Joaquim Bento Cavalcante Neto

Simulação Auto-adaptativa Baseada em Enumeração Espacial Recursiva de Modelos Bidimensionais de Elementos Finitos

Dissertação apresentada ao Departamento de Engenharia Civil da PUC-Rio como parte dos requisitos para obtenção do título de Mestre em Ciências em Engenharia Civil: Estruturas.

Orientador: Luiz Fernando C. R. Martha

Departamento de Engenharia Civil
ICAD - Laboratório de CAD Inteligente

Pontifícia Universidade Católica do Rio de Janeiro

Rio de Janeiro, 26 de Setembro de 1994

Aos meus pais, Joaquim Bento e Francisca Maria,
e aos meus irmãos, Patrícia, Alexandre e André.

Agradecimentos

A minha família, pelo apoio, incentivo, confiança e carinho em todos os momentos da minha vida.

A Luiz Fernando Martha, pela amizade formada durante a convivência, fundamental no decorrer deste período todo, e também pela orientação, apoio, incentivo e confiança recebidos durante o desenvolvimento do trabalho.

A todas as pessoas que contribuíram de maneira direta e indireta para a realização deste trabalho. Em especial, a Marcelo Tílio M. de Carvalho pelas longas discussões sobre o algoritmo de geração da malha e sobre o trabalho como um todo; Waldemar Celes Filho, sempre aberto e disposto a colaborar em todos os aspectos do trabalho; Luiz Gil S. Guimarães, pelo esclarecimento de dúvidas no seu trabalho, que serviu de base na parte do módulo de análise numérica e nas funções de pós-processamento; Marcos Aurélio M. Noronha, pela ajuda nos gráficos do capítulo 6; Eduardo Nobre Lages, Ivan F.M. Menezes e João Luiz Campos, pela discussão em alguns aspectos envolvidos; Eduardo Thadeu Corseuil e Paulo Rodacki Gomes, pela ajuda nas figuras dos exemplos com seus atributos no capítulo 6; Luiz Cristóvão G. Coelho, pelas referências sobre algoritmos de geração de malhas na literatura; Camilo da Fonseca Freire, pelas correções em algumas figuras.

A minha turma de mestrado, muito mais que simples companheiros de turma, mas verdadeiros amigos para qualquer hora. Em especial, a Carlos Eduardo Kubrusly da Silva, Marcos Aurélio M. Noronha, Marcos Euclides G. Viana e Paulo Rodacki Gomes.

Ao Peter Hohl, pelas fotografias apresentadas no trabalho, pela amizade e suporte das estações de trabalho.

A todos os amigos, professores e funcionários do Departamento de Engenharia Civil, em especial aos amigos da minha turma de mestrado e a Eduardo Setton, Alexandre Mont'Alverne, Andréia A. Diniz, Evandro Parente, Áurea Holanda, Lucas Barroso, Gerson Melo, Claudia Campos.

A todos os amigos do ICAD/TeCGraf, pelo apoio recebido e pela amizade em todos os momentos, que faz do grupo um ambiente ideal para o desenvolvimento de qualquer trabalho.

A todos os amigos de Fortaleza, em especial aos da minha turma de faculdade: Armando, Aleksander, Alessandra, Luciano, Marcos Aurélio, Marcelo, Paulo André, Pedro, Daniele, Jorge, Marcos Novaes, Baco, Luiz Carlos, Júnior, Cláudio; aos amigos do "oriente": Jackson, Eduardo, Nélio, Marconi, Marcelo, Ricardo, Sérgio, Bosco e as amigas do "ocidente": Morgana, Karine, Carol, Tiana, Soraya, Luciane, Malu, Roberta; e finalmente ao professor Luiz Alberto, pelo incentivo para vir fazer o mestrado na PUC-Rio.

A todos os amigos de "pilantragem", pela descontração nas horas de lazer e apoio em todos os momentos, em especial a: Andréia A. Diniz, Anna M. Hester, Arlindo Cardarett Vianna, Beatriz Castier, Carlos Eduardo Kubrusly da Silva (Turco), Eduardo Nobre Lages (Boi), Eduardo Thadeu Corseuil (Lorde), Ivan F. M. Menezes, João Luiz Campos (Açaí), Luiz Gil S. Guimarães, Marcos Euclides G. Viana (Marreco), Mônica F. da Costa, Paulo Rodacki Gomes (Cabelo) e Raquel Oliveira Prates.

Ao CNPq e ao convênio PUC–PETROBRÁS, pelo auxílio financeiro.

Resumo

Este trabalho descreve um ambiente computacional gráfico interativo para análise integrada e auto-adaptativa de modelos bidimensionais de elementos finitos, com aplicação em problemas estruturais e mecânicos, baseado em uma técnica de *Enumeração Espacial Recursiva*. O sistema é *integrado* porque envolve um modelador geométrico para criar a geometria do modelo, um pré-processador para geração da malha e aplicação de atributos, um módulo de análise para avaliar a solução do modelo, e um pós-processador para visualização dos resultados. O sistema integrado é *auto-adaptativo* porque, além de envolver todos esses conceitos acima descritos, também tem a capacidade de, feita a criação do modelo com uma malha inicial e seus atributos e analisá-lo uma vez, decidir onde refinar a malha, refazer a análise e repetir este procedi-mento até que um critério de convergência pré-estipulado seja atingido. A estratégia proposta envolve muito mais que simplesmente uma técnica de geração auto-adaptativa de malhas. Também é considerada a maneira como o sistema gerencia o processo auto-adaptativo como um todo e como ele redistribui os atributos físicos e geométricos do modelo na nova malha criada. Além disso, a estratégia de auto-adaptação proposta é suficientemente confiável, robusta e rápida para possibilitar uma simulação interativa de problemas práticos de engenharia. Acredita-se que este trabalho é um passo decisivo na direção de uma estratégia de automação real em modelagem por elementos finitos.

Abstract

This work describes an interactive graphics computational environment for auto-adaptive, integrated, two-dimensional finite element analysis, with application to structural mechanics problems. The auto-adaptive strategy is based on recursive spatial enumeration techniques. The system is *integrated* in the sense that it involves a geometric modeler to create the model geometry, a pre-processor for mesh generation and attribute assignment, a numerical analysis module to evaluate the finite element response, and a post-processor for the result visualization. The system is *auto-adaptive* in the sense that, besides considering all the above concepts, it has the capability of, after initial model creation, deciding where to refine the mesh, redoing the analysis, and repeating the procedure until a pre-defined convergence criterium is achieved. The proposed strategy involves much more than simply devising an auto-adaptive mesh generation technique. It is also considered how the system manages the process at all and how it redistributes the physical and geometrical attributes in the new created mesh. In addition, the auto-adaptive proposed strategy is reliable, robust, and fast enough to provide an interative simulation of pratical engineering problems. The author believes that this work is a decisive step in the direction of a real automated strategy in finite element modeling.

Sumário

Lista de Figuras

Lista de Símbolos

$[K]$ – Matriz de rigidez do elemento finito.

$[B]$ – Matriz de compatibilidade de deslocamentos.

$[E]$ – Matriz tensão-deformação do material e do tipo de análise.

$|Ji|$ – Determinante da matriz Jacobiana no ponto de integração i.

ω_i – Peso associado ao ponto de integração i.

$\{R\}$ – Vetor de carregamento nodal equivalente.

$[N]$ – Matriz das funções de forma do elemento finito.

$\{p\}$ – Vetor de carregamento distribuído aplicado.

u – Campo de deslocamentos.

u_p – Campo de deslocamentos prescritos.

$\hat{u}$ – Solução numérica para o campo de deslocamentos.

$\widetilde{u}$ – Solução numérica nodal para o campo de deslocamentos.

Ω – Domínio do problema.

Γ_u – Contorno de deslocamentos prescritos.

Γ_t – Contorno de forças de superfície prescritas.

ε – Campo de deformações.

σ – Campo de tensões.

$\hat{\sigma}$ – Solução numérica para o campo de tensões.

σ^* – Estimativa para o campo de tensões.

D – Operador que define as tensões em função das deformações.

G – Operador linear que relaciona o campo de tensões e as forças de superfície.

e – Erro em função do campo de deslocamentos.

e_σ – Erro em função do campo de tensões.

$\|e\|$ – Erro de discretização medido pela norma de energia.

$\|e_\sigma\|_{L_2}$ – Erro de discretização medido pela norma de energia L_2.

$\|\overline{e}\|$ – Estimativa do erro de discretização medido pela norma de energia.

$\|u\|$ – Raiz quadrada do dobro da energia de deformação.

η – Erro relativo calculado no modelo.

$\overline{\eta}$ – Aproximação para o erro relativo calculado no modelo.

η^* – Erro relativo pré-definido pelo usuário.

ξ – Razão de erro.

h – Tamanho característico do elemento finito.

p – Grau do polinômio da formulação do elemento finito.

θ – Índice de efetividade.

E – Módulo de elasticidade.

ν – Coeficiente de Poisson.

1

Introdução

Na análise de problemas de engenharia, o uso de métodos numéricos tem crescido muito nas últimas décadas. Isto se deve ao fato de na maioria das vezes o analista não conhecer a solução analítica para o problema e precisar então de modelos numéricos para conseguir esses resultados.

O Método dos Elementos Finitos é um dos métodos mais utilizados nos dias de hoje, devido, principalmente, à sua gama variada de aplicações e à boa acurácia dos resultados obtidos. Ele permite, por exemplo, o estudo de deslocamentos e tensões em peças mecânicas, barragens, minas, bem como a determinação de fluxo de calor, pressão neutra e muitas outras análises. Basicamente, estes tipos de análise têm em comum o fato de que se baseiam na solução de um problema para o qual são estabelecidas equações diferenciais parciais relacionando variáveis de campo fundamentais dentro de um determinado domínio, e satisfazendo condições de restrições para essas variáveis fundamentais e suas derivadas na fronteira do domínio. De uma maneira geral, pode-se dizer que a idéia central do método é subdividir o domínio em pequenas regiões (*elementos*) onde o comportamento do campo possa ser aproximado por polinômios ou por funções harmônicas. Essas funções são expressas com base em valores do campo nos vértices (*nós*) destes elementos; estes valores, as incógnitas do problema discreto, são determinados através da minimização de um funcional associado à equação diferencial.

Tendo em vista algumas hipóteses simplificadoras, bem como a dependência de discretização do domínio escolhida, o uso adequado do método requer o conhecimento da formulação dos elementos utilizados e o reconhecimento de fenômenos importantes no modelo físico. Observa-se, então, que a obtenção de resultados satisfatórios da análise numérica é diretamente influenciada pela experiência e pelo conhecimento do analista no uso do método.

Procedimentos adaptativos surgem, portanto, com o objetivo de dar ao analista meios de controlar a qualidade dos resultados da análise numérica e automatizar a busca de uma malha ótima que atenda aos requisitos de uma análise consistente com o modelo físico em questão, dada uma taxa de erro definida. Estes procedimentos vem

de encontro então à necessidade de maior confiabilidade e robustez no uso do método, sem entretanto desprezar a experiência e o conhecimento no seu uso, que sempre são importantes, mas minimizando erros na análise.

1.1 *Método dos Elementos Finitos Adaptativo*

Em meados dos anos 50, quando o método dos elementos finitos apareceu nas análises de engenharia, cabia ao engenheiro analista a tarefa de descrever a malha, através de arquivos texto, diretamente para os programas de análise existentes na época. Um dos primeiros sistemas de uso geral em engenharia foi o ICES (Integrated Civil Engineering System), desenvolvido no MIT (Massachusets Institute of Technology).

A década seguinte trouxe o crescimento de pesquisas sobre o método, com o desenvolvimento de novos elementos e técnicas numéricas mais eficientes. O surgimento do programa SAP (Structural Analysis Program), desenvolvido em Berkeley em 1965 e cujo código fonte foi distribuído pelo mundo, influenciou de modo marcante o desenvolvimento do método.

No começo da década de 70, surgiram os primeiros trabalhos sobre adaptação no método dos elementos finitos. Entretanto, o conceito de adaptação automática da malha baseada em estimadores de erro só foi introduzido consistentemente por Babuska no final da década de 70. Junto com a idéia, a computação gráfica passou a ser utilizada para visualização de malhas, permitindo ao analista minimizar erros existentes nos antigos arquivos textos. Babuska e Rheimboldt (1976, 1978, 1979) desenvolveram os fundamentos matemáticos do método adaptativo baseado no erro da discretização. Estimativas de erro *a posteriori* tornaram possível a concepção de uma estratégia adaptativa para o método dos elementos finitos, de forma que um refinamento poderia ser feito apenas em certos elementos.

Na década de 80, Zienkiewicz et al. (1982) introduziram uma formulação hierárquica no método adaptativo. Nesta década, as idéias adaptativas cresceram paralelamente ao crescimento da computação gráfica. Técnicas de computação gráfica interativa foram usadas para promover um ambiente onde a criação da malha fosse um processo assistido pelo computador. Estes programas, que criavam dados para a análise, passaram a ser chamados de pré-processadores. Técnicas de modelagem geométrica e de geração automática de malhas foram utilizadas por Shephard (1986), com o objetivo de associar o método adaptativo a sistemas gráficos.

Zienkiewicz e Zhu (1987) introduziram uma técnica de estimativa de erro de fácil implementação em qualquer programa de elementos finitos. Esta técnica foi posteriormente aperfeiçoada em 1992. Procedimentos mais sofisticados para estimar o erro numérico de modelos de elementos finitos também estão disponíveis na literatura (Babuska, Szabó, 1991). A principal questão a ser abordada, então, passou a ser a estratégia de redefinição da malha, dada uma estimativa de erro, de forma eficiente e robusta.

Nos dias de hoje (1994), além do aspecto numérico, as pesquisas também evoluem na busca de novas estruturas de dados que possibilitem a criação de programas que façam essa adaptação de uma forma robusta e confiável, com um mínimo de interferência do usuário. Técnicas modernas de modelagem geométrica, como enumeração espacial recursiva através de *quadtree* (2D) ou *octree* (3D), têm um papel importante na geração automática de malhas baseadas em estimativas de erro (Baehmann, 1989).

1.2 *Estratégias de Adaptação*

A análise adaptativa de modelos de elementos finitos envolve estimativas de erro *a priori* ou *a posteriori* que tornam a solução do problema otimizada. As estimativas de erro *a priori* fornecem somente informações qualitativas sobre o modo de convergência da solução quando o número dos graus de liberdade tende a infinito, além de requererem o conhecimento prévio das características da solução exata. Normalmente, essas estimativas não dão informações sobre o erro real no modelo, o que somados às dificuldades em sua formulação, restringe a aplicação a alguns poucos casos.

As estimativas de erro *a posteriori* têm recebido muita atenção dos pesquisadores e sua aplicação tem crescido e se mostrado eficiente. Muitos estimadores têm sido propostos baseados em informações advindas da solução, como normas de energia, etc., e estas estimativas são feitas localmente a nível do elemento ou globalmente ao nível da malha. Os estimadores, usualmente, buscam a malha ótima tentando igualar o valor do erro nos diversos elementos.

Todos os conceitos de sistemas adaptativos empregados procuram reduzir o erro de aproximação no método dos elementos finitos. Existem duas maneiras clássicas de reduzir esse erro de discretização: os chamados *refinamento h* e *refinamento p*, que são duas maneiras de aumentar o número de graus de liberdade, de forma a reduzir o erro em uma análise seguinte (figura 1.1).

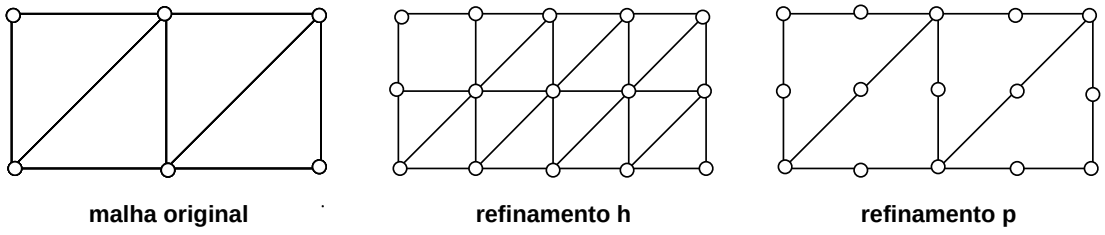


Figura 1.1: Refinamento do tipo "h" e "p" (Cook, 1989).

O *refinamento h* se refere a uma diminuição no tamanho característico h dos elementos, subdividindo-se cada elemento, sem alterar o grau do polinômio usado. O *refinamento p* se refere a um aumento do grau p do polinômio completo de mais alto grau na formulação, acrescentando nós aos elementos ou graus de liberdade adicionais (rotações, por exemplo) aos nós, ou ambos, sem alterar o número de elementos usados. Uma terceira estratégia de refinamento consiste na modificação da posição dos nós, movendo-os na direção das regiões onde haja gradientes acentuados na solução, mantendo-se o mesmo número de graus de liberdade do modelo e a ordem dos polinômios dos elementos. Este é o chamado *refinamento r*. Este procedimento aumenta a densidade da malha em regiões mais críticas e torna o modelo menos refinado em outras áreas. Pode-se ainda combinar as alternativas acima descritas, objetivando conseguir uma convergência mais acelerada, criando procedimentos do tipo *r-h* (Las Casas, 1990; Las Casas, 1991; Cyrino, 1989), *r-p* (Las Casas, 1990; Las Casas, 1991) ou *h-p* (Ribeiro, 1991).

Uma seqüência de refinamentos da malha produz uma convergência para o resultado correto. Este processo é conhecido como *convergência h, p* ou *r* dependendo do procedimento de refinamento adotado. Um programa de análise é chamado *adaptativo* se a adição de graus de liberdade, ou modificação de nós, e a reanálise são feitas com um minímo de intervenção do analista. O programa é dito *auto-adaptativo* se ele pode decidir automaticamente onde graus de liberdade adicionais são necessários dentro da malha, modificar essa malha, refazer a análise, e repetir o procedimento até que um critério de convergência pré-estipulado seja alcançado (Cook, 1989).

Um sistema auto-adaptativo subtende muito mais que simplesmente uma técnica de geração automática da malha, apesar de certamente ser esta a parte mais importante. O sistema deve abordar também a maneira como a redistribuição dos seus atributos e condições de contorno é gerenciada, a um nível de eficiência e rapidez que torne factível o seu uso em análises práticas de engenharia.

1.3 *Proposta da Tese*

Este trabalho propõe um sistema integrado como estratégia de auto-adaptação. Um sistema integrado para análise de elementos finitos envolve um modelador geométrico para criar a geometria do modelo, um pré-processador para geração da malha e aplicação de atributos, um módulo de análise para avaliar a solução do mo-delo e um pós-processador para visualização dos resultados. Um sistema integrado auto-adaptativo, além de todos esses aspectos, compreende todas as características de auto-adaptação mencionadas anteriormente. Neste contexto, os principais objetivos deste trabalho são:

- O desenvolvimento de uma estratégia de auto-adaptação que seja confiável, robusta e rápida suficientemente para possibilitar uma simulação interativa;
- A consideração adequada dos atributos físicos e geométricos do modelo dentro do processo auto-adaptativo; em particular, uma estratégia que proporcione um refinamento regular das curvas de fronteira e de interface entre materiais distintos;
- O desenvolvimento de um sistema gráfico integrado auto-adaptativo, com uma interface de múltiplas janelas, para análise e visualização de resultados de modelos bidimensionais, com aplicação em problemas estruturais e mecânicos;
- A criação de um sistema de fácil extensão, onde técnicas de programação orientada a objetos são exploradas.

