an object-oriented philosophy – Application to beam elements and to the Cosserat continuum. Engineering with Computers, (accepted)
Lee T, Park HC, Lee SW (1997) A superconvergent stress recovery technique with equilibrium constraint. Int. J. Num. Meth. Eng. 40(6):1139–1160
Levy CH, Figueiredo LH de, Gattass M, Lucena CJP, Cowan DD (1996) IUP/LED: a portable user interface development tool. Software: Practice & Experience 26(7):737–762
Liszka T, Orkisz J (1980) The finite difference method at arbitrarily irregular grids and its application in applied mechanics. Computers and Structures 11(1–2):83–95
Lo SH (1989) Delaunay triangulation of non-convex planar domains. Int. J. Num. Meth. Eng. 28(11):2695–2707
Lo SH (1998) 3D mesh refinement in compliance with a specified node spacing function. Comput. Mech. 21(1):11–19
Lo SH, Lee CK (1992) Solving crack problems by an adaptive refinement procedure. Engineering Fracture Mechanics 43(2):147–163
Mackerle J (1993) Mesh generation and refinement for FEM and BEM – A bibliography (1990–1993). Finite Elements in Analysis and Design 15(2):177–188. (Cites 272 references in FEM and 23 in BEM)
Mackerle J (1994) Error analysis, adaptive techniques and finite and boundary elements – A bibliography (1992–1993). Finite Elements in Analysis and Design 17(3):231–246. (Cites 312 references in FEM and 59 in BEM)
Mahomed N, Kekana M (1998) An error estimator for adaptive mesh refinement analysis based on strain energy equalisation. Comput. Mech. 22(4):355–366
Martha LF, Menezes IFM, Lages EN, Parente Jr. E, Pitangueira RLS (1996) An OOP class organization for materially nonlinear finite element analysis. In: Joint Conference of Italian Group of Computational Mechanics and Ibero-Latin American Association of Computational Methods in Engineering, pp. 229–232. Padova, Italy
Mosalam KM, Paulino GH (1999) Towards adaptive nonlinear finite element analysis considering smeared cracking effects. (To be submitted)
Mukherjee YX, Mukherjee S (1997) The boundary node method for potential problems. Int. J. Num. Meth. Eng. 40(5):797–815
Noor AK, Babuška I (1987) Quality assessment and control of finite element solutions. Finite Elements in Analysis and Design 3(1):1–26 (Cites 196 references)
Oden JT, Demkowicz L (1989) Advances in adaptive improvements – A survey of adaptive finite element methods in computational mechanics. In: Noor AK and Oden JT (Eds.), State-Of-The-Art Surveys On Computational Mechanics, chapter 13, pp. 441–467. The American Society of Mechanical Engineers (ASME), New York. (Cites 184 references)
Ortiz M, Quigley IV JJ (1991) Adaptive mesh refinement in strain localization problems. Comp. Meth. Appl. Mech. Eng. 90(1–3):781–804
Owen DRJ, Souza Neto EA, Zhao SY, Perić D, Loughran JG (1998): Finite element simulation of the rolling and extrusion of multi-phase materials – application to the rolling of prepared sugar cane. Comp. Meth. Appl. Mech. Eng. 15(3–4): 479–495
Paulino GH, Gray LJ (1999) Error estimation and adaptivity for the symmetric Galerkin boundary element method. ASCE J. Eng. Mech. (accepted)
Paulino GH, Shi F, Mukherjee S, Ramesh P (1997) Nodal sensitivities as error estimates in computational mechanics. Acta Mechanica 121:191–213
Paulino RH (1997) Application of multigrid techniques for solving fluid dynamic problems. M.Sc. thesis, Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Rio de Janeiro, R.J. – Brazil (in portuguese).
Potyondy DO (1993) A software framework for simulating curvilinear crack growth in pressurized thin shells. Ph.D. thesis, School of Civil and Environmental Engineering, Cornell University, Ithaca, N.Y. – U.S.A.
Potyondy DO, Wawrzynek PA, Ingraffea AR (1995a) Discrete crack growth analysis methodology for through cracks in pressurized fuselage structures. Int. J. Num. Meth. Eng. 38(10):1611–1633
Potyondy DO, Wawrzynek PA, Ingraffea AR (1995b) An algorithm to generate quadrilateral or triangular element surface meshes in arbitrary domains with applications to crack propagation. Int. J. Num. Meth. Eng. 38(16):2677–2701
Perić D, Yu J, Owen DRJ (1994) On error estimates and adaptivity in elastoplastic solids: Applications to the numerical simulation of strain localization in classical and Cosserat continua. Int. J. Num. Meth. Eng. 37(8):1351–1379
Rannacher R, Suttmeier F-T (1997): A feed-back approach to error control in finite element methods: Application to linear elasticity. Comput. Mech. 19(5):434–446
Rannacher R, Suttmeier F-T (1998) A posteriori error control in finite element methods via duality techniques: Application to perfect plasticity. Comput. Mech. 21(2):123–133
Shaw RD, Pitchen RG (1978) Modifications to the Suhara-Fukuda method of network generation. Int. J. Num. Meth. Eng. 12(1):93–99
Shephard MS (1986) Adaptive finite element analysis and CAD. In: Babuška, I. et al. (Eds.), Accuracy Estimates and Adaptive Refinements in Finite Element Computations, chapter 12, pp. 205-225. John Wiley & Sons
Strouboulis T, Haque KA (1992a) Recent experiences with error estimation and adaptivity, Part I: Review of error estimators for scalar elliptic problems. Comp. Meth. Appl. Mech. Eng. 97(3):399–436
Strouboulis T, Haque KA (1992b) Recent experiences with error estimation and adaptivity, Part II: Error estimation for h-adaptive approximations on grids of triangles and quadrilaterals. Comp. Meth. Appl. Mech. Eng. 100(3):359–430
Stroustrup B (1991) The C++ Programming Language. Addison Wesley, Second edition
Szabó B, Babuška I (1991) Finite Element Analysis. John Wiley & Sons, New York
Tabbara M, Blacker TP, Belytschko T (1994) Finite element derivative recovery by moving least square interpolants. Comp. Meth. Appl. Mech. Eng. 117:211–223
Tworzydlo WW, Oden JT (1993) Towards an automated environment in computational mechanics. Comp. Meth. Appl. Mech. Eng. 104(1):87–143
Wiberg N-E, Abdulwahab F (1993) Patch recovery based on superconvergent derivatives and equilibrium. Int. J. Num. Meth. Eng. 36(16):2703–2724
Wiberg N-E, Abdulwahab F, Ziukas S (1994) Enhanced superconvergent patch recovery incorporating equilibrium and boundary conditions. Int. J. Num. Meth. Eng. 37(20):3417–3440
Wiberg N-E, Li XD, Abdulwahab F (1996) Adaptive finite element procedures in elasticity and plasticity. Engineering with Computers 12(2):120–141