Deve ficar claro que não é objetivo deste trabalho a pesquisa de novos estimadores numéricos de erro. O enfoque principal é a estratégia de redefinição da malha e redistribuição dos atributos para se adequar a uma análise de erro genérica. Neste contexto, o processo proposto pode ser classificado como sendo uma estratégia que usa um estimador de erro *a posteriori* e um *refinamento h*.

O sistema desenvolvido recebe as informações da malha, com seus atributos físicos e mecânicos, de um modelo de elementos finitos gerada por um pré-processador gráfico existente (Mtool, 1992) e é capaz de decidir automaticamente onde refinar a malha, tratando consistentemente as condições de contorno e outros atributos, e refazer a análise até que um critério de erro seja atendido. Neste trabalho, o critério adotado é um erro relativo na norma de energia global, fornecido pelo usuário. Vale ressaltar que, a cada etapa de convergência para atingir o critério de parada, o usuário pode intervir no processo, modificando o erro relativo desejado; além disso, é possível visualizar todos os resultados da análise através de funções de pós-processamento gráfico existentes.

O estimador de erro usado é o proposto por Zienckiewicz e Zhu (1987), utilizando uma norma de energia para o erro, e a recuperação de tensões nodais pelo

projetor de Hinton e Campbell (1974), com suavização de valores com base na média dos valores nodais dos elementos adjacentes. Este estimador de erro foi implementado no módulo de análise numérica segundo uma disciplina de programação orientada a objetos, o que permite uma flexibilidade de análise utilizando todos os tipos de elementos triangulares ou quadrilaterais disponíveis. Vale ressaltar que não se questiona neste trabalho a acurácia do estimador escolhido; ele foi usado simplesmente por ser de fácil implementação e bastante difundido na área. Além disso, a disciplina de programação orientada a objetos facilita a troca do estimador, pois basta trocar um método no programa, reutilizando procedimentos existentes e afetando o mínimo possível toda a estrutura do sistema integrado proposto.

Com relação ao refinamento, este é orientado pelo tamanho característico de cada elemento, projetado com base em uma razão de erro fornecida e no grau p do polinômio da formulação do elemento. Este refinamento é tratado segundo uma técnica de *Enumeração Espacial Recursiva* em dois níveis: refinamento do contorno do modelo, e refinamento do seu domínio. Para o contorno, o refinamento usa uma estrutura de dados de árvore binária (*binary tree*); para o domínio, uma estrutura de dados de árvore quaternária (*quadtree*). Em ambas as árvores, o tamanho das células é definido em função do tamanho característico dos elementos na região correspondente.

O processo de geração de malhas propriamente dito procura aliar as vantagens das técnicas de *quadtree* e da triangulação de Delaunay por contração do contorno, partindo de uma discretização de bordo fornecida. A idéia do algoritmo consiste na combinação das técnicas de *quadtree*, para geração dos elementos no interior da região, e de triangulação de Delaunay, para geração dos elementos entre o contorno da região e a malha gerada por *quadtree*. Um aspecto fundamental deste algoritmo, que resulta em uma melhor eficiência computacional, consiste na utilização das informações de adjacência da estrutura *quadtree* também para o processo de triangulação de Delaunay.

Outros autores adotaram um estratégia semelhante para o processo auto-adaptativo baseado na técnica de *quadtree*. O trabalho do grupo de Shephard (Baehmann, 1989) é um destes. Entretanto, estes trabalhos apresentam os mesmos problemas da geração de malhas sem auto-adaptatividade por *quadtree*. Estes problemas são oriundos da compatibilização da malha gerada de acordo com a árvore quaternária e o contorno do modelo. Esta compatibilização é feita por algoritmos geométricos complexos, dependentes de tolerância, e também resulta em uma discretização não regular do contorno do modelo. O método proposto neste trabalho está fundamentado na

solução destes problemas.

Como o sistema tem a capacidade de refinar automaticamente a malha, ele também precisa refazer consistentemente as condições de contorno para o modelo refinado para refazer a análise. Este aspecto, também abordado neste trabalho, não é em geral tratado em outros trabalhos de auto-adaptatividade na literatura, mas é de fundamental importância para uma estratégia de auto-adaptação.

1.4 *Organização da Tese*

A apresentação da estratégia geral de auto-adaptação proposta é feita no capítulo 2. Salienta-se a importância de se manter a consistência geométrica do modelo e a consistência dos atributos mecânicos das curvas e regiões.

O capítulo 3 explica a organização de classes para análise por elementos finitos segundo uma disciplina de programação orientada a objetos, dando enfoque aos métodos implementados para avaliação de erro.

O processo de auto-refinamento das fronteiras do modelo é descrito no capítulo 4. São explicadas a estrutura de dados de árvore binária (*binary tree*) que refaz a discretização das fronteiras, e a classe *Curva* criada, segundo a mesma metodologia de programação orientada a objetos, para controle da geometria e da reaplicação consistente dos atributos na malha refinada.

O capítulo 5 explica o método de geração da malha usado na auto-adaptação utilizando a estrutura de dados do tipo árvore quaternária (*quadtree*). Com exceção da parte dependente da estimativa de erro em uma etapa da geração da árvore, este método pode ser usado como um método de geração de malhas em domínios arbitrários, em processos de análise não necessariamente adaptativos.

No capítulo 6, são mostrados alguns exemplos da aplicação da estratégia de auto-adaptação. Este capítulo também trata a visualização gráfica com múltiplas janelas. Finalmente, no capítulo 7, são apresentadas as conclusões e sugestões para trabalhos futuros.

2

Estratégia de Auto-Adaptação

Este capítulo descreve a estratégia de auto-adaptação proposta. A estratégia define dois módulos: o módulo de análise numérica e o módulo auto-adaptativo. Uma malha inicial definida é analisada pelo módulo de análise numérica e é exportada para o módulo auto-adaptativo para a adaptação da malha e uma posterior análise. O ciclo se repete até que se satisfaça um critério de erro pré-definido. Dois aspectos são importantes. O primeiro é que, a cada etapa do processo, pode-se visualizar os resultados obtidos (tensões, deslocamentos, etc.) da nova malha analisada e intervir no processo, modificando o paramêtro de erro pré-definido. O segundo aspecto é que, como se trata de um sistema auto-adaptativo, é preciso, a cada nova etapa, refazer os atributos consistentemente com a malha gerada. Este aspecto não é detalhado em outros trabalhos de auto-adaptação, que só enfocam o aspecto da modificação da malha e não mencionam como os atributos são tratados.

2.1 *Enumeração Espacial Recursiva e Triangulação*

Diversos algoritmos com diferentes estratégias para geração automática de malhas de elementos finitos em regiões bidimensionais têm sido publicados na literatura. Dentre estes, destacam-se os algoritmos baseados em técnicas de enumeração espacial recursiva utilizando *quadtree* (Yerry, Sheppard, 1984; Baehmann et al., 1987) e os baseados nos critérios para triangulação de Delaunay (Joe, 1986; Chew, 1989; Lo, 1989; Vianna, 1992; Potyondy, 1993).

A *quadtree* é uma estrutura de dados que pode ser usada para implementar algoritmos de enumeração espacial recursiva (Samet, 1984). Estes algoritmos subdividem uma dada região do espaço (bidimensional ou tridimensional) em regiões de forma similar. O processo é então repetido um número arbitrário de vezes para as regiões menores. Estes algoritmos trabalham somente com um número pequeno de formas de regiões, como triângulos, quadrados ou hexágonos para regiões bidimensionais, e tetraedros, cubos ou hexaedros para regiões tridimensionais. Em regiões bidimensionais, a

forma mais usada é o quadrado. Neste tipo de *quadtree*, cada região ou não é subdividida, ou é subdividida em quatro subregiões similares, dependendo do caso. Essa informação pode ser convenientemente armazenada em uma estrutura de dados do tipo árvore, onde cada nó ou tem quatro filhos ou nenhum filho (*quadtree*), no caso bidimensional, ou tem oito filhos ou nenhum filho (*octree*), no caso tridimensional. A figura 2.1 mostra um exemplo simples de uma região subdividida e sua *quadtree*. As regiões não divididas, que correspondem a nós folhas na árvore, são chamadas *células*. O tamanho de uma célula pode ser facilmente determinado pela sua profundidade na árvore.

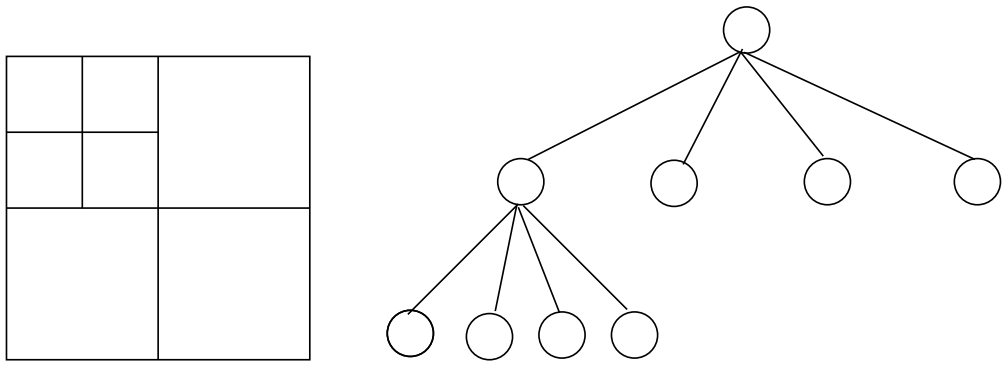


Figura 2.1: Uma região decomposta e sua árvore quaternária (*quadtree*).

A utilização da técnica de *quadtree* para geração de malhas bidimensionais, devido às propriedades de uma árvore quaternária, fornece algoritmos muito mais rápidos do que os demais, além de propiciar uma boa transição entre regiões com diferentes graus de refinamento da malha. A principal desvantagem desta técnica, na sua forma original, é que a discretização do contorno da região é comandada também pela árvore quaternária. Isto faz com que o contorno fique discretizado de uma maneira irregular, com a criação de vértices em posições não controladas. Por causa disto, esta técnica não é automaticamente compatível com uma discretização fixa, previamente fornecida, para o contorno da região. A conformação com uma discretização fornecida para o contorno é bastante útil para se combinar diferentes algoritmos de geração de malha, e é essencial em problemas que se deseja modificar uma malha localmente.

Nos algoritmos baseados na triangulação de Delaunay, a discretização do contorno não só é garantida como faz parte dos dados de entrada para o próprio algoritmo. Exemplos de algoritmos que utilizam esta técnica para a geração de malhas de elementos finitos em regiões com contornos arbitrários (i.e., não convexos e com furos) podem

ser encontrados nos trabalhos de Shaw (1978) e Lo (1989). Embora existam algumas variações entre esses algoritmos, em geral utiliza-se um procedimento de contração do contorno, juntamente com critérios de Delaunay. O processo de geração da malha é desenvolvido em duas etapas. Primeiro são gerados os pontos no interior da região; diferentes técnicas são aplicadas para esta etapa. Em seguida, são gerados os triângulos da malha, utilizando-se a estratégia da contração do contorno. Estes algoritmos, entretanto, não são muito eficientes, apresentando uma complexidade quadrática no pior caso.

Deve-se observar que existem algoritmos para triangulação por Delaunay de regiões com restrições (i.e., regiões não convexas e com furos) com desempenho $\Theta(nlogn)$ (Chew, 1989; De Florani, Puppo, 1992), sendo n o número de nós definidos pela discretização do contorno do modelo. Estes algoritmos, entretanto, são de implementação complexa e normalmente não são utilizados para elementos finitos.

Neste trabalho, a geração de malha de elementos finitos é feita combinando-se as técnicas de *quadtree* e de contração do contorno segundo um critério de Delaunay, de forma a manter as qualidades de rapidez e de transição suave da técnica de *quadtree*, aliadas a uma conformação com uma discretização fornecida para o contorno da região.

2.2 *Modelo Inicial de Elementos Finitos*

A estratégia de auto-adaptação proposta assume que existe uma malha inicial para o modelo de elementos finitos. Além disso, a estratégia precisa de uma descrição geométrica de cada uma das curvas que compõem o contorno do modelo e de suas regiões, bem como a descrição das condições de contorno (condições de suporte e solicitação) destas curvas e dos atributos (material, espessura, etc.) das regiões. Esta descrição pode ser feita através de arquivos texto em formato pré-definido (ver apêndice A). A malha inicial pode ser gerada usando um pré-processador gráfico existente (Mtool, 1992), e também fornecida através de um arquivo em formato padrão. Outra opção é gerar a malha automaticamente com base em uma discretização a priori das curvas fornecidas.

As seções seguintes descrevem todas as fases do processo proposto, adotando-se um exemplo hipotético simples, que tem como objetivo uma melhor definição das fases. Trata-se de uma chapa em forma de "L", com deslocamentos impedidos na extremidade esquerda e uma carga distribuída vertical, de cima para baixo, na extremidade direita. A malha inicial deste exemplo é mostrada na figura 2.2.

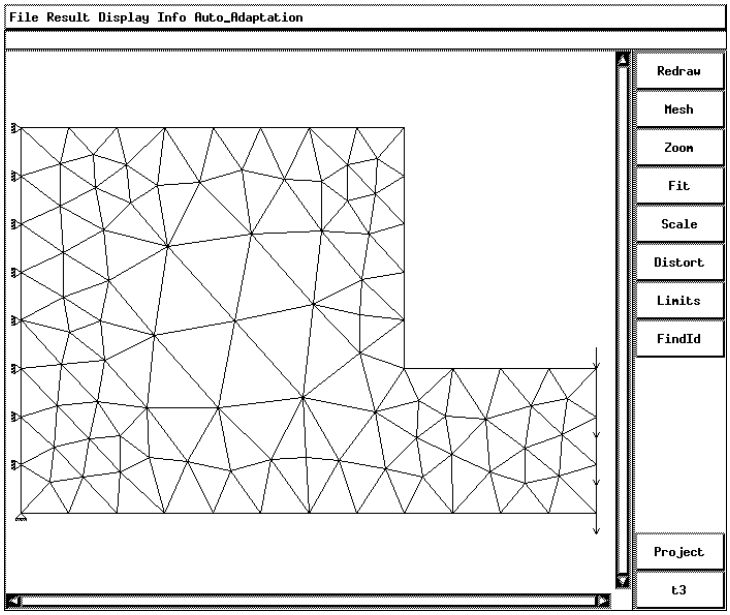


Figura 2.2: Exemplo e sua malha inicial.

2.3 *Análise por Elementos Finitos*

O modelo inicial deve passar por uma primeira análise, para que se tenha resultados numéricos necessários para o processo de refinamento, já que se trata de uma estratégia de auto-adaptação *a posteriori*.

O módulo de análise do sistema, desenvolvido em um trabalho anterior desta linha de pesquisa (Guimarães, 1992), utiliza uma disciplina de programação orientada a objetos, que permite a flexibilidade de análise de todos os tipos de elementos triangulares ou quadrilaterais, sem restrição a um certo tipo de elemento na malha. Neste trabalho, este módulo foi modificado para incorporar a avaliação do erro numérico, conforme será descrito no capítulo 3. O resultado da análise, junto com a descrição das curvas, regiões e de seus atributos, vão servir de requisitos para o refinamento da malha a ser efetuado.

O módulo de análise numérica é requisitado, em cada etapa do processo, após a malha ser refinada devido à avaliação de erro da etapa anterior. Os arquivos de descrição das curvas, regiões e seus atributos, porém, só intervirão no processo na primeira análise. Para tanto, segundo o paradigma de programação orientada a objetos, no módulo auto-adaptativo foi criada uma classe *Curva*, que armazena as informações geométricas e mecânicas referentes às curvas, conforme descrito no capítulo 4.

2.4 *Discretização do Contorno do Domínio*

Na estratégia proposta, para a geração auto-adaptativa da nova malha no domínio do modelo é necessário que se tenha o contorno das regiões do domínio formado por curvas de bordo já discretizadas. Portanto, a primeira fase do processo proposto é a discretização dessas curvas. Isto é feito com base nas propriedades geométricas e nos tamanhos característicos dos elementos de bordo (vizinhos às curvas) determinados a partir da estimativa de erro calculada pelo módulo de análise.

Esta discretização das curvas de bordo é feita independentemente da discretização do domínio do modelo e é uma das vantagens do processo proposto, pois isto vai resultar em uma discretização mais regular no contorno e consistente com as características geométricas de suas curvas.

O algoritmo usado para refinar o contorno é uma versão unidimensional (em cada curva) do algoritmo utilizado para refinar o domínio, baseado na técnica de *quadtree*. O refinamento de cada curva utiliza uma técnica de enumeração espacial recursiva, baseada em uma estrutura de dados de árvore binária (*binary tree*), que será descrita em detalhes no capítulo 4. Este processo de refinamento é feito para todas as curvas do contorno e vai definir a nova discretização para a geração da nova malha no domínio. A figura 2.3 mostra o resultado do refinamento de bordo para o exemplo estudado.

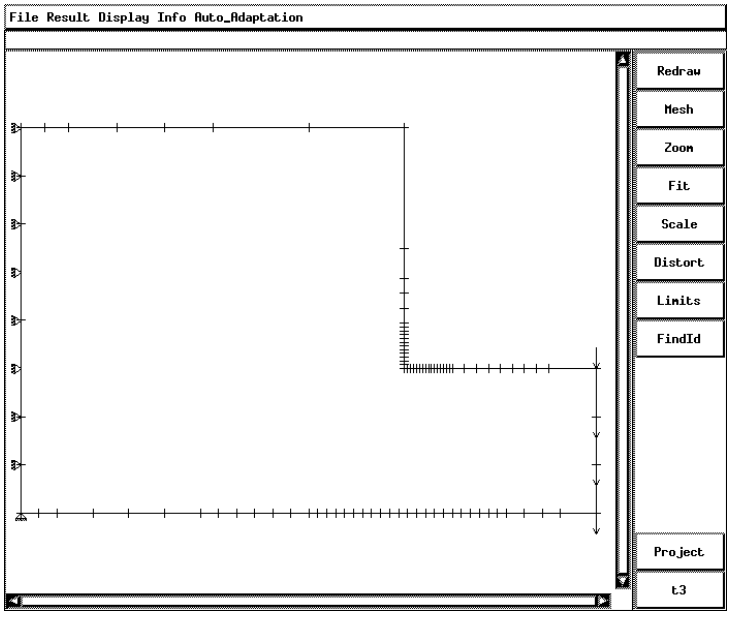


Figura 2.3: Discretização do contorno.

Ao final do processo de refinamento das curvas de bordo, refaz-se consistentemente as condições de contorno. Por exemplo, para uma curva com carga uniformemente distribuída, calcula-se o novo carregamento nodal equivalente consistente com o novo refinamento. Por isso, a estratégia proposta necessita das descrições das condições de suporte e de carregamento das curvas. A disciplina de programação orientada a objetos utilizada no módulo auto-adaptativo faz com que esse tratamento seja genérico para todos os tipos de curvas (linha reta, linha poligonal, arco de círculo, curva de Bézier, etc.), o que permite a flexibilidade de tratar domínios com geometria definida por quaisquer tipos de curvas. Esse processo de tratamento de atributos também será descrito no capítulo 4.

2.5 *Geração da Malha no Interior do Domínio*

Com as curvas do domínio já com a nova discretização baseada no erro numérico (erro de discretização) existente no modelo, passa-se à fase da geração da nova malha. O algoritmo usado para refinar o domínio é baseado na técnica de *quadtree*.