Wriggers P, Scherf O (1995) An adaptive finite element algorithm for contact problems in plasticity. Comput. Mech. 17(1–2):88–97

Yang HTY, Heinstein M, Shih J-M (1989) Adaptive 2D finite element simulation of metal forming processes. Int. J. Num. Meth. Eng. 28(6):1409–1428

Yerry MA, Shephard MS (1984) Automatic three-dimensional mesh generation by modified-octree technique. Int. J. Num. Meth. Eng. 20(11):1965–1990

Zhu JZ, Zienkiewicz OC, Hinton E, Wu J (1991) A new approach to the development of automatic quadrilateral mesh generation. Int. J. Num. Meth. Eng. 32(4):849–866

Zienkiewicz OC, Huang GC, Liu YC (1990) Adaptive FEM computation of forming processes – application to porous and non-porous materials. Int. J. Num. Meth. Eng. 30(8):1527–1553

Zienkiewicz OC, Kelly DW, Gago J, Babuška I (1982) Hierarquical finite element approaches, error estimator and adaptive refinement. In: Whiteman, J. (Ed.), Mathematics of Finite Elements and Applications (IV). Academic Press, New York, NY – U.S.A.

Zienkiewicz OC, Pastor M, Huang M (1995) Softening, localization and adaptive remeshing. Capture of discontinuous solutions. Comput. Mech. 17(1–2):98–106

Zienkiewicz OC, Taylor RL (1989) The Finite Element Method, Vol. 1: Basic Formulation and Linear Problems. McGraw-Hill, London

Zienkiewicz OC, Zhu JZ (1987) A simple error estimator and adaptive procedure for practical engineering analysis. Int. J. Num. Meth. Eng. 24(2):337–357

Zienkiewicz OC, Zhu JZ (1992a): The superconvergent patch recovery and a posteriori error estimates. Part 1: The recovery technique. Int. J. Num. Meth. Eng. 33(7):1331–1364

Zienkiewicz OC, Zhu JZ (1992b) The superconvergent patch recovery and a posteriori error estimates. Part 2: Error estimates and adaptivity. Int. J. Num. Meth. Eng. 33(7):1365–1382

Zienkiewicz OC, Zhu JZ (1994) The SPR recovery and boundaries. Int. J. Num. Meth. Eng. 37(18):3195–3196