Esta técnica foi estendida neste trabalho para que a geração de malha no interior de uma região fique conformada com uma discretização fornecida para o seu contorno. Isto é feito gerando-se a malha no interior pela técnica de *quadtree*, deixando-se uma faixa próxima ao contorno para ser discretizada por uma técnica de triangulação por Delaunay. Este procedimento tem vantagens em relação a outros algoritmos auto-adaptativos (Baehmann, 1989; Barbalho, 1994), onde a discretização de bordo é definida pela atuação da estrutura de dados usada na discretização do domínio na fronteira do modelo. Conforme mencionado anteriormente, essa conformação com uma discretização de bordo fornecida é importante, pois assim pode-se combinar diferentes técnicas de algoritmos de geração de malha e tratar o problema apenas modificando a malha localmente. O algoritmo de geração da malha no domínio é descrito detalhadamente no capítulo 5.

O tamanho da célula da *quadtree* é controlado por dois fatores: o tamanho dos segmentos das curvas do contorno discretizadas e o tamanho característico dos elementos no interior da região ditado pela análise de erro numérico. Observa-se que este processo não só refina regiões onde se espera uma discretização mais expressiva, como "desrefina" regiões que não necessitam de tanto refinamento como o feito na malha original.

A figura 2.4 mostra a nova malha, onde se pode observar esse aspecto, além de notar-se a boa transição na malha, que é uma característica do algoritmo de *quadtree*,

pois ele realmente "sente" a discretização do contorno, já que a criação da malha é fortemente influenciada por ela.

Essa nova malha, com os atributos consistentemente refeitos, é reanalisada pelo módulo de análise; o ciclo continua até que o critério desejado seja atendido e se obtenha uma malha final otimizada.

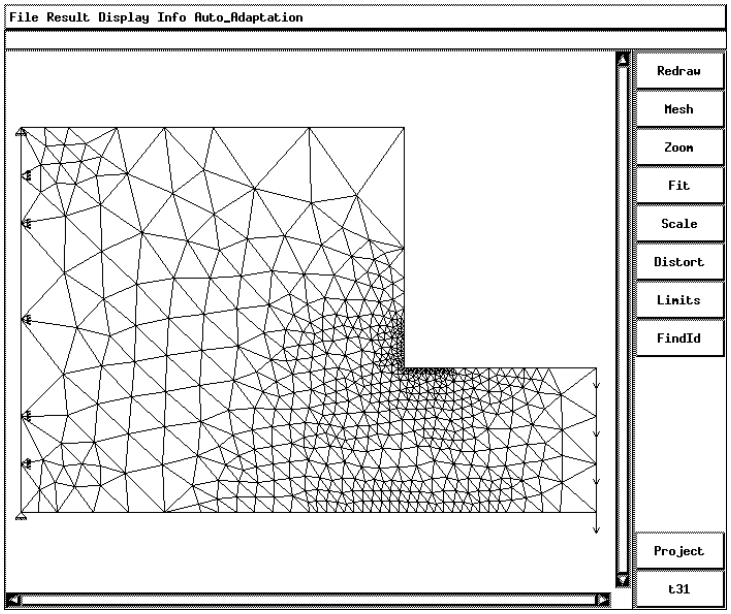


Figura 2.4: Nova malha gerada.

2.6 *Gerenciamento dos Dados*

Para o gerenciamento da geração auto-adaptativa da malha, é utilizada uma estrutura de dados que consiste de uma lista de regiões, onde cada uma possui uma lista de identificadores das curvas que delimitam a sua fronteira. Como a discretização da fronteira é que orienta a geração da malha naquela região, regiões que compartilham uma mesma curva têm assegurada a compatibilização em suas fronteiras. Essa lista de regiões é que vai orientar a geração da nova malha do modelo. Além disso, cada região possui também uma lista de seus atributos (material, espessura, etc.) armazenados a partir da leitura dos arquivos de descrição definidos para o modelo, pois, como para as curvas do modelo, os elementos das regiões devem ter seus atributos consitentemente refeitos para possibilitar a reanálise.

Para cada região deve-se então seguir um mesmo procedimento para o desenvolvimento da estratégia de auto-adaptação. Primeiro, percorre-se a lista de curvas da região, já rediscretizadas e com seus atributos refeitos, e, através dos seus identificadores, prepara-se os dados para o algoritmo de geração da malha. Este algoritmo, por sua vez, devolve uma malha que deve ser incluída na estrutura de dados da malha global de elementos finitos que está sendo gerada. Em seguida, os atributos de região são reaplicados nos novos elementos criados, de maneira que os novos elementos de uma região herdam os mesmos atributos dessa região na malha anterior.

Esse procedimento é repetido para todas as regiões e obtém-se uma nova malha para se refazer a análise. Um aspecto aqui se torna importante: como o processo é repetido região por região e o algoritmo de geração de malha por *quadtree* necessita do erro da malha para uma fase da criação da árvore, a malha antiga e a malha nova de uma região co-existem durante o processo, com a diferença que a nova malha vai se formando à medida que uma nova região é usada, e a malha antiga é liberada quando a etapa acaba.

3

Organização de Classes para Análise por Elementos Finitos

Este capítulo descreve a organização de classes do módulo de análise por elementos finitos usado no sistema. Este módulo, desenvolvido sob uma filosofia de programação orientada a objetos, foi implementado, dentro da linha de pesquisa em que este sistema auto-adaptativo está inserido, por trabalho anterior (Guimarães, 1992) e modificado para incluir a estimativa de erro necessária ao sistema. É importante ressaltar que o paradigma de programação orientada a objetos utilizado foi fundamental para esta expansão do módulo, uma vez que somente foi necessário introduzir novas classes e métodos, sem precisar alterar toda a estrutura anterior, que foi totalmente reutilizada para o cálculo de erro.

3.1 *Conceitos Básicos de Programação Orientada a Objetos*

Os dois princípios básicos da programação orientada para objetos são a *encapsulação* e a *herança*. Estes princípios são caracterizados nesta seção que está fundamentada principalmente no trabalho de Cox (1991). Aqui são repetidos alguns conceitos colocados por Guimarães (1992) por uma questão de completitude deste trabalho.

3.1.1 *Relação Cliente-Servidor*

Para apresentar o princípio da encapsulação, é fundamental a definição do conceito da relação "Cliente – Servidor".

Começa-se imaginando um programa que se divide num módulo principal e em vários outros, secundários, que executam tarefas específicas, sempre que solicitados pelo módulo principal. Ao módulo principal, dá-se o nome de *cliente*; os demais são os *servidores*.

Na programação convencional, a relação entre cliente e servidores se materializa através da transferência de dados, onde as duas partes possuem o conhecimento das estruturas de dados e dos operadores. Isto é, o cliente decide *como* cada operação deve ser executada, chamando a rotina específica de cada servidor.

Quando se vai para o ambiente orientado a objetos, esta relação se modifica, já que, agora, o cliente deixa de decidir como cada operação deve ser executada e passa a solicitar a execução de tarefas através de *mensagens* enviadas ao servidor que apenas definem a tarefa. Em outras palavras, o cliente apenas decide *qual* operação deve ser executada acionando genericamente (através de mensagens) a operação a ser executada.

Para que isto se torne possível, cada servidor cria uma fronteira envolvendo seus procedimentos (rotinas), chamados de *métodos*, e dados. Isto é conhecido como princípio de *encapsulação*. A encapsulação transfere o conhecimento e a responsabilidade da execução das tarefas para cada servidor. Com isto, o cliente lida apenas com algoritmos globais que acionam os métodos genéricos através de mensagens, cabendo aos servidores manipular seus procedimentos e dados privados.

3.1.2 *Classes e Objetos*

Para definir o princípio da *herança*, é fundamental caracterizar um outro conceito importante da programação orientada a objetos: o conceito de *classe*. Pode-se entendê-la como uma categoria ou grupo de objetos que possuem o mesmo tipo de dados e procedimentos (métodos). Para formar um grupo qualquer, é fundamental que seus elementos possuam características comuns (as mesmas que definem o grupo), mas não se pode esquecer que são diferentes entre si, apesar de pertencerem à mesma classe.

Os *objetos*, no tipo de programação, podem ser vistos como elementos de uma classe, também chamados de *instâncias*. As linguagens de programação mais modernas, como C e Pascal, permitem a definição de tipos de dados, formando estruturas com dados agregados. A relação existente entre um objeto e a sua classe é a mesma que se estabelece entre uma variável declarada e o seu tipo, ou seja, a variável tornou-se uma instância daquela estrutura definida. Cada objeto forma uma entidade completa com seus próprios dados. Quando uma tarefa é executada, o objeto utiliza os procedimentos de sua classe e sua área de dados privada. Como nenhum detalhe deste processo é transparente ao cliente, qualquer alteração nos procedimentos não implica em conseqüências fora do seu domínio, ou seja, toda mudança é localizada.

3.1.3 *Hierarquia de Classes*

Como todo conjunto, uma classe pode ser composta de *subclasses*, que seriam seus subconjuntos. Da relação inversa, nasce a *superclasse*.

A hierarquia sugerida por estas relações concretiza-se através do princípio de herança. Por definição, para existir, uma classe deve possuir métodos. Qualquer classe hierarquicamente inferior, ou seja, uma subclasse, herdará o modelo de dados e os métodos da sua superclasse. Isto significa que os métodos estarão disponíveis para utilização, sem repetição de código. Portanto, uma classe possui, além de seus métodos, os herdados de uma eventual superclasse. Como a superclasse possui algumas subclasses, é comum que alguns de seus métodos não interessem a uma determinada subclasse. Por isso, a herança é seletiva, isto é, cada subclasse seleciona os métodos que deseja herdar. Além disto, uma subclasse pode redefinir um método herdado de uma superclasse. Este conceito é chamado de *overwriting*.

Conforme mencionado anteriormente, a comunicação entre cliente e servidores se dá através de mensagens enviadas aos objetos que identificam as tarefas a serem executadas. A classe de um objeto, ao receber uma mensagem, consulta seu *protocolo*, que é uma lista de todas as suas mensagens relacionadas com seus respectivos métodos, identificando o método correspondente. A seleção do método depende da classe do objeto que recebe a mensagem e da sua posição na hierarquia de classes.

Portanto, o cliente envolve-se somente com a existência dos objetos e suas mensagens, deixando para os servidores a manipulação das classes, suas categorias e heranças.

3.2 *Organização de Classes do Módulo de Análise*

A fase de projeto de um sistema é fundamental, pois é lá que se estrutura toda a sua organização. O desenvolvimento envolve a especificação e a implementação da representação de dados do programa. Em programação orientada a objetos, são três as etapas a serem vencidas (Fenves, 1990):

- Seleção de classes: Determina-se as classes de objetos necessárias para representar o problema e a sua solução; cria-se subclasses para aumentar o grau de especificidade de representação do problema.
- Especificação das classes: Define-se as operações nos objetos de uma classe ao se especificar o que faz cada mensagem, ou seu método associado; a especificação contém uma descrição precisa da operação invocada por cada mensagem.
- Implementação das classes: Seleciona-se variáveis locais para objetos; para cada mensagem, programa-se um método para executar a operação especificada.

O módulo de análise numérica, na sua concepção original para análise linear elástica (Guimarães, 1992), enfocou o procedimento mais importante a ser avaliado que é a montagem da matriz de rigidez do elemento, sob a qual foi especificada sua organização de classes.

O problema básico a ser resolvido, então, é o cálculo da matriz de rigidez de um elemento finito. A obtenção canônica desta matriz está calcada na expressão (3.1) (Zienckiewicz, 1989):

$$\Big[K\Big] = \int_{vol} \Big[B\Big]^T \Big[E\Big] \Big[B\Big] dv, \tag{3.1}$$

onde

$\Big[K\Big]$ = matriz de rigidez do elemento.

$\Big[B\Big]$ = matriz de compatibilidade de deslocamentos (relaciona deformações internas com deslocamentos nodais).

$\Big[E\Big]$ = matriz tensão-deformação do material e do tipo de análise.

A expressão (3.1) é válida para qualquer tipo de elemento. Entretanto, este trabalho está direcionado basicamente para elementos finitos baseados nas formulações *isoparamétricas* e *subparamétricas* (Cook, 1989), onde a expressão (3.1) é usualmente avaliada a partir de uma integração numérica de Gauss, como mostra a expressão (3.2):

$$\Big[K\Big] = \sum_{i=1}^{n} \Big[B\Big]^T \Big[E\Big] \Big[B\Big] \Big|Ji\Big| \omega_i, \tag{3.2}$$

onde

n = número de pontos de integração de Gauss.

$\Big|Ji\Big|$ = determinante da matriz Jacobiana no ponto de integração i.

ω_i = peso associado ao ponto de integração i.

Além disso, a análise do problema de cálculo do carregamento nodal equivalente, mostrado na sua forma canônica pela expressão (3.3) e introduzido neste trabalho, influenciou na criação de novas classes e na reorganização de algumas existentes no trabalho original. A nova organização de classes é mostrada na seqüência deste capítulo. A implementação do estimador de erro utilizado na estratégia auto-adaptativa baseia-se nesta nova organização.

$$\Big\{R\Big\} = \sum_{i=1}^{n} \Big[N\Big]^T \Big\{p\Big\} \Big|Ji\Big| \omega_i, \tag{3.3}$$

onde

$\{R\}$ = vetor do carregamento nodal equivalente.

n = número de pontos de integração de Gauss.

$[N]$ = matriz das funções de forma do elemento.

$\{p\}$ = vetor de carregamento distribuído aplicado.

$|Ji|$ = determinante da matriz Jacobiana no ponto de integração i.

ω_i = peso associado ao ponto de integração i.

3.2.1 *Seleção e Especificação das Classes*

O processo de seleção das classes foi baseado no estudo das entidades envolvidas no cálculo da matriz de rigidez, procurando-se determinar como as informações fluem no sistema e quem as detem. Os eventos a serem avaliados neste aspecto são a montagem da matriz $[B]$, a montagem da matriz $[E]$ e a integração numérica. Ao avaliar-se o procedimento de montagem da matriz $[B]$, vê-se que suas dimensões dependem do elemento finito, pois é função do seu número de nós, e do tipo de análise a que ele está submetido. Os termos não nulos desta matriz dependem das funções de forma do elemento finito e de suas derivadas em relação aos eixos cartesianos, avaliadas em seus pontos de integração. A distribuição destes valores entre os termos da matriz depende do tipo de análise que solicita o elemento. O processo de montagem da matriz $[E]$ depende diretamente do material e do tipo de análise a que o elemento está submetido. Sua dimensão é definida exclusivamente pelo tipo de análise, que também afeta a distribuição dos valores característicos do material, como o coeficiente de Poisson e o módulo de elasticidade, entre os termos da matriz que dependem destas grandezas. Finalmente, o processo de integração de Gauss depende da ordem de integração, surgindo a figura da matriz Jacobiana que depende unicamente da forma do elemento finito.

No trabalho original de Guimarães (1992), baseado em todos os aspectos descritos acima, foram organizadas as classes que estruturam a solução do problema, dentro do paradigma da programação orientada a objetos, denominadas classe *Elemento*, classe *Material*, classe *Modelo de Análise* e classe *Gauss*.

No presente trabalho, uma nova classe, chamada *Elemento de Carregamento*, foi criada para gerenciar o cálculo das cargas equivalentes nodais. As instâncias desta

classe podem ser vistas como elementos finitos "fictícios", que só existem para transportar as solicitações distribuídas nos elementos para os seus nós. A análise do cálculo do carregamento nodal equivalente mostra que também se necessita das funções de forma e suas derivadas, do mesmo modo que no cálculo da matriz de rigidez para o elemento. Portanto, a nova classe *Elemento de Carregamento* precisa dos métodos que fornecem as funções de forma do elemento finito, anteriormente fornecidas pela classe *Elemento*. Isto fez surgir uma nova classe, chamada *Forma do Elemento*, que fica encarregada de fornecer essas informações, pois sem isso seria necessário repetir, na classe *Elemento de Carregamento*, todos os métodos relativos às funções de forma existentes na classe *Elemento*. Com isso, tanto a classe *Elemento* como a classe *Elemento de Carregamento* acessam os métodos dessa nova classe, mantendo a consistência da filosofia de programação.

Além disso, o programa sofreu uma expansão para abordar outros tipos de análise, o que resultou no aparecimento de outras duas classes: a classe *Algoritmos para Solução de Sistemas Não-Lineares*, e a classe *Algoritmos para Solução de Problemas de Autovalor*.

Uma grande modificação estrutural também foi feita com a criação da classe *Controlador de Análise*, que orienta todos as possíveis análises para seus respectivos procedimentos, controlando todo o processo de entrada de dados, montagem do sistema de equações, solução, e saída de dados, organizando globalmente o módulo de análise numérica. Ela consiste, em seu estado atual, em duas subclasses, uma para problemas mecânicos e outra para análise transiente de temperatura, esta introduzida por trabalho desenvolvido por Barros (1994).

Resumindo, a nova organização de classes é composta pelas classes *Forma do Elemento*, *Elemento*, *Elemento de Carregamento*, *Gauss*, *Material*, *Modelo de Análise*, *Algoritmos para Solução de Sistemas Não-Lineares*, *Algoritmos para Solução de Problemas de Autovalor* e *Controlador de Análise*. A especificação dessas classes na sua versão atual e as mudanças nas classes originais são detalhadas nas sub-seções a seguir.

A utilização da filosofia de programação se mostrou de fundamental importância na expansão do módulo de análise numérica, fato comum em se tratando do desenvolvimento incremental a que programas de elementos finitos estão sujeitos.

3.2.1.1 *Classe Forma do Elemento*

É natural se imaginar uma classe *Elemento* com a responsabilidade de fornecer ao cliente

(algoritmo que utiliza a classe) todas as informações necessárias à resolução do problema que dependam exclusivamente do elemento finito. Como dito, originalmente (Guimarães, 1992) a classe *Elemento* foi criada armazenando todas as informações geométricas do elemento e o conhecimento das funções de forma e suas derivadas. A principal modificação estrutural em relação à organização anterior foi a criação da classe *Forma do Elemento*, que tirou da classe *Elemento* toda a responsabilidade pelo fornecimento de aspectos relativos às funções de forma, suas derivadas, etc. Como a abrangência da classe *Forma do Elemento* é ampla, foram criadas subclasses. A figura 3.1 mostra a organização das subclasses da classe *Forma do Elemento*; nota-se a existência de quatro ramos de hierarquia criados para suportar as tarefas destinadas a essa classe.

Subclasses da classe Forma do Elemento:

LINE		
	LINE2	linha com 2 nós
	LINE3	linha com 3 nós
	LINE4	linha com 4 nós
SURFACE		
	TRIA3	triângulo com 3 nós
	TRIA6	triângulo com 6 nós
	TRIA10	triângulo com 10 nós
	QUAD4	quadrilátero com 4 nós
	QUAD5	quadrilátero com 5 nós
	QUAD6	quadrilátero subparamétrico com 6 nós
	QUAD8	quadrilátero com 8 nós
	QUAD9	quadrilátero Lagrangeano com 9 nós
	QUADS	quadrilátero Serendipity geral
	INFINITE	elementos infinitos
SOLID		
	BRICK8	paralelepípedo com 8 nós
	BRICK20	paralelepípedo com 20 nós
	TETR4	tetraedro com 4 nós
	TETR10	tetraedro com 10 nós
	WEDGE6	cunha com 6 nós
	WEDGE15	cunha com 15 nós
BAR		barra de treliça e viga com 2 nós

Figura 3.1: Subclasses implementadas da classe Forma do Elemento.

A principal vantagem dessa organização de classes é que o desenvolvimento incremental de uma família de elementos se torna fácil e de baixo custo; para a implementação de uma nova forma de elemento, basta avaliar dentro da organização da classe onde ele se encaixa e implementar somente os seus métodos específicos, pois a relação hierárquica da classe, caracterizada pela herança de métodos, faz com que o novo elemento herde automaticamente todos os outros métodos de que necessita de classes hierarquicamente superiores.

A especificação de uma classe envolve o seu protocolo (lista de mensagens) e uma descrição sumária e precisa de cada método invocado por uma mensagem. A

figura 3.2 mostra a especificação da classe *Forma do Elemento*. Alguns métodos não são implementados em todas as subclasses. Por exemplo, praticamente todas as subclasses de formas de superfície (SURFACE) herdam da superclasse os métodos de computação da matriz do Jacobiano e das derivadas das funções de forma em relação às coordenadas cartesianas.

É importante descrever os dados de um objeto típico da classe Forma do Elemento. O objeto contém em sua estrutura a incidência do elemento na lista de nós.

Métodos para a classe Forma do Elemento:	
Init	Inicializa a classe.
New	Cria uma instância (objeto) da classe.
Métodos para objetos:	
Connectivity	Retorna a incidência nodal do objeto.
DerivMaprst	Retorna as derivadas das funções de mapeamento geométrico do objeto em relação aos eixos locais avaliadas no ponto fornecido.
DerivShprst	Retorna as derivadas das funções de forma do objeto (para interpolação de deslocamentos) em relação aos eixos locais no ponto fornecido.
Derivxyz	Retorna as derivadas das funções de forma do objeto em relação aos eixos globais avaliadas no ponto fornecido.
Free	Libera toda a memória usada por um objeto da classe.
GaussType	Retorna a subclasse da classe Gauss que é consistente com a forma do elemento.
GetEdge	Retorna o tipo da forma, número de nós, e conectividade de uma aresta de uma forma dada para um par de nós de quinas.
GetFace	Retorna o tipo da forma, número de nós, e conectividade de uma face de uma forma dada para um par de nós de quinas e um terceiro nó definindo a direção da normal à face.
GetDomain	Retorna o tipo da forma, número de nós, e conectividade para uma forma dada.
Jacobian	Retorna a matriz Jacobiana do objeto, sua inversa e seu determinante.
LocalSys	Retorna a matriz de rotação para a relação sistema local/global.
Make	Faz uma forma de elemento para uma conectividade dada.
MapFunc	Retorna as funções de mapeamento geométrico avaliadas no ponto fornecido.
NodalCoord	Retorna as coordenadas cartesianas dos nós do objeto.
NumMapNodes	Retorna o número de nós do objeto utilizados no mapeamento de sua geometria.
NumShpNodes	Retorna o número de nós do objeto utilizados para interpolar seus deslocamentos.
OptmRespOrder	Retorna a ordem de integração ótima (número de pontos em cada direção) para avaliação de resposta (tensões).
Print	Imprime os dados sobre a forma do elemento.
Read	Lê os dados da forma dos elementos de um arquivo fornecido.
ShpFunc	Retorna as funções de forma avaliadas no ponto fornecido.
TRMatrix	Retorna a matriz de transformação utilizada para extrapolar as tensões dos pontos de integração de Gauss para os nós do objeto.

Figura 3.2: Protocolo e descrição dos métodos da classe Forma do Elemento.

3.2.1.2 *Classe Gauss*

O objetivo desta classe é organizar o processo de integração de Gauss. Ela foi subdividida em cinco subclasses, onde cada uma possui os conhecimentos específicos para proceder a integração de acordo com o domínio do elemento finito. A figura 3.3 descreve suas suclasses e métodos.

Subclasses da classe Gauss:	
LINE-QUADRATURE	quadratura sobre um domínio linear
TRIA-QUADRATURE	quadratura sobre um domínio triangular
QUAD-QUADRATURE	quadratura sobre um domínio quadrilateral
TETR-QUADRATURE	quadratura sobre um domínio tetraédrico
CUBE-QUADRATURE	quadratura sobre um domínio cúbico
WEDGE-QUADRATURE	quadratura sobre um domínio em forma de cunha
Métodos para a classe:	
Init	Inicializa a classe.
Métodos para objetos:	
NumPts	Retorna o número de pontos de integração para uma ordem fornecida.
Points	Retorna as coordenadas naturais (locais) dos pontos de integração e seus pesos para uma ordem fornecida.

Figura 3.3: Subclasses, protocolo e descrição dos métodos da classe Gauss.

3.2.1.3 *Classe Material*

O material foi considerado uma entidade tridimensional, onde o tipo de análise (classe *Modelo de Análise*) é que define se o elemento finito será tratado uni, bi ou tridimensionalmente, quando da solução do problema. A classe armazena as grandezas que caracterizam o material (módulo de elasticidade, coeficiente de Poisson, etc.). Na atual implementação foram mantidas as subclasses originais: material isotrópico e material ortotrópico. Na expansão do sistema, alguns novos métodos foram introduzidos.

Subclasses da classe Material:	
ISO	Material isotrópico
ORTHO	Material ortotrópico
Métodos para a classe:	
Init	Inicializa a classe.
New	Cria uma instância (objeto) da classe.
Métodos para objetos:	
Ematrix	Retorna a matriz constitutiva do objeto (tridimensional).
Free	Libera toda memória usada pela instância.
GetDat	Retorna dados do material.
GetLabel	Retorna o nome do material.
GetType	Retorna tipo do material.
GKMatrix	Retorna matriz de condutividade tridimensional.
Read	Preenche um objeto com dados lidos de um arquivo fornecido.
Write	Imprime os dados do objeto.
Rho	Retorna a densidade do objeto.
SficHeat	Retorna parâmetro de calor específico.

Figura 3.4: Subclasses, protocolo e descrição dos métodos da classe Material.

3.2.1.4 *Classe Modelo de Análise*

Talvez a mais importante contribuição desta linha de pesquisa para esta área tenha sido a criação da classe *Modelo de Análise*. Ela foi criada com a função de "limpar" todas as outras, livrando-as de qualquer tipo de interdependência. Ela também tem a função de evitar que o cliente possua qualquer conhecimento específico sobre as características do elemento e/ou particularidades do tipo de análise efetuada. Por exemplo, comparando-se uma análise de estado plano de tensões com uma análise axissimétrica, se não existisse a classe *Modelo de Análise*, na classe *Elemento* seria preciso definir um método específico para a montagem da matriz $[B]$ para cada um desses tipos de análise. A classe *Modelo de Análise* evita este tipo de repetição que fere os princípios da filosofia de programação adotada. A figura 3.5 descreve as subclasses e métodos desta classe.

Subclasses da classe Modelo de Análise:

TRUSS3D	análise de treliça espacial
BEAM3D	análise de pórtico espacial
PLANE-STRESS	análise de estado plano de tensão
PLANE-STRAIN	análise de estado plano de deformação
AXISSYMETRIC	análise axissimétrica
PLATE-BENDING	análise de placas à flexão
PLATE-SHRBEND	análise de placas à flexão com cisalhamento
SHELL	análise de cascas
SOLID	análise de sólidos
TEMPERATURE2D	análise de temperatura bidimensional
TEMPERAXISYMM	análise axissimétrica de temperatura
TEMPERATURE3D	análise de temperatura tridimensional

Métodos para a classe:

Init	Inicializa a classe.

Métodos para objetos:

CMatrix	Retorna a matriz tensão-deformação para uma dada matriz constitutiva tridimensional do material.
DimBMatrix	Retorna o número de linhas da matriz $[B]$.
DofGlobDir	Retorna as direções globais consistentes com o tipo de análise.
GaussStresses	Retorna as tensões avaliadas em pontos de Gauss.
KMatrix	Retorna a matriz de condutividade.
MountBMatrix	Gerencia a montagem da matriz $[B]$.
NodalDispl	Monta o vetor de deslocamentos nodais de um vetor de deslocamentos do elemento.
NodalStresses	Retorna as tensões nodais avaliadas a partir das tensões nos pontos de Gauss para uma dada matriz de transformação (extrapolação).
NumDofNode	Retorna o número de graus de liberdade por nó do problema.
PrincStress	Calcula as tensões principais de um dado tensor de tensões nas coordenadas cartesianas.
PrintStress	Imprime as tensões nodais e em pontos de Gauss por elemento.
RigidCoeff	Ajusta o fator de rigidez (fator que multiplica o triplo produto de matrizes no cálculo da matriz de rigidez) de acordo com o tipo de análise.
FloorNode	Atualiza a matriz de rigidez do objeto de acordo com as equações de restrições de nós dependentes.
OffSetNode	Atualiza a matriz de rigidez do objeto de acordo com as informações de offset dos nós do objeto.
SkewNode	Atualiza a matriz de rigidez do objeto de acordo com as informações dos nós nos apoios inclinados.

Figura 3.5: Subclasses, protocolo e descrição dos métodos da classe Modelo de Análise.

3.2.1.5 *Classe Elemento*

Com a criação da classe *Forma do Elemento*, a classe *Elemento* ficou mais simples em relação à implementação anterior, pois não precisa mais fornecer os métodos que foram englobados pela nova classe. Isto acarretou até na mudança na hierarquia de subclasses da classe. Por exemplo, foi criada uma subclasse Paramétrica, que engloba todos os elementos paramétricos existentes nas análises, sem distinção se bi ou tridimensional, que fica a cargo da atuação da *Forma do Elemento*. As subclasses atuais e os métodos da classe *Elemento* são descritos na figura 3.6.

Subclasses da classe Elemento:

PARAMETRIC	elementos paramétricos
TRUSS	barra de treliça tridimensional com 2 nós
BEAM	barra de quadro tridimensional com 2 nós
DKT	placa triangular de Kirchhoff discreto com 3 nós
DKQ	placa quadrilateral de Kirchhoff discreto com 4 nós
FSHELL3	casca triangular com 3 nós (T3 + DKT)

Métodos para a classe:

Init	Inicializa a classe.
New	Cria uma instância (objeto) da classe.

Métodos para objetos:

BMatrix	Retorna a matriz deformação-deslocamento do objeto avaliada no ponto fornecido.
Capacity	Retorna a matriz de capacitância para temperatura.
Conductivity	Retorna a matriz de condutividade térmica.
Error	Retorna o erro no elemento.
Free	Libera toda a memória usada pelo elemento.
Mass	Retorna a matriz de massa do objeto.
NMatrix	Retorna a matriz das funções de forma do objeto avaliada no ponto fornecido.
NormError	Retorna a norma de erro em tensões no elemento.
NormDisp	Retorna a norma de erro em deslocamentos no elemento.
Order	Retorna a ordem de integração do objeto (número de pontos de integração em cada direção).
Print	Imprime os dados do objeto.
Read	Lê os dados do objeto no arquivo fornecido e os armazena.
ResponseOrder	Retorna a ordem de integração para computar os resultados.
Stiffness	Retorna a matriz de rigidez do objeto.
Stress	Retorna as tensões nos pontos de gauss e nodais do objeto.
Thickness	Retorna a espessura dos elementos planos ou de casca e 1.0, se for sólido.

Figura 3.6: Subclasses, protocolo e descrição dos métodos da classe Elemento.

Com relação aos dados de um objeto dessa classe, além do seu número e da espessura e ordem de integração, ele contém identificadores para os correspondentes objetos das classes *Forma do Elemento*, *Material*, *Modelo de Análise* e *Gauss*.

3.2.1.6 *Classe Elemento de Carregamento*

Essa classe foi criada para cálculo de carregamentos nodais equivalentes e/ou fluxos nodais equivalentes. Suas subclasses e métodos são descritos na figura 3.7.

Os dados de um objeto típico dessa classe englobam além dos valores de carregamento aplicados, identificadores para os correspondentes objetos das classes *Forma do Elemento*, *Elemento* (do elemento finito associado) e *Gauss*.

Subclasses da classe Elemento de Carregamento:

FORCE-UNIFORM	carga distribuída uniformemente
FORCE-VARIABLE	carga distribuída variavelmente
BEAM-FORCE	carga de viga concentrada
BEAM-UNIFORM	carga de viga linearmente distribuída
HEAT-FLUX-UNIFORM	fluxo de calor distribuído uniformemente
HEAT-FLUX-VARIABLE	fluxo de calor distribuído variavelmente
HEAT-CONVECTION-UNIFORM	convecção distribuída uniformemente

Métodos para a classe:

Init	Inicializa a classe.
New	Cria uma instância (objeto) da classe.

Métodos para objetos:

EquivForce	Calcula a carga equivalente nodal no elemento.
Evaluate	Avalia carga numa posição paramétrica do elemento e retorna os seus valores em um vetor.
EtransfMatrix	Calcula a matriz energia de transferência da carga do elemento, isto é, a contribuição para a matriz de coeficientes global (se aplicável).
EtransfParam	Retorna paramêtro de transferência de energia.
FixEndForc	Retorna ação de engastamento perfeito no elemento.
Free	Libera memória usada pelo objeto.
InqEtransf	Indaga energia de transferência da carga do elemento.
InqFixEndForc	Indaga contribuição da ação de engastamento perfeito do elemento para as forças internas do elemento finito.
LoadVal	Carrega os valores de uma carga do elemento baseado em dados fornecidos.
LocalSys	Avalia a matriz de rotação do sistema local localizado em um ponto fornecido.
NMatrix	Retorna a matriz de interpolação.
ReadVal	Preenche um objeto com dados lidos de uma arquivo fornecido.

Figura 3.7: Subclasses, protocolo e descrição dos métodos da classe Elemento de Carregamento.

3.2.1.7 *Classes de Algoritmos de Solução*

Com a expansão do sistema para abranger outros tipos de análise, foram criadas duas novas classes: a classe *Algoritmos para Solução de Sistemas Não-Lineares* e a classe *Algoritmos para Solução de Problemas de Autovalor*. A descrição dessas classes está mostrada nas figuras 3.8 e 3.9.

Subclasses da classe Algoritmos para Solução de Sistemas Não-Lineares:

LINEAR	Linear
SNR	Newton-Raphson
MNR	Newton-Raphson modificado
ARCLENGTH	Comprimento de Arco
TEMPER-STEADY	Temperatura em estado estacionário
TEMPER-DIRINTv	Análise transiente - Integração direta (forma d)
TEMPER-DIRINTd	Análise transiente - Integração direta (forma v)

Métodos para a classe:

Init	Inicializa a classe.

Métodos para objetos:

Solver	Avalia a solução iterativa incremental.

Figura 3.8: Subclasses, protocolo e descrição dos métodos da classe Algoritmos para Solução de Sistemas Não-Lineares.

Subclasses da classe Algoritmos para Solução de Problemas de Autovalor:

NONE-EIG	Problema sem autovalores
JACOBI	Método de Jacobi
HQR	Método HouseHolder-QR
SUBSPACE	Método da Iteração de Subespaços

Métodos para a classe:

Init	Inicializa a classe.

Métodos para objetos:

Values	Retorna os autovetores e autovalores do problema.
Print	Imprime os autovetores e autovalores no arquivo de relatório.
Pos	Imprime os autovetores e autovalores no arquivo de pós-processamento.

Figura 3.9: Subclasses, protocolo e descrição dos métodos da classe Algoritmos para Solução de Problemas de Autovalor.

3.2.1.8 *Classe Controlador de Análise*

Essa classe foi criada para organizar todo o processo de análise dos tipos exis-tentes e possíveis tipos a serem introduzidos. Ela é responsável pela leitura de dados, direcionamento para o tipo de análise desejada, e a saída dos dados de resultado do processo. Esta classe é de fundamental importância para controlar expansão do sistema, como a

inclusão da análise de temperatura (Barros, 1994). A descrição atual de suas subclasses e métodos está mostrada na figura 3.10.

Subclasses da classe Controlador de Análise:

STRUCTMECH	diretiva para problema de mecânica estrutural
TEMPDIFUSION	diretiva para problema de temperatura

Métodos para a classe:

Init	Inicializa a classe.
AnalysisType	Obtém o controlador de análise de acordo com o seu tipo.

Métodos para objetos:

CompEqs	Calcula as equações.
Profile	Monta o vetor perfil da matriz do sistema de equações.
EquivForce	Adiciona forças nodais equivalentes ao vetor de carga.
GblStiff	Calcula a matriz de rigidez global.
GblConduct	Calcula a matriz de condutividade global.
GblLoad	Calcula carga global.
AddSpring	Adiciona rigidez de mola à matriz de rigidez global.
AddPrescData	Adiciona dados prescritos ao sistema de equações.
GblMass	Calcula a matriz de massa global.
PrintHeader	Imprime o cabeçalho do arquivo de relatório.
PrintBegin	Imprime o começo do arquivo de relatório.
PrintNodes	Imprime os nós da malha no arquivo de relatório.
PrintMat	Imprime os materiais no arquivo de relatório.
PrintElem	Imprime os dados de elemento no arquivo de relatório.
PrintSprings	Imprime rigidez de mola no arquivo de relatório.
PrintPrescData	Imprime os dados prescritos no arquivo de relatório (deslocamentos prescritos e temperatura prescrita).
PrintSkew	Imprime apoios inclinados no arquivo de relatório.
PrintOffSets	Imprime offsets no arquivo de relatório.
PrintNodeFloor	Imprime nós de piso no arquivo de relatório.
PrintLoads	Imprime cargas no arquivo de relatório.
PrintDispl	Imprime os deslocamentos no arquivo de relatório.
PrintTemp	Imprime as temperaturas no arquivo de relatório.
PrintStress	Imprime as tensões no arquivo de relatório e suaviza as tensões para calcular a estimativa de erro.
PrintError	Imprime o erro no arquivo de relatório e prepara seus dados para a impressão no arquivo de pós-processamneto.
PosInData	Inclui dados iniciais no arquivo de pós-processamento.
PosDispl	Inclui deslocamentos calculados no arquivo de pós-processamento.
PosTemp	Inclui as temperaturas no arquivo de pós-processamento.
PosUpDate	Inclui todos os resultados finais da análise no arquivo de pós-processamento.

Figura 3.10: Subclasses, protocolo e descrição dos métodos da classe Controlador de Análise.

Alguns métodos fazem sentido para um tipo de controlador e outros não, como por exemplo o cálculo da matriz de condutividade para análise linear elástica num problema de mecânica estrutural; neste caso, as mensagens para essa subclasse simplesmente não são respondidas por ela, pois não se aplicam no seu escopo de atuação.

3.3 *Adequabilidade das Classes*

A adequabilidade das classes pode ser comprovada através do algoritmo genérico de cálculo da matriz de rigidez, como mostra a figura 3.11. O algoritmo, mostrado em pseudo-código de forma simplificada, corresponde ao método utilizado pela maioria das subclasses da classe Elemento que implementam elementos isoparamétricos e subparamétricos para diversos tipos de análise. Neste pseudo-código, um método é invocado através da função `msg()`, cujo primeiro argumento é o identificador do objeto, o segundo é o identificador da mensagem, e os outros os parâmetros da mensagem.

```
// obtém número de graus de liberdade por nó do objeto Modelo de Análise do elemento
numdofnode = msg( anmodel, NumDofNode )
// obtém o número de nós do elemento corrente do objeto Forma do Elemento do elemento
numnodes = msg( shpelem, NumMapNodes )
// calcula o número de graus de liberdade do elemento
ndof = numdofnode * numnodes
// obtém as coordenadas dos nós do elemento
msg( shpelem, NodalCoord, elemcoord )
// obtém a ordem de integração do elemento
msg( elem, Order, order )
// obtém o número de pontos de Gauss
numgausspts = msg( gauss, NumPts, order )
// obtém as coordenadas e os pesos do pontos de Gauss
msg( gauss, Points, order, gaussrst, gausswgt )
// obtém a matriz [E] do material
msg( mat, EMatrix, ematrix )
// obtém o número de componentes de tensão
numstrcmps = msg( anmodel, DimBMatrix )
// monta a matriz [C] para o tipo de análise
msg( anmodel, CMatrix, ematrix, cmatrix )

foreach gauss point [i], i = 1..numgausspts
     begin
     // obtém a matriz [B]
     msg( elem, BMatrix, gaussrst[i], elemcoord, detjac, bmatrix )
     // obtém as funções de forma avaliadas no ponto de Gauss corrente
     msg( shpelem, MapFunc, gaussrst[i], mapfunc )
     // obtém a espessura do elemento
     msg( elem, Thickness, thickness )
     // obtém o coeficiente de rigidez com base na espessura e o ajusta
     // de acordo com o tipo de análise
     msg( anmodel, RigidCoeff, numnodes, elemcoord, mapfunc, thickness, rigidcoeff )
     // calcula o coeficiente multiplicador do triplo produto
     coeff = rigidcoeff * detjac * gausswgt[i]
     // calcula o triplo produto e acumula na matriz de rigidez
     TripProdBtEB( numstrcmps, ndof, bmatrix, ematrix, coeff, elmstiff )
     end
```

Figura 3.11: Algoritmo de cálculo da matriz de rigidez $\left[K\right]$.

Outros exemplos que comprovam a adequabilidade das classes são o cálculo da matriz $\left[B\right]$ e do carregamento nodal equivalente, como mostrado nas figuras 3.12 e 3.13.

```
// obtém número de nós utilizados no mapeamento da geometria
nummapnodes = msg( shpelem, NumNodes )
// obtém o número de nós utilizados na interpolaçãode deslocamentos
numshpnodes = msg( shpelem, NumShpNodes )
// obtém as funções de forma no ponto de Gauss corrente
msg( shpelem, ShapeFunc, gaussrst, shapefunc )
// obtém as derivadas das funções de mapeamento em relação às
// coordenadas locais no ponto de Gauss corrente
msg( shpelem, DerivMaprst, gaussrst, derivmaprst )
// obtém as derivadas das funções de forma em relação as coordenadas
// locais no ponto de Gauss corrente
msg( shpelem, DerivShprst, gaussrst, derivshprst )
// obtém a inversa da matriz Jacobiana e seu determinante
msg( shpelem, Jacobian, nummapnodes, derivmaprst, elemcoord, jacinv, detjac )
// obtém as derivadas das funções de forma em relação às coordenadas
// cartesianas
msg( shpelem, Derivxyz, numshpnodes, jacinv, derivshprst, derivxyz )
// Monta a matriz [B]
msg( anmodel, MountBMatrix, numshpnodes, elemcoord, shapefunc, derivxyz, bmatrix )
```

Figura 3.12: Algoritmo de cálculo da matriz $[B]$.

```
// obtém número de graus de liberdade por nó
numdofnode = msg( anmodel, NumDofNode )
// obtém o número de nós do elemento corrente
numnodes = msg( shpelem, NumMapNodes )
// calcula o número de graus de liberdade do elemento de carregamento
ndof = numdofnode * numnodes
// obtém as direções globais dos graus de liberdade
msg( anmodel, DofGlobDir, dirglob )
// obtém as coordenadas dos nós do elemento de carregamento
msg( shpelem, NodalCoord, elemcoord )
// obtém a ordem de integração ótima para o elemento de carregamento
msg( shpelem, OptmRespOrder, order )
// obtém o número de pontos de Gauss
numgausspts = msg( gauss, NumPts, order )
// obtém as coordenadas e os pesos dos pontos de Gauss
msg( gauss, Points, order, gaussrst, gausswgt )

foreach gauss point [i], i = 1..numgausspts
     begin
     // obtém matriz [N] para o ponto de Gauss corrente
     msg( loadelem, NMatrix, gaussrst[i], coord, detjac, nmatrix )
     // avalia a carga o ponto de Gauss corrente
     msg( loadelem, Evaluate, gaussrst[i], loadval )
     // transforma o valor da carga para direção de cada grau de liberdade
     TransfValueDof( loadval, dirglob, dofval )
     // calcula o coeficiente multiplicador para o ponto de Gauss corrente
     coeff = detjac * gaussrst[i]
     // adiciona a contribuição deste ponto de Gauss para o vetor de
     // forças equivalentes nodais
     AddValueEqvForc( numnodes, numdofnodes, nmatrix, dofval, coeff, equivforce )
     end
```

Figura 3.13: Algoritmo de cáculo do carregamento nodal equivalente.

3.4 *Implementação da Estimativa de Erro*

Esta seção mostra a definição do estimador de erro usado e sua implementação dentro da filosofia de programação orientada a objetos. Ressalta-se aqui que o estimador utilizado foi escolhido apenas porque é o mais difundido na área e é de fácil implementação. Acredita-se que a filosofia de programação empregada permita que se troque o estimador com um mínimo de alteração na organização de classes. O estimador de erro é baseado em trabalho desenvolvido por Zienkiewicz e Zhu (1987).

3.4.1 *Formulação do Problema*

A solução de um problema linear elástico com deslocamentos u definidos em um domínio Ω pode ser colocada segundo uma equação diferencial de equilíbrio a um carregamento q como

$$Lu - q \equiv S^T DSu - q \;=\; 0, \tag{3.4}$$

com deslocamentos prescritos em um contorno Γ_u

$$u = u_p, \tag{3.5a}$$

e forças de superfície prescritas em um contorno Γ_t

$$GDSu = t_p, \tag{3.5b}$$

com $\Gamma = \Gamma_u \cup \Gamma_t$.

Nas expressões acima, o operador diferencial matricial S define as deformações em função dos deslocamentos

$$\varepsilon \;=\; Su; \tag{3.6}$$

a matriz D define as tensões em função das deformações

$$\sigma \;=\; D\varepsilon, \tag{3.7}$$

e G é um operador linear que relaciona o campo de tensões e as forças de superfície no contorno do domínio.

Numa aproximação por elementos finitos com

$$u \approx \hat{u} \;=\; N\widetilde{u}, \tag{3.8}$$

onde $\widetilde{u}$ é a solução numérica nodal para o campo de deslocamentos e N são as funções de forma que relacionam deslocamentos nodais com deslocamentos em qualquer ponto do domínio de um elemento, obtém-se equações de aproximação (por um processo padrão de Galerkin ou, equivalentemente, por minimização da energia potencial) para obter

$$K\widetilde{u} - f \;=\; 0, \tag{3.9}$$

onde

$$K = \int_{\Omega} \Big(SN\Big)^{T} D \Big(SN\Big) d\Omega,$$

$$f = \int_{\Omega} N^{T} q d\Omega + \int_{\Gamma_t} N^{T} t_p d\Gamma,$$

e as tensões são calculadas como

$$\hat{\sigma} = \Big(DSN\Big)\widetilde{u}. \tag{3.10}$$

A solução aproximada $\hat{u}$ e $\hat{\sigma}$ difere da solução exata u e σ, e essa diferença é o erro. Assim, tem-se para erro em deslocamentos e em tensões as expressões:

$$e \;=\; u - \hat{u}, \tag{3.11a}$$

$$e_{\sigma} \;=\; \sigma - \hat{\sigma}. \tag{3.11b}$$

3.4.2 *Normas de Energia*

Geralmente é difícil especificar erros segundo as equações (3.11); várias medidas baseadas em integrais são então mais convenientes. Uma das mais comuns é a chamada "norma de energia" escrita para problemas de elasticidade como

$$\|e\| = \left(\int_{\Omega} (Se)^{t} D\,(Se)\;\, d\Omega\right)^{1/2} =$$

$$= \left(\int_{\Omega} (e_{\varepsilon})^{t} D\,(e_{\varepsilon})\;\, d\Omega\right)^{1/2} = \tag{3.12}$$

$$= \left(\int_{\Omega} (e_{\sigma})^{t} D^{-1}\,(e_{\sigma})\;\, d\Omega\right)^{1/2}.$$

Uma medida mais direta é chamada norma L_2, que pode ser associada com erros em qualquer quantificação. Para as tensões, a norma L_2 para o erro é

$$\|e_\sigma\|_{L_2} = \left(\int_\Omega (e_\sigma)^t \, (e_\sigma) \; d\Omega \right)^{1/2}. \tag{3.13}$$

Embora todas essas normas sejam definidas para todo o domínio, nota-se que o quadrado de cada norma pode ser obtido somando as contribuições dos elementos como

$$\|e\|^2 = \sum_{i=1}^{m} \|e\|_i^2, \tag{3.14}$$

onde i representa um elemento e m o número total de elementos do modelo. Para uma malha considerada "ótima", tenta-se fazer as contribuições para esse quadrado da norma igual para todos os elementos.

Entretanto, o valor absoluto da norma de energia ou da norma L_2 tem pouco significado físico. É preferível adotar um erro relativo

$$\eta \;=\; \frac{\|e\|}{\|u\|}, \tag{3.15}$$

que pode ser determinado para todo o domínio ou para subdomínios de elementos, onde $\|u\|$ é o valor positivo da raiz quadrada do dobro da energia de deformação:

$$\|u\| = \left(\int_\Omega \sigma^t D^{-1} \sigma \; d\Omega \right)^{1/2}. \tag{3.16}$$

3.4.3 *Estimador de Erro*

O cálculo da norma de erro subtende uma estimativa para as tensões analíticas. Neste trabalho, essa estimativa é feita pelo projetor de Hinton e Campbell (Hinton, Campbell, 1974), com suavização dos valores com base na média dos valores nodais dos elementos adjacentes. Outra possibilidade seria o processo superconvergente para recuperação das tensões, conforme trabalho recente de Zienkiewicz e Zhu (1992a). Vale lembrar que esta estimativa é apenas um método do módulo de análise e pode ser trocado facilmente, sem influir no estimador usado. A norma de energia do erro estimado em tensões, definida na equação (3.12), resulta, para uma estimativa das tensões analíticas σ^*, em

$$\|\overline{e}\| = \left(\int_\Omega (\sigma^* - \hat{\sigma})^t \, D^{-1} \, (\sigma^* - \hat{\sigma}) \; d\Omega \right)^{1/2}. \tag{3.17}$$

O estimador de erro vai então definir como o modelo deve ser refinado. Esse refinamento vai depender de um critério de acurácia que se deseja satisfazer. Neste trabalho, o critério será a condição que o erro relativo estimado $\overline{\eta}$ do modelo, seja menor que um valor pré-definido η^*:

$$\overline{\eta} \leq \eta^*, \tag{3.18}$$

com $\overline{\eta}$ sendo calculado por

$$\overline{\eta} = \frac{\|\overline{e}\|}{\sqrt{\|\hat{u}\|^2 + \|\overline{e}\|^2}}, \tag{3.19}$$

considerando as aproximações $\|u\| \approx \|\overline{u}\|$ e $\|e\| \approx \|\overline{e}\|$ e a relação

$$\|\overline{u}\|^2 = \|\hat{u}\|^2 + \|\overline{e}\|^2, \tag{3.20}$$

onde $\|\hat{u}\|^2$ o dobro da energia de deformação obtida da solução numérica.

Se for assumido que o erro deve ser igualmente distribuído entre os elementos, a condição (3.18) pode ser estendida para o limite no erro em cada elemento. Então para cada elemento, sendo m o número total de elementos, deve-se ter

$$\|\overline{e}\|_i \;\leq\; \eta^* \left[\; (\|\hat{u}\|^2 + \|\overline{e}\|^2)/m \;\right]^{1/2} = \overline{e}_m, \tag{3.21}$$

Como o erro é calculado para cada elemento, pode-se verificar onde o refinamento se faz necessário. A razão de erro

$$\xi_i = \|\overline{e}\|_i / \overline{e}_m > 1, \tag{3.22}$$

indica os elementos a serem refinados, e seus valores, assumindo um certo grau de convergência, decidem o tamanho do elemento desejado. Essa razão de erro, implementada no módulo de análise numérica, é exportada para o módulo auto-adaptativo, que calcula os novos tamanhos dos elementos para atender o critério de erro. Assumindo que o tamanho do elemento seja h_i, e que a taxa de convergência do erro seja $O(h^p)$ (na área coberta pelo elemento), concluímos que o novo tamanho do elemento deve ser

$$h = h_i / \xi_i^{1/p}. \tag{3.23}$$

Normalmente, assume-se que p seja igual à ordem do polinônimo da formulação do elemento; esta regra simples não é válida perto de singularidades, onde deve-se ter um tratamento especial. Duas observações se fazem importantes. A primeira é que o novo

tamanho do elemento é calculado pelo módulo auto-adaptativo, baseado na razão de erro calculada pelo módulo de análise numérica, através de método da classe *Elemento* do módulo auto-adaptativo, que usa os mesmos conceitos da programação orientada a objetos descritos permitindo o tratamento de vários tipos de elementos, sem limitá-lo a um elemento específico. A segunda é que esse cálculo do tamanho característico do elemento tanto serve para refinar regiões onde se necessita de uma melhor discretização, quanto para "desrefinar" áreas onde uma discretização maior é permitida.

3.4.4 *Implementação do Estimador de Erro*

A implementação do erro neste módulo de análise foi feita incluindo-se somente métodos novos nas classes existentes. Mais uma vez, a adequabilidade das classes criadas pode ser comprovada, através do algoritmo genérico de cálculo do erro mostrado na figura 3.14. O algoritmo é apresentado em pseudo-código de forma simplificada e corresponde ao método usado pelas subclasses da classe *Elemento*.

```
// obtem número de graus de liberdade por nó do objeto Modelo de Análise do elemento
ndofnode = msg( anmodel, NumDofNode )
// obtem o número de nós do elemento corrente do objeto Forma do Elemento do elemento
numnodes = msg( shpelem, NumMapNodes )
// calcula o número de graus de liberdade do elemento
ndof = numdofnode * numnodes
// obtem a ordem de integração do elemento
msg( elem, Order, order )
// obtem o número de pontos de Gauss
numgausspts = msg( gauss, NumPts, order )
// obtem as coordenadas e os pesos do pontos de Gauss
msg( gauss, Points, order, gaussrst, gausswgt )
foreach gauss point [i], i = 1..numgausspts
     begin
     // obtem as funções de forma no ponto de Gauss corrente
     msg( shpelem, ShapeFunc, gaussrst, shapefunc )
     // obtem uma estimativa para as tensões analíticas no ponto de Gauss
     msg( anmodel, AnalitStr, numnodes, shapefunc, nodestress, galitstress )
     // obtem as derivadas das funções de mapeamento em relação às coordenadas
     // locais no ponto de Gauss corrente
     msg( shpelem, DerivMaprst, gaussrst, derivmaprst )
     // obtem a inversa da matriz Jacobiana e seu determinante
     msg( elem, Jacobian, nummapnodes, derivmaprst, elemcoord, jacinv, detjac )
     // obtem a espessura do elemento
     msg( elem, Thickness, thickness )
     // obtem o coeficiente de rigidez com base na espessura e o ajusta
     // de acordo com o tipo de análise
     msg( anmodel, RigidCoeff, numnodes, elemcoord, mapfunc, thickness, rigidcoeff )
     // calcula o coeficiente multiplicador do triplo produto
     coeff = rigidcoeff * detjac * gausswgt[i]
     // adiciona a contribuição do erro para o ponto de Gauss no erro do elemento
     msg( anmodel, ElementError, gastress, coeff, galitstress, elemerr )
     end
// calcula o erro final no elemento
elemerr = sqrt( elemerr )
```

Figura 3.14: Algoritmo de cálculo da norma de energia do erro no elemento.

4

Auto-refinamento das Fronteiras

Dentro da estratégia de auto-adaptação deste trabalho, descrita no capítulo 2, este capítulo descreve a estratégia de refinamento das curvas nas fronteiras das regiões do domínio bem como a redistribuição dos atributos nestas fronteiras após o refinamento feito.

O refinamento das fronteiras das regiões é feito independente da discretização do domínio do modelo, sendo suportado por uma organização de classes existente no módulo auto-adaptativo, implementado dentro de uma filosofia de programação orientada a objetos.

4.1 *Refinamento das Fronteiras das Regiões*

A estratégia utilizada para refinar as fronteiras das regiões é uma versão unidimensional (em cada curva) do algoritmo utilizado para refinar o domínio, baseado na técnica de *quadtree*. O refinamento de cada curva na fronteira utiliza uma técnica de enumeração espacial recursiva baseada em uma estrutura de dados de árvore binária (*binary tree*). A idéia é gerar uma discretização regular nas curvas da fronteira em função do tamanho característico dos elementos da malha adjacentes a essas curvas. Esses tamanhos característicos são calculados baseados na estimativa de erro de cada elemento. O refinamento é dividido em etapas, conforme descrito nas sub-seções a seguir.

4.1.1 *Inicialização da Árvore Binária*

A primeira fase do refinamento consiste em inicializar a árvore binária que irá orientar o refinamento da curva. Essa árvore é inicializada com as coordenadas da célula pai da árvore parametrizada ao longo da curva entre zero (coordenada mínima) e um (coordenada máxima). Isto é feito para permitir um procedimento de refinamento genérico para qualquer tipo de curva existente no modelo, seja ela linha reta, linha poligonal, arco de círculo, curva de Bézier, etc. A profundidade da árvore neste ponto é inicializada como zero, já que se trata inicialmente de um único nível.

4.1.2 *Refinamento da Árvore Binária*

Inicializada a árvore, parte-se para o seu refinamento. Primeiro, acha-se que nós da malha de elementos finitos pertencem à curva. Isto é feito através de algoritmos de geometria computacional baseados em tolerância geométrica. Esses nós são ordenados de acordo com sua parametrização em relação à curva, considerando-se a orientação de seus nós extremos, e incluídos em uma lista de nós que a curva contém.

De posse dos nós que pertencem à curva, acha-se que elementos finitos são adjacentes a ela. Este procedimento é facilitado pelo conhecimento de quais elementos são adjacentes a um nó, informação que também é útil para diversos outros fins, como obtenção dos contornos de tensões. Assim, esta informação de adjacência é disponível na estrutura de dados dos nós da malha. Insere-se então esses elementos em uma lista de elementos adjacentes à curva.

Percorre-se então todos os elementos finitos desta lista, refinando a árvore segundo o procedimento a ser descrito. Primeiro, calcula-se o tamanho característico do elemento. Em seguida, divide-se o tamanho anterior por este novo tamanho, obtendo-se um número de segmentos contidos no lado do elemento que é adjacente à curva. Estes segmentos serão utilizados no refinamento da árvore. A figura 4.1 mostra a curva superior da fronteira do modelo mostrado na figura 2.2 com os seus elementos adjacentes da malha inicial. Os segmentos gerados para cada elemento também estão mostrados na figura. Para cada segmento, calcula-se o seu ponto médio e parametriza-o em relação à curva em questão. Encontra-se em que célula da árvore este ponto médio está localizado (a célula pai é a própria curva parametrizada de zero a um). Se o tamanho da célula é maior que o tamanho característico do elemento em questão, subdivide-se a célula em duas, incrementando-se um nível de profundidade na árvore, e assim sucessivamente até ser menor. Passa-se para o próximo segmento e repete-se o processo.

O controle do refinamento da árvore binária para cada elemento adjacente à curva poderia ter sido feito apenas com base no ponto médio do lado do elemento na curva. Entretanto, optou-se por trabalhar com segmentos menores que o lado, que já carregam uma informação sobre o refinamento necessário, pois este procedimento resultou em um refinamento mais regular.

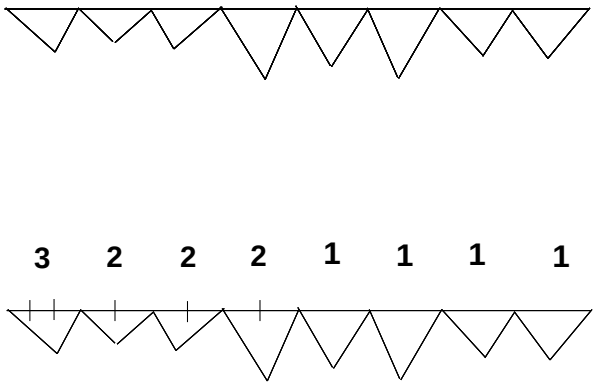


Figura 4.1: Curva com os elementos adjacentes e o número de segmentos neles gerados.

Após percorrer todos os segmentos de todos os elementos adjacentes à curva, tem-se a árvore refinada segundo o erro dos elementos adjacentes à curva. A figura 4.2 mostra a nova discretização gerada na curva e os níveis da árvore correspondentes essa nova discretização. Cada célula folha da árvore (célula que não possui filhos) vai gerar um lado de elemento na nova discretização da curva. Nota-se que a curva foi mais refinada na extremidade esquerda, onde o erro numérico é maior, enquanto foi possível aumentar o tamanho do lado do elemento na extremidade direita, onde o erro é menor.

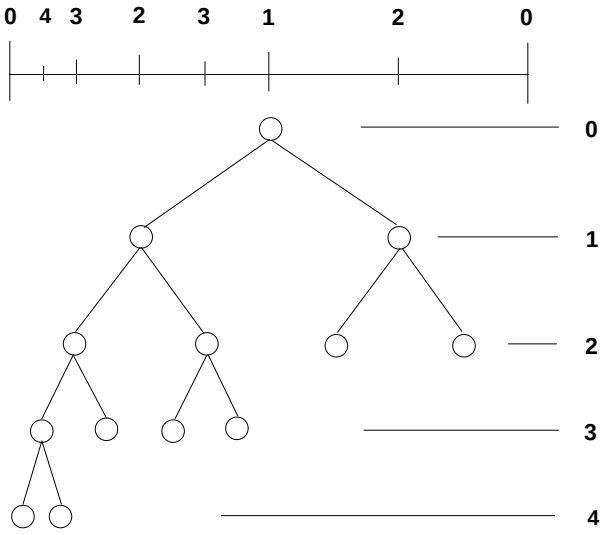


Figura 4.2: Curva discretizada e sua árvore binária.

No algoritmo descrito acima, o que é considerado para determinação dos segmentos é o lado do elemento finito, independentemente do tipo de elemento, ou seja, para elementos quadráticos é considerado o lado sem levar em conta os nós no meio do lado.

4.1.3 *Atualização da Discretização da Curva*

Após a árvore estar definida para a curva considerada, inclui-se a nova discretização na lista de nós da curva. A nova discretização é a definida pelas coordenadas extremas das células folhas da árvore. Para evitar que a cada coordenada (máxima ou mínima) de uma célula folha se faça uma busca na lista de nós da curva para verificar se aquela coordenada já foi inserida, faz-se o seguinte procedimento: obtém-se a coordenada paramétrica mínima (que é zero) da célula pai da árvore, acha-se a coordenada cartesiana correspondente em relação à curva considerada, e inclui-se essa coordenada na lista de nós da curva; percorre-se então toda a árvore, transformando-se para coordenadas cartesianas todas as coordenadas máximas das células folhas e incluindo-as na lista de nós da curva, tendo-se ao final, de forma ordenada, toda a nova discretização da curva.

O motivo de se considerar somente a coordenada máxima de uma célula folha se deve ao fato de que sempre uma coordenada mínima de uma célula folha é igual a uma coordenada máxima da célula folha vizinha. Deste modo evita-se repetições de pontos na discretização da curva. Isto também justifica a inclusão da coordenada mínima da célula pai da árvore no primeiro passo; caso contrário o primeiro ponto da curva (correspondente à coordenada paramétrica zero) não seria incluido na discretização. A figura 4.3 mostra o resultado final do refinamento nas curvas da fronteira para o exemplo da figura 2.2.

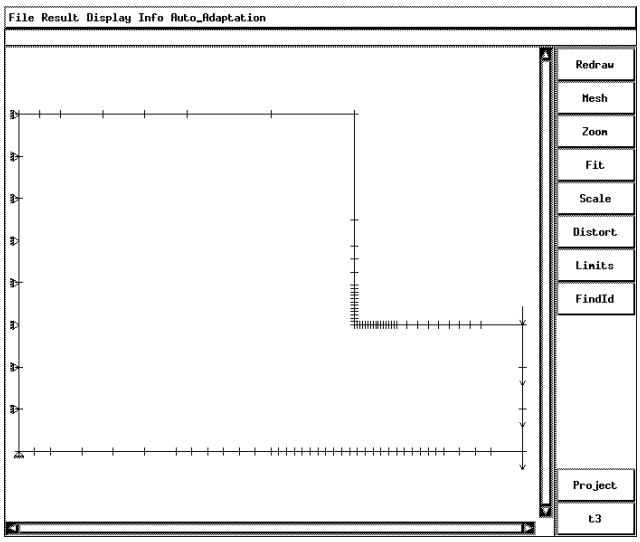


Figura 4.3: Refinamento nas curvas da fronteira do modelo efetuado pela técnica da árvore binária.

4.2 *Redistribuição dos Atributos nas Fronteiras das Regiões*

Na redistribuição dos atributos das curvas com relação à sua nova discretização, três casos devem ser tratados: atributos nodais pontuais, atributos aplicados sobre a curva que vão para os nós, e atributos aplicados sobre a curva que vão para os lados dos elementos adjacentes às curvas.

4.2.1 *Atributos Nodais Pontuais*

Existem alguns atributos que são aplicados diretamente sobre alguns nós da malha de elementos finitos. Este é o caso de suporte, deslocamentos prescritos, forças nodais, temperaturas prescritas, etc. Esse nós correspondem aos pontos extremos das curvas do modelo e de bordo, isto é, estes nós são sempre mantidos na estratégia de refinamento pela árvore binária. Entretanto, a lista de nós que pertecem a uma curva é alterada a cada etapa. Portanto deve-se atualizar os atributos nodais dos nós extremos nesta lista segundo os atributos nodais pontuais.

No arquivo de descrição de atributos das curvas, os atributos nodais pontuais são definidos através de suas coordenadas e dos valores do atributo propriamente dito. Essa descrição só é lida na primeira etapa do processo e é armazenada na lista de atributos das curvas. Essa lista de atributos é sempre consultada, a cada etapa, para a atualização dos atributos nodais pontuais.

4.2.2 *Atributos Nodais Advindos de Curvas*

Na descrição de atributos das curvas é possível definir atributos nodais para uma curva inteira. Neste caso, o módulo auto-adaptativo se encarrega de transferir esse atributos para os nós da curva a qual eles pertecem. Por exemplo, para impedir a translação na direção x de todos os nós de uma determinada curva, basta definir que a curva terá a translação nesta direção impedida; o módulo auto-adaptativo automaticamente transfere esta restrição para os nós da malha que são gerados nesta curva.

O tratamento deste caso se torna simples, pois para cada curva são armazenados os atributos que devem ser levados aos nós a cada nova malha gerada; deve-se, então, percorrer a nova lista de nós criada e levar os atributos da curva para os nós a cada etapa. Obviamente, se um nó extremo da curva tem um atributo distinto do atributo da curva, o atributo nodal pontual prevalece.

4.2.3 *Atributos em Lados de Elementos Advindos de Curvas*

O módulo de análise numérica tem a capacidade de calcular carregamento nodais equivalentes para possibilitar a análise. Por isso, alguns atributos, como forças e momentos, fluxo de calor, convecção, podem ser definidos para o modelo de análise através do lado do elemento finito que está recebendo o atributo.

O tratamento destes casos é feito de uma maneira semelhante à dos atributos nodais advindos das curvas. Uma vez definido o atributo para a curva, percorre-se a nova lista de elementos adjacentes existente na estrutura de dados da curva, e aplica-se esses atributos da curva para o lado do elemento finito correspondente. Esse procedimento só é feito após uma nova malha gerada.

4.3 *Organização de Classes Para Tratamento das Fronteiras*

O tratamento do refinamento e da redistribuição dos atributos nas fronteiras pede uma organização do módulo auto-adaptativo que propicie um procedimento genérico para quaisquer tipos de curvas e elementos. Desta forma, novos tipos de curvas podem ser introduzidas com facilidade, assim como novos tipos de elementos finitos. A filosofia de programação orientada a objetos é ideal para este tratamento genérico e acarreta uma implementação natural da estratégia. Isto é descrito a seguir.

4.3.1 *Seleção das Classes*

O processo de seleção das classes, no escopo de programação orientada a objetos, baseou-se em dois aspectos importantes dentro do processo de tratamento das fronteiras. O primeiro aspecto é o cálculo do tamanho característico do elemento conforme definido pela expressão (3.23). O tamanho característico do elemento é usado no refinamento das curvas e depende da razão de erro para o elemento que é exportada pelo módulo de análise numérica, do tamanho anterior do elemento, e do grau do polinômio da formulação do elemento. Esse aspecto motivou o aparecimento de uma outra classe *Elemento*, agora no módulo auto-adaptativo.

O segundo aspecto é o uso da árvore binária parametrizada para permitir o tratamento do refinamento para curvas genéricas (linha reta, linha poligonal, arco de círculo, curva de Bézier, etc.), além da necessidade do armazenamento dos atributos específicos para cada tipo de curva de acordo com a descrição dos atributos lidos pelo sistema. Isto motivou o aparecimento da classe *Curva*.

4.3.1.1 *Classe Elemento*

A classe *Elemento* foi criada principalmente com a responsabilidade de fornecer todas as informações necessárias sobre o tamanho característico dos elementos para orientar o refinamento das curvas de fronteira e de malha no interior do domínio. Além disso, como o módulo auto-adaptativo permite a visualização de respostas (tensões, deslocamentos, etc.) a cada etapa do processo, algumas informações relativas aos elementos são importantes para os algoritmos que implementam as técnicas de visualização empregadas e devem ser fornecidas por essa classe.

A abrangência atual desta classe é relativa a todos os elementos aos quais a formulação de erro se aplica, bem como a elementos bidimensionais para os quais é possível visualizar os resultados de uma análise por elementos finitos. A figura 4.4 mostra as subclasses da classe *Elemento*.

T3	triângulo com três nós
T6	triângulo com seis nós
Q4	quadrilátero com quatro nós
Q8	quadrilátero com oito nós
QUAD	quadrilátero Serendipity geral
INTERFACE	elemento de interface
INFINITE	elementos infinitos

Figura 4.4: Subclasses da classe Elemento.

4.3.1.2 *Classe Curva*

A classe *Curva* fornece as informações necessárias ao refinamanto das curvas das fronteiras e da redistribuição dos seus atributos. Essas informações são disponíveis para todos os tipos de curvas suportados pelo módulo auto-adaptativo, fazendo com que o refinamento e a redistribuição dos atributos seja genérico para todas elas. A figura 4.5 mostra a organização atual das subclasses desta classe *Curva*.

LINE	linha reta
POLY	linha poligonal
CIRCLE	círculo completo
ARC	arco de círculo
BEZIER	curva de Bézier

Figura 4.5: Subclasses da classe Curva.

4.3.2 *Especificação das Classes*

A especificação das classes criadas é descrita aqui pelos seus protocolos (lista mensagens) e pela descrição dos métodos invocados por essas mensagens.

4.3.2.1 *Classe Elemento*

Esta classe especifica os métodos relativos ao cálculo do tamanho característico dos elementos e dos métodos usados na vizualização dos resultados. A figura 4.6 mostra o protocolo da classe com a respectiva descrição dos métodos invocados. No método do cálculo do tamanho característico uma observação é importante: o cálculo do novo tamanho do elemento necessita do tamanho anterior; este tamanho é definido como o tamanho da aresta de um triângulo equilátero que têm a mesma área do triângulo em questão para elementos triangulares e o tamanho da aresta do quadrado que têm a mesma área do quadrilátero para elementos quadrilaterais.

Métodos para a classe:

Init	Inicializa a classe.
New	Cria uma nova instância (objeto) da classe.

Métodos para objetos:

NumNodes	Retorna o número de nós do elemento.
StressOrder	Retorna a ordem de integração para tensões.
Thickness	Retorna a espessura do elemento.
Read	Lê e armazena os dados relativos ao elemento e armazena-os.
Write	Escreve os dados do elemento.
Connect	Retorna a conectividade do elemento.
NodesCoords	Retorna as coordenadas dos nós do elemento.
DefNodesCoords	Retorna as coordenadas dos nós do elemento para o modelo deformado.
GetGp	Retorna o ponto de Gauss mais próximo de uma coordenada dada.
GetGpCoord	Retorna a coordenada do ponto de Gauss em um elemento.
DrawGp	Desenha o ponto de Gauss.
IntPol	Retorna o polígono interno do elemento para representação de isofaixas de resultados para uma malha definida pela distribuição dos pontos de gauss nos elementos (malha dual).
EdgePol	Retorna o polígono referente cada lado do elemento para representação de isofaixas de resultados para a malha dual.
VertexPol	Retorna o polígono referente aos nós de canto de elemento para representação de isofaixas de resultados para a malha dual.
NodalPol	Retorna o polígono referente aos nós do elemento para representação de isofaixas de resultados sem suavização na malha.
SmoothNodalPol	Retorna o polígono referente aos nós do elemento para representação de isofaixas de resultados com suavização na malha.
Draw	Desenha o elemento representado por uma união de linhas.
DrawFill	Desenha o elemento representado por uma área.
IncFill	Preenche a incidência do elemento.
CcwFace	Retorna as coordenadas dos nós do elemento em sentido anti-horário.
StressBar	Retorna as tensões principais nos pontos de Gauss do elemento.
SizeError	Calcula o novo tamanho do elemento baseado no seu erro.
EdgeError	Retorna os segmentos do lado do elemento que irão influenciar no refinamento das curvas das fronteiras.
FillAdjEdgeGp	Preenche o polígono referente aos cada lado do elemento para o campo de resposta desejado.
FillAdjVertGp	Preenche o polígono referente aos nós de canto de elemento para o campo de resposta desejado.

Figura 4.6: Protocolo e descrição dos métodos da classe Elemento.

4.3.2.2 *Classe Curva*

No processo de refinamento das curvas, o uso uma árvore binária paramétrica para tratar curvas genéricas implica na existência de métodos que relacionem coordenadas cartesianas da curva com coordenadas paramétricas dessa curva (parametrizada de zero a um). Por exemplo, existe a necessidade de um método que dado um ponto x, y da curva, devolva sua coordenada paramétrica nesta curva; um método para o contrário, ou seja, dada uma coordenada paramétrica, achar o ponto x,y correspondente; um método para dado dois pontos paramétricos da curva, achar o comprimento entre eles (usado para achar o comprimento de uma célula na árvore binária). Além disso, no processo de redistribuição de atributos, alguns métodos, tais como os que lêem os atributos para cada tipo de curva, também são necessários. A figura 4.7 mostra o protocolo e a descricão dos métodos da classe *Curva*.

Métodos para a classe:	
Init	Inicializa a classe.
New	Cria uma nova instância (objeto) para a classe.
Métodos para objetos:	
Read	Preenche a lista de dados da curva de um arquivo lido.
ReadAtt	Preenche a lista de atributos da curva de um arquivo lido.
XYtoT	Retorna a coordenada paramétrica para uma coordenada cartesiana da curva dada.
TtoXY	Retorna a coordenada cartesiana para uma cooordenada paramétrica da curva dada.
Tlen	Retorna o comprimento de um segmento da curva localizado entre duas coordenadas paramétricas dadas.
AdjVer	Preenche a lista de nós da curva com os nós da malha de elementos finitos à ela adjacentes.
PolVer	Retorna uma poligonal equivalente à curva, a ser usada nos algoritmos de geometria computacional.
NumVer	Retorna o número de nós da malha existentes na lista de nós da curva.
PutVer	Põe um nó de elemento finito na lista de nós da curva.
XYVer	Retorna coordenada cartesiana de um nó da curva segundo sua posição na lista de nós da curva.
FreeVer	Libera a lista de nós da curva.
AdjElm	Preenche a lista de elementos da curva com os elementos da malha à ela adjacentes.
NumElm	Retorna o número de elementos da malha existentes na lista de elementos da curva.
TypeElem	Retorna o tipo do elemento, se linear, etc.
SizeElm	Retorna o tamanho característico de um elemento na curva definido pela sua posição na lista de elementos desta curva.
MidElm	Retorna as coordenada paramétricas de um conjunto de segmentos da curva definidos pela relação dos tamanhos novo e antigo do elemento fornecido.
FreeElm	Libera a lista de elementos da curva.
Free	Libera uma instância (objeto) da classe.

Figura 4.7: Protocolo e descrição dos métodos da classe Curva.

A estrutura de um objeto da classe *Curva* tem em seu interior uma lista de dados da curva, uma lista de atributos, uma lista de nós da malha pertencentes à curva e uma

lista de elementos da malha adjacentes à curva. Todos os métodos que gerenciam a lista de elementos e nós da curva são implementados na superclasse *Curva* e herdados por suas subclasses. Os métodos que redistribuem os atributos também são implementados na superclasse, já que são iguais para todas as curvas, independentemente do seu tipo. Por outro lado, os métodos que tratam da lista de dados da curva, onde se incluem os dados geométricos e de atributos, e das parametrizações que são necessárias ao refinamento, são definidos diretamente nas subclasses, pois dependem de cada tipo de curva. Esta organização facilita a expansão para a inclusão de novas curvas sem alterar a estratégia de refinamento ou redistribuição dos atributos.

5

Combinação das Técnicas de Quadtree e Delaunay para Geração da Malha de Elementos Finitos

Este capítulo descreve o método para geração de malhas dentro do processo auto-adaptativo. O algoritmo procura aliar as vantagens das técnicas de *quadtree* e da triangulação de Delaunay por contração do contorno partindo de uma discretização de bordo fornecida. A idéia básica é gerar os elementos no interior da região pela técnica de *quadtree* (Baehmann et al., 1987), deixando uma faixa próxima à fronteira da região para ser gerada pela técnica da triangulação por contração do contorno (Shaw, Pitchen, 1978). Esta última técnica ainda é modificada para usar informações da estrutura *quadtree* já construída de modo a evitar, na formação de um triângulo, que um vértice longe da aresta base seja verificado, desta forma acelerando o método.

Para a geração de malha no domínio, três observações são importantes. A primeira é que, apesar de usar técnicas de triangulação, esse método também pode gerar malhas com elementos quadrilaterais. Embora isto não seja feito neste trabalho, é indicado como pode ser feito através da junção de dois triângulos escolhidos para formar um quadrilátero. A segunda é que o algoritmo proposto pode ser usado para geração automática de malhas em domínios arbitrários (Cavalcante et. al, 1993a), partindo-se de uma discretização do contorno fornecida, sem necessariamente ser uma etapa de um processo auto-adaptativo, bastando para isso não se considerar os procedimentos descritos na sub-seção de refinamento da árvore devido ao erro numérico calculado nos elementos. Finalmente, o algoritmo também é aplicado para elementos quadráticos, bastando para isso, ao final, criar nós de meio de lado.

5.1 *Geração da Malha no Interior do Domínio Através da Quadtree*

Esta seção define como a malha é criada no interior do domínio através do uso da *quadtree*, deixando uma região entre a malha interna assim gerada e o contorno do domínio para ser gerada usando uma técnica de triangulação de Delaunay. O uso da *quadtree* conduz a que a densidade interior das células da árvore seja sensível à discretização fornecida do contorno e também garante uma boa transição entre regiões com diferentes graus de refinamento da malha a ser gerada.

A geração da malha no interior do modelo pode ser dividida em algumas fases. Estas fases são descritas, nas sub-seções a seguir, para o mesmo exemplo utilizado nos capítulos anteriores. A figura 5.1 mostra a discretização de bordo que serve como entrada para o processo do exemplo analisado.

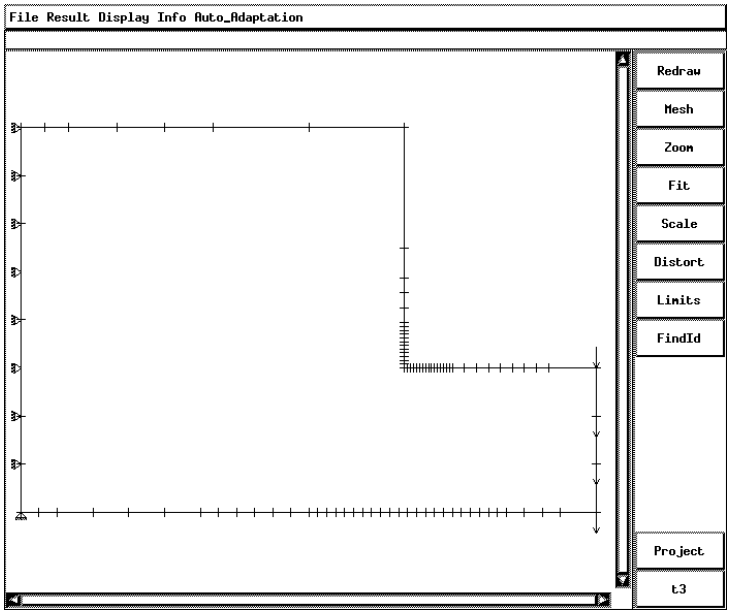


Figura 5.1: Exemplo e sua discretização.

5.1.1 *Criação da Árvore Inicial*

O processo é iniciado pela criação da *quadtree* baseada na discretização do contorno (externo e furos) que é fornecida. Primeiro, as coordenadas máximas e mínimas do contorno são usadas para gerar uma caixa envolvente do modelo, que será o nó pai da *quadtree*. Percorre-se, então, todos os segmentos da discretização do contorno, um por vez, para criar a árvore usando-se o seguinte procedimento: o comprimento e o ponto médio do segmento corrente é calculado, e uma busca é feita no estado atual da árvore para saber em que célula este ponto está; verifica-se então se o tamanho desta célula é maior que uma percentagem do comprimento do segmento e neste caso subdivide-se esta célula, continuando o processo até o tamanho da célula ser menor que esta percentagem. Passa-se então a um novo segmento e repete-se o procedimento. Com relação a esta percentagem, é recomendado um fator entre 0.7 e 1.4 (Potyondy, 1993); neste algoritmo, usou-se 1.0 (100%), pois testes com vários tipos de contorno foram feitos e de um modo

geral o uso deste valor foi o que deu melhores resultados. A figura 5.2 mostra a árvore gerada pelo processo acima descrito.

Com relação às células da árvore, vale dizer que elas podem ser de quatro tipos: *células exteriores*, que são células completamente fora do domínio; *células vértices*, que são células que contém no seu interior algum vértice do contorno do domínio; *células do contorno*, que são células atravessadas por algum segmento desse contorno; e *células interiores*, que estão completamente dentro do domínio. Essa classificação será importante nas próximas fases.

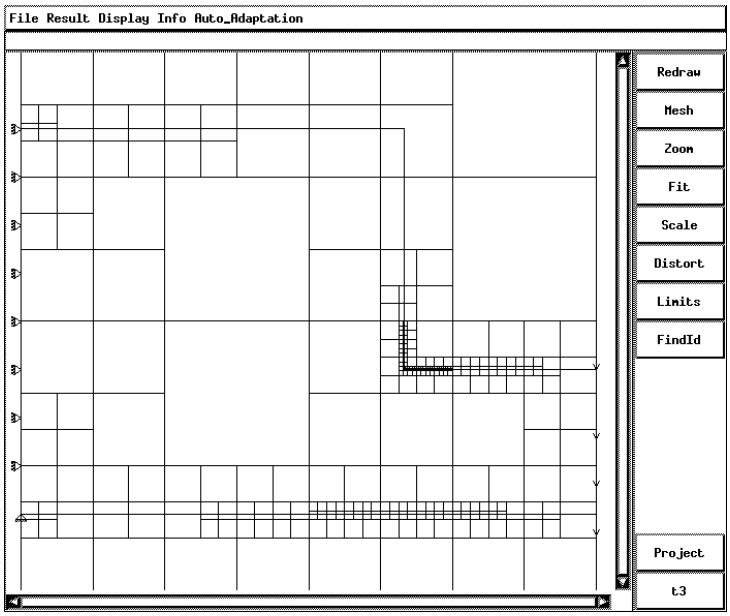


Figura 5.2: Criação da árvore inicial.

5.1.2 *Ajustes Devido ao Erro Numérico dos Elementos*

Criada a árvore inicial, a compatibilidade com o erro numérico de cada elemento é imposta no interior do modelo. Isto é feito do seguinte modo: visita-se cada elemento da malha original fornecida e calcula-se o seu ponto médio; encontra-se a célula da *quadtree* que contém este ponto médio; se o tamanho desta célula é maior que o tamanho característico do elemento finito ditado pela análise de erro, subdivide-se a célula e assim sucessivamente até seu tamanho ser menor que o tamanho característico do elemento; pega-se o próximo ponto médio de elemento finito e faz-se o mesmo, até ter ajustado a árvore para todos os elementos. A figura 5.3 mostra a árvore após feito esses ajustes devido ao erro dos elementos.

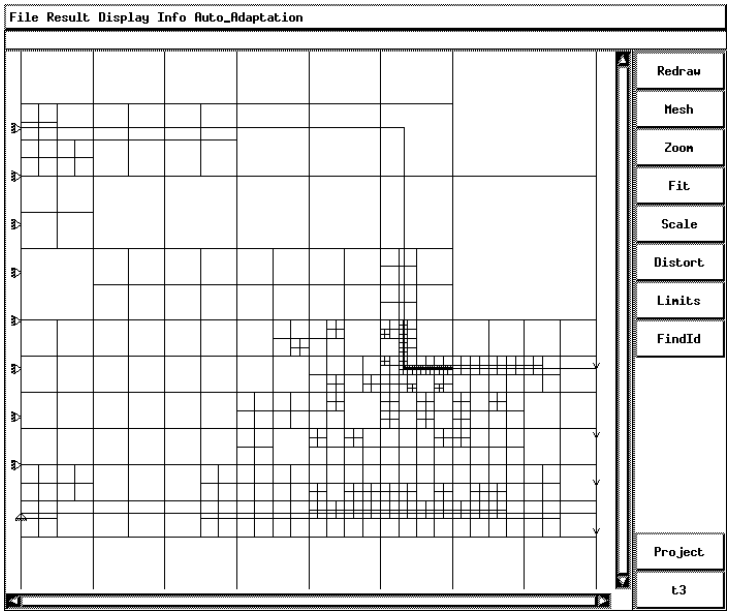


Figura 5.3: Árvore após ajustes devidos ao erro nos elementos.

5.1.3 *Ajustes para um Nível de Diferença entre Células Adjacentes*

Após as fases anteriormente descritas, ajustes devem ser feitos para garantir somente um nível de diferença de profundidade na árvore entre células adjacentes. Este tipo de *quadtree* onde as células adjacentes são do mesmo tamanho ou no máximo com um nível de diferença de profundidade entre elas é chamada de *quadtree restrita* (Von Herzen, Barr, 1987). Isto é necessário pois, posteriormente, serão criados elementos finitos em células interiores da *quadtree* através de padrões e isto exige que na árvore exista somente um nível de diferença entre células adjacentes. Estes padrões são usados para gerar elementos finitos dada a configuração de uma célula e de suas células adjacentes. Isto será abordado com mais detalhes na seqüência. Além disto, um único nível de diferença assegura uma boa qualidade de transição entre os elementos na malha a ser gerada. A figura 5.4 mostra a árvore depois de assegurado um único nível de diferença entre células adjacentes.

5.1.4 *Eliminação de Células Perto do Contorno*

Após estas fases de ajustes, todas as células da árvore, que forem interiores ao domínio irão formar elementos finitos através de padrões e os outros serão gerado através de

triangulação de Delaunay por contração de contorno. Precisa-se então classificar bem o que são células interiores, pois células que estiverem muito perto do contorno poderão ocasionar elementos finitos de forma ruim quando da triangulação da faixa do contorno. A solução é reclassificar células interiores que estejam a uma distância de um segmento do contorno menor que uma percentagem do seu comprimento. No caso desse algoritmo, adotou-se um fator de 0.2 (20%).

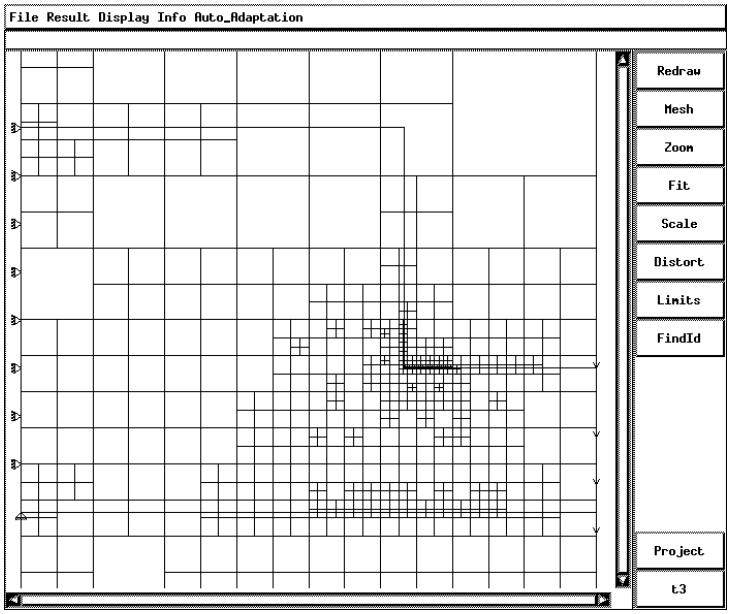


Figura 5.4: Árvore após fase de um único nível de diferença.

5.1.5 *Geração de Malha em Células Interiores por Padrões*

Após estes ajustes feitos, a árvore está pronta para concluir a primeira parte do algoritmo, que é a geração de malha no interior do domínio, criando os elementos finitos nas células interiores através de padrões. Como já foi dito anteriormente, estes padrões são usados para gerar elementos finitos dada a configuração de uma célula e sua adjacência. Existem alguns tipos de padrões para elementos triangulares e outros para elementos preferencialmente quadrilaterais, como se pode ver na figura 5.5 (Baehmann et al., 1987). Nota-se daí a importância de garantir apenas um único nível de diferença entre células adjacentes na árvore, como foi descrito em fase anterior.

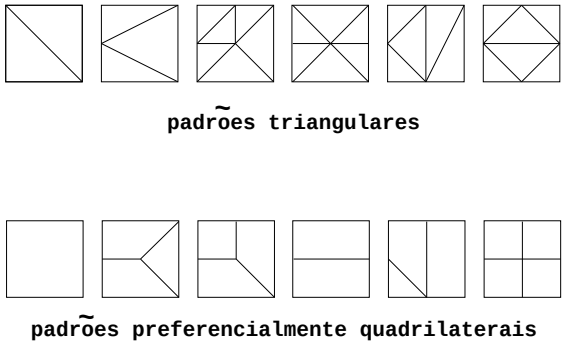


Figura 5.5: Padrões para elementos triangulares e quadrilaterais.

Com o uso destes padrões, gera-se então a primeira parte da malha, que normalmente ocupa a maior área da domínio. Vê-se na figura 5.6 a malha gerada pelo uso dos padrões em células interiores e nota-se que algumas células supostamente interiores não geraram elementos nesta fase, pois foram reclassificadas na fase de eliminação de células perto do contorno como células de contorno e não como interiores.

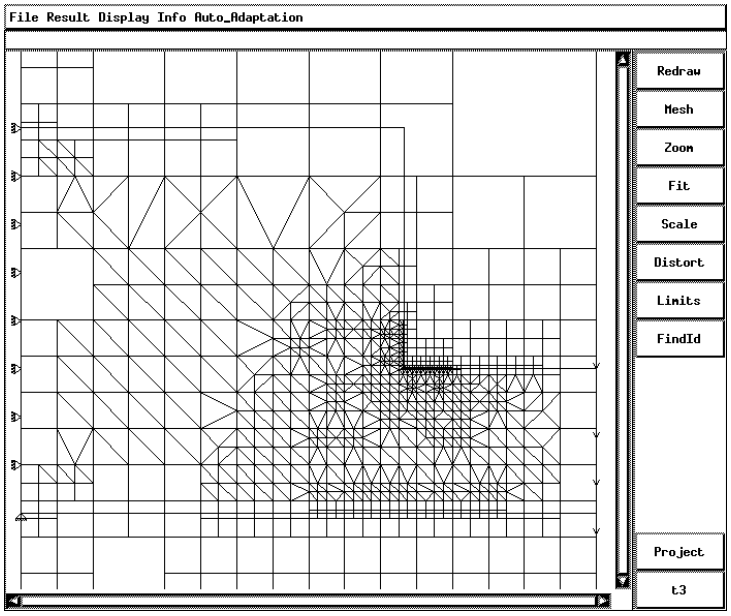


Figura 5.6: Geração de malha no interior do modelo.

5.2 *Geração de Malha no Contorno*

Esta seção descreve como a malha é criada na região entre a *quadtree* interna e o contorno. Isto é feito através da técnica de triangulação de Delaunay por uma contração de contorno (Shaw, Pitchen, 1978; Vianna, 1992; Potyondy, 1993). Isto completa o método de geração de malhas de elementos finitos proposto. Também aqui pode-se dividir em algumas fases, mostradas nas sub-seções a seguir.

5.2.1 *Inicialização da Lista de Arestas Ativas do Contorno*

O processo se inicia com a formação de uma lista de arestas ativas do contorno, originalmente contendo todos os segmentos do contorno que são fornecidos. Escolhe-se uma aresta desta lista, que será a aresta base para formação de um novo triângulo.

5.2.2 *Escolha de Vértice Interior para Formação de um Elemento Finito*

De posse da aresta base, deve-se achar um vértice do interior ou do contorno corrente que forme com a aresta o melhor triângulo. Seguindo o critério de Delaunay, o melhor triângulo é aquele que tem o maior ângulo formado pelo vértice candidato com os vértices da aresta base, como se pode ver pela figura 5.7.

A escolha do melhor triângulo em um algoritmo de contração do contorno genérico é feita testando a aresta base contra todos os outros vértices, o que torna esse algoritmo de complexidade quadrática. No algoritmo proposto, a seleção dos possíveis candidatos a vértice do triângulo é feita da seguinte maneira: a cada aresta base é formada uma lista de células da árvore adjacentes às células que contêm o vértice inicial e final da aresta base; essas células são armazenadas se forem células interiores que portanto possuem vértices interiores possíveis candidatos, ou se forem células vértices, que são as células que possuem vértices do contorno original pois também podem vir a formar um melhor triângulo. Desta forma, o teste do maior ângulo é feito para uma lista reduzida de vértices, evitando testes exaustivos contra vértices que estejam bem longes da aresta base. Além disso, a seleção dos vértices candidatos à formação do melhor triângulo é feita de forma bastante eficiente, pois explora a busca em uma árvore quaternária, já construída anteriormente. É possível que, na criação da lista de células adjacentes, não exista nenhuma célula da árvore que seja interior ou vértice (somente células do contorno), ou seja, não se ache nenhum vértice possível candidato. Estes

casos são raros e só acontecem quando o contorno original tem uma discretização irregular e mal comportada para uma malha de elementos finitos. Nestes casos, extende-se a pesquisa na direção de cada adjacência (que são oito para cada célula: pelos lados e cantos) até que, em cada direção, se tenha uma célula com um vértice possível candidato. Vale ressaltar que cada vértice testado só poderá ser escolhido caso esteja obviamente dentro do domínio a ser gerada a malha e também se a aresta do elemento que ele irá formar não interceptar nenhuma aresta já existente; testes geométricos são feitos então usando ferramentas de geometria computacional para assegurar isto.

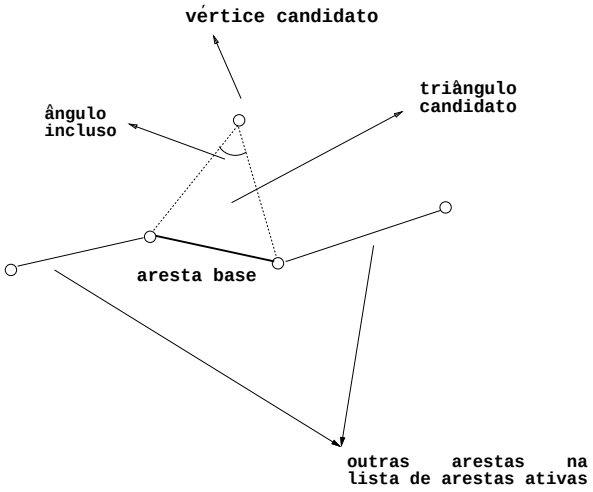


Figura 5.7: Definição do melhor triângulo pela técnica de Delaunay.

Caso se deseje gerar uma malha com elementos quadrilaterais ao invés de triangulares, após a aresta base identificar o melhor triângulo, deve-se selecionar um outro triângulo usando as duas arestas do melhor triângulo corrente formado como arestas base. Os dois triângulos combinados devem formar um quadrilátero aceitável dentro de certos critérios. Isto é mostrado na figura 5.8, onde três possibilidades existem: nenhum triângulo adjacente é selecionado (então só o melhor triângulo é atualizado na estrutura de elementos finitos), um único triângulo adjacente é selecionado, ou dois triângulos adjacentes são selecionados.

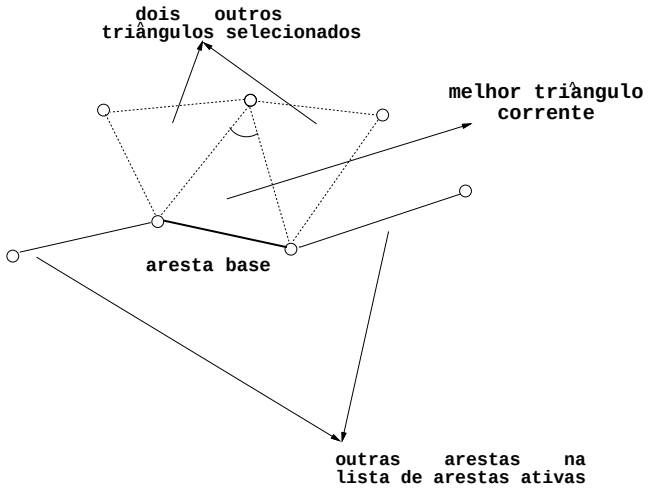


Figura 5.8: Identificação dos dois melhores triângulos adjacentes.

Se um ou dois triângulos adjacentes forem selecionados, então tem-se um ou dois quadriláteros possíveis que devem ter sua forma estudada para a sua aceitação ou não. Para isso, uma medida numérica de forma S é definida na figura 5.9 (Potyondy, 1993), que é uma medida do desvio de qualquer ângulo interno de 90 graus. O elemento é aceitável se $S \leq 1.5$, que corresponde a ângulos internos entre 45 e 135 graus.

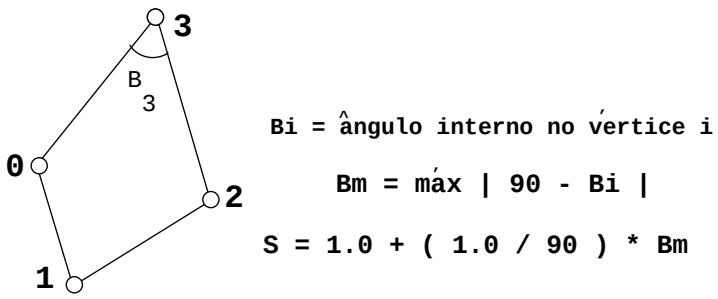


Figura 5.9: Definição da medida de forma quadrilateral S.

5.2.3 *Atualização da Estrutura de Dados*

Quando se chega a essa fase, um novo elemento finito triangular ou quadrilateral foi formado. Esse elemento é adicionado à lista de elementos finitos já existentes e é feita uma atualização da lista de arestas ativas para fazer a contração do contorno propriamente dita. A técnica de contração de contorno tradicional é tal que, formado um elemento finito, a aresta base é retirada da lista de arestas ativas e cada uma das arestas formadas (duas na triangulação ou três na quadrilateração) é inserida na lista, se for uma nova aresta, ou retirada dela, se já existir. Isto faz com que o contorno se contraia até completar todo o domínio, isto é, até gerar a malha em toda a região.

No presente algoritmo, o processo de retirar a aresta base e inserir ou não as novas é o mesmo da contração de contorno tradicional, mas com duas modificações. Primeiro, a contração não cobre todo o domínio pois já existem elementos finitos no interior do modelo gerado por *quadtree*: a contração pára na *quadtree*, representando um número de atualizações na lista de arestas menor. Segundo, para evitar uma procura na lista de arestas ativas a cada novo elemento formado, é criada uma estrutura de dados auxiliar: cada nó desta estrutura é referenciado pelo identificador de um vértice da malha e contém uma lista com as arestas que esse vértice formou (cada aresta é identificada por sua posição na lista de arestas ativas). Assim, ao se formar uma aresta nova, para verificar se ela já existe na lista de arestas ativas, basta procurar na estrutura auxiliar de um dos vértices dessa aresta, sem precisar fazer uma busca completa cada vez na lista. Isto acelera a busca, já que a lista de arestas ativas em certos momentos da contração pode ser bastante grande enquanto que, na estrutura auxiliar, cada vértice forma um número reduzido de arestas.

5.2.4 *Finalização da Contração do Contorno*

Se após a atualização, a lista de arestas ativas ficar vazia, significa que a malha foi gerada, pois todas as arestas já foram utilizadas como base. Senão, seleciona-se uma nova aresta e repete-se o processo. A figura 5.10 mostra a malha final gerada.

5.2.5 *Suavização da Malha*

Finalmente, após ser gerada toda a malha, executa-se um processo de *suavização*. A suavização é simplesmente uma média, para cada vértice, considerando as coordenadas

dos vértices pertencentes aos elementos finitos adjacentes ao vértice em questão. Essa média é repetida quatro ou cinco vezes para dar um resultado final satisfatório e afeta somente os vértices interiores, já que não se pode alterar as coordenadas dos vértices originais do contorno. A figura 5.11 mostra a malha final suavizada; comparando-se com a figura 5.10, pode-se notar a melhora conseguida na razão de aspecto dos elementos.

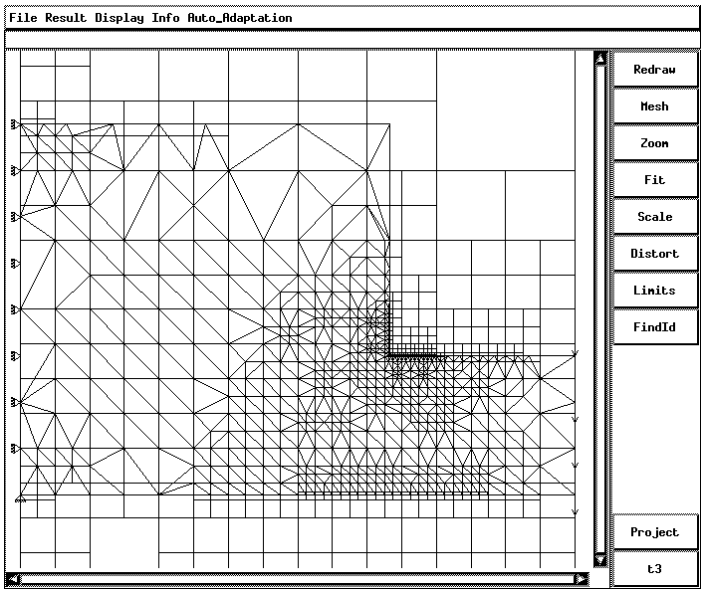


Figura 5.10: Malha final gerada.

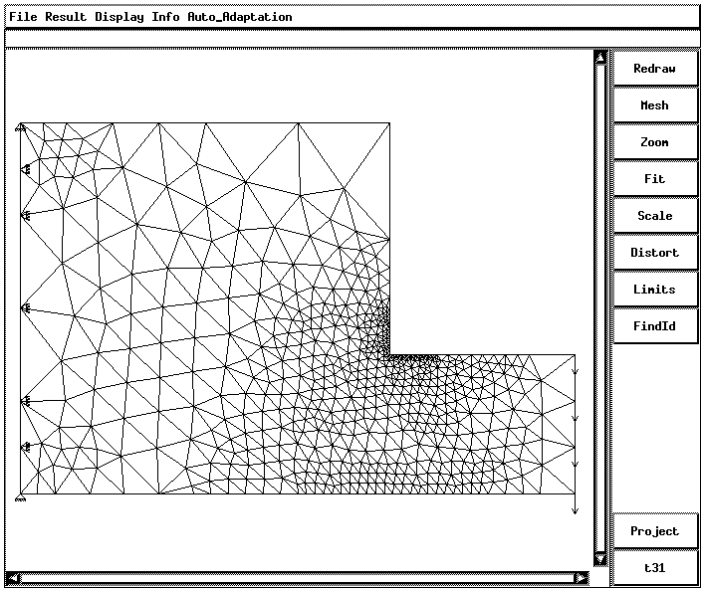


Figura 5.11: Malha final suavizada.

5.3 *Comparação com Outros Algoritmos Baseados em Quadtree*

O algoritmo descrito neste capítulo basicamente pode ser considerado um algoritmo híbrido entre o proposto por Shephard (Baehmann et al., 1987) e o proposto por Potyondy (1993), tentando combinar o que há de melhor nos dois.

O algoritmo proposto por Shephard é todo baseado na estrutura *quadtree*, isto é, a malha é totalmente gerada através da manipulação das células da *quadtree*, criadas de acordo com o modelo em questão. Este algoritmo é robusto, e bastante rápido, já que percorre uma árvore quaternária para gerar a malha. Entretanto, ele tem a característica de não manter a discretização original do contorno, criando novos vértices no contorno não definidos originalmente.

O algoritmo de Potyondy (1993), assim como o de Vianna (1992), usa a *quadtree* para criar os vértices interiores do domínio que servirão para a triangulação junto com a contração do contorno para gerar a malha. Esses algoritmos mantêm a discretização de bordo original do usuário, mas são de complexidade quadrática, pois testam a aresta base contra todos os outros vértices possíveis candidatos, mesmo os que estão longe desta aresta e nunca dariam um melhor elemento finito. Em modelos maiores isto leva a um tempo excessivo na criação da malha.

O algoritmo proposto procura aliar as boas características de ambos os métodos citados e criar uma geração eficiente, robusta e rápida. A principal vantagem deste algoritmo é que ele mantém a discretização original do contorno. Em um sistema interativo, esta discretização é normalmente definida pelo usuário e é importante para fazer a ligação com outros algoritmos (e.g., mapeamentos), para simulação de propagação de trincas, e para o usuário ter um maior controle sobre a malha a ser gerada. Na estratégia auto-adaptativa proposta, esta discretização é definida pelo auto-refinamento das fronteiras, o que resulta em uma discretização mais regular no contorno, e é de fundamental importância para modificações locais da malha.

Dentro deste contexto, o algoritmo usa a *quadtree* para gerar malha somente no interior do modelo, simplificando um pouco o processo proposto por Shephard (Baehmann et al., 1987), pois não precisa das operações para tratamento da *quadtree* na região próxima do contorno a fim de evitar elementos finitos ruins nesta área. Na fase da contração do contorno, a substituição da procura de vértices para formação de triângulos em uma lista de vértices pela procura nas células adjacentes para cada aresta base evita testes exaustivos contra vértices longe da aresta base que não dariam elementos finitos melhores. Isto torna o desempenho do algoritmo bem superior, pois ele

não é mais de de complexidade quadrática, já que ele percorre a árvore *quadtree* para achar a adjacência, levando a ganhos bem altos em modelos grandes e complexos. Além disto, a estrutura auxiliar criada para armazenar as arestas formadas por cada vértice evita buscas exaustivas na lista de arestas ativas toda vez que um novo elemento finito é formado, facilitando a atualização da estrutura de dados.

6

Análise de Resultados Numéricos

Esse capítulo apresenta vários resultados da estratégia auto-adaptativa proposta, com o objetivo de demonstrar a sua eficácia. Também é enfocado a capacidade de visualização gráfica do modelo e de seus resultados através do uso da computação gráfica interativa. Todo este escopo constitue o ambiente desenvolvido no módulo auto-adaptativo, que se encarrega da geração da nova malha e permite o pós-processamento gráfico dos resultados tanto da malha final como a cada etapa do processo.

6.1 *Sistema Gráfico para Visualização Integrada*

O módulo auto-adaptativo foi desenvolvido baseado no uso do sistema gráfico GKS/puc (TeCGraf, 1989) e do IUP/LED, um *toolkit* portátil para interface com o usuário (Levy, 1993), além de ter sido desenvolvido em linguagem C, utilizando uma disciplina de programação orientada a objetos. Esse ambiente permite a portabilidade automática, sem alteração de código, para várias plataformas onde o sistema funciona, como as estações de trabalho baseadas no sistema operacional Unix (Sun Sparcstation, IBM RS6000) e micro-computadores da linha PC.

O módulo fornece informações qualitativas e quantitativas da malha e dos resultados referentes à mesma, a cada etapa do processo, ou no resultado final da adaptação da malha. Informações quantitativas da malha podem ser obtidas através de funções de consulta, que fornecem dados como número de nós, inicidência de elementos, etc. Informações qualitativas consistem basicamente na visualização de resultados do modelo de dois tipos: resultados escalares, que se caracterizam por grandezas escalares representadas nos pontos de Gauss ou nos pontos nodais, suavizadas ou não; e resultados vetoriais, representados por três valores referentes às componentes de um vetor. Para o módulo, não importa a natureza da grandeza que está sendo representada, isto é, um campo escalar de temperatura é representado da mesma maneira que um campo escalar de uma componente de tensão σ_x, por exemplo. Atualmente essa representação só está disponível para tensões, pois por enquanto a estratégia auto-adaptativa só se

aplica a problemas de elasticidade. Existem ainda algumas funções que manipulam a imagem do modelo, tais como detalhe (*detail*), ampliação (*zoom*), enquadramento (*fit*), para auxiliar na visualização do modelo.

A entrada para o processo de auto-adaptação é feita através de um arquivo padrão, denominado arquivo neutro, que tenta ser suficientemente genérico para abordar todas as particularidades envolvidas nos diversos tipos de análise por elementos finitos.

6.1.1 *Estratégia de Múltiplas Janelas*

O sistema de interface utilizado, o IUP/LED, não só assegura uma portabilidade entre várias plataformas, como permite a exploração de várias idéias de gerenciamento de janelas múltiplas para dar ao usuário meios mais eficientes de visualização dos resultados. Desta maneira, o usuário não fica preso somente a uma janela para representar todas as respostas que deseja, mas tem a flexibilidade de abrir várias janelas ao mesmo tempo com representações de respostas diferentes, o que favorece muito a sua análise.

Para ser possível esse gerenciamento no presente sistema, foi desenvolvida uma hierarquia de janelas de respostas que podem ser visualizadas. O nível mais alto desta hierarquia é a chamada janela principal do módulo, que indica o modelo a ser auto-adaptado com as suas condições de contorno (figura 6.1).

A janela principal do módulo é responsável pela abertura de janelas para visualização de resultados, que se situam em um nível hierárquico inferior à ela. Essas janelas de visualização podem ser de três tipos: para representação de deformada do modelo, para representação de resultados escalares e para representação de resultados vetoriais. Essas janelas hierarquicamente inferiores, mesmo originalmente criadas para um determinado tipo de resposta, podem, uma vez abertas, ter os outros dois tipos de respostas ativadas. Ou seja, uma janela originalmente ativada para a representação da deformada do modelo pode mostrar também resultados escalares e vetorias, por exemplo. Além disso, pode-se abrir mais de uma janela de resultados, o que permite, por exemplo, se comparar dois resultados escalares diferentes ao mesmo tempo. A figura 6.2a mostra um exemplo com a janela principal e janelas filhas com representações de tensões do modelo. A janela principal também é encarregada de disparar o processo auto-adaptativo do modelo. Neste caso, todas as janelas de nível hierárquico inferior são fechadas, já que a malha do modelo vai ser alterada e conseqüentemente os resultados atuais não podem ser representados.

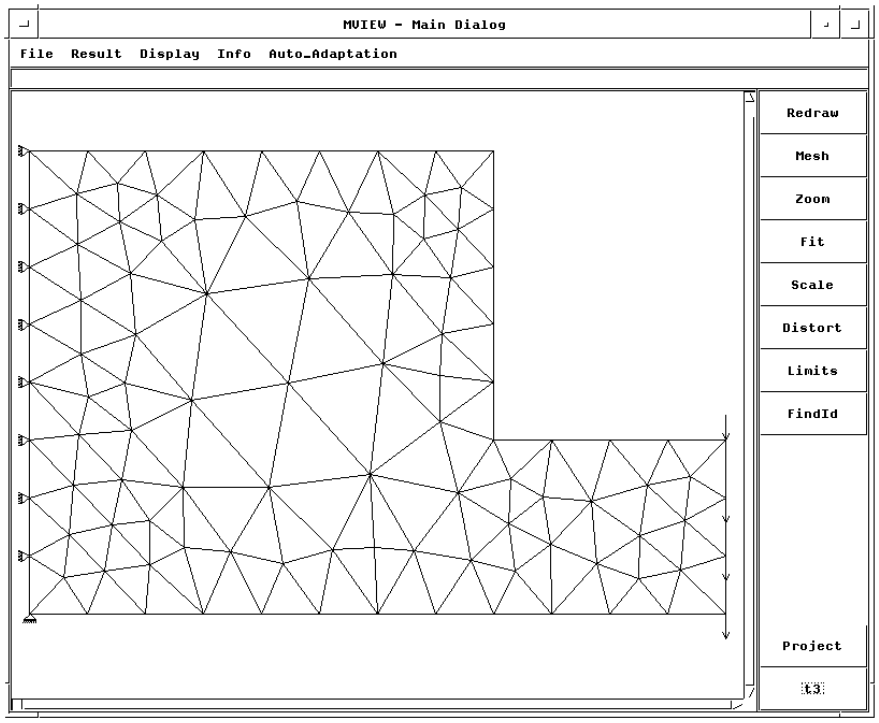


Figura 6.1: Janela principal.

Um terceiro nível de hierarquia corresponde a janelas referentes a gráficos de respostas do modelo. Esses gráficos podem ser de três tipos: gráficos para história de resultados de um determinado ponto (nó ou ponto de Gauss) ao longo de diversos passos, no caso de uma análise dinâmica ou transiente; gráficos para grandezas escalares ao longo de um caminho entre dois pontos selecionados pelo usuário, em caso de elementos de interface; e gráficos para representação de resultados escalares, como por exemplo um diagrama de tensões onde o usuário passa uma linha arbitrária sobre o modelo e o programa apresenta o diagrama de tensões ao longo da linha fornecida. Uma janela de gráfico sempre se refere a uma janela de resultados aberta. A figura 6.2b mostra um exemplo de um modelo, com uma janela de resultados e uma janela de gráfico relativo a um diagrama de tensões da resposta.

As funções de auxílio de visualização, como ampliação (*zoom*), escala (*scale*), etc. estão disponíveis para os três níveis de hierarquia das janelas, enquanto que as funções de consulta de dados quantitativos da malha existem apenas nos dois primeiros níveis.

Figura 6.2: (a) Janela principal com janelas de representações de resultados.
(b) Janela principal com uma janela de resultados e outra de gráfico.

6.1.2 *Representação de Resultados*

As técnicas de representação de resultados foram baseadas em trabalhos anteriores da linha de pesquisa (Guimarães, 1992; Mview, 1993). A representação de resultados escalares é feita através de isofaixas preenchidas com uma cor referente a uma escala de cores, que está associada a uma escala de valores dos resultados. Deste modo, pode-se obter uma boa descrição do comportamento da estrutura, bem como a identificação de regiões com gradientes elevados. A quantidade de cores na escala é fixa e foi definida igual a 9, com os seus valores extremos correspondendo aos valores máximos e mínimos do campo escalar. Os intervalos, inicialmente, são obtidos através de interpolação linear dos valores extremos. Existem, no entanto, funções que permitem alterar a escala de valores. A visualização dos resultados escalares, através de isofaixas, pode ser feita com base em resultados de três tipos: resultados nos pontos de Gauss, resultados nos pontos nodais sem suavização, ou nos pontos nodais suavizados. As figuras 6.3, 6.4 e 6.5 mostram exemplos destes três tipos de representação.

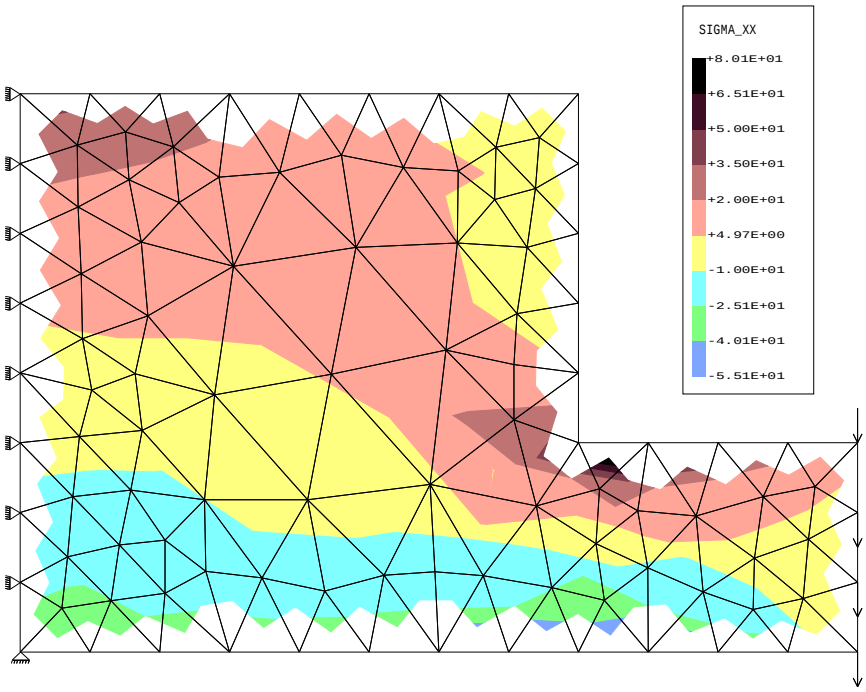


Figura 6.3: Representação de resultados em pontos de Gauss.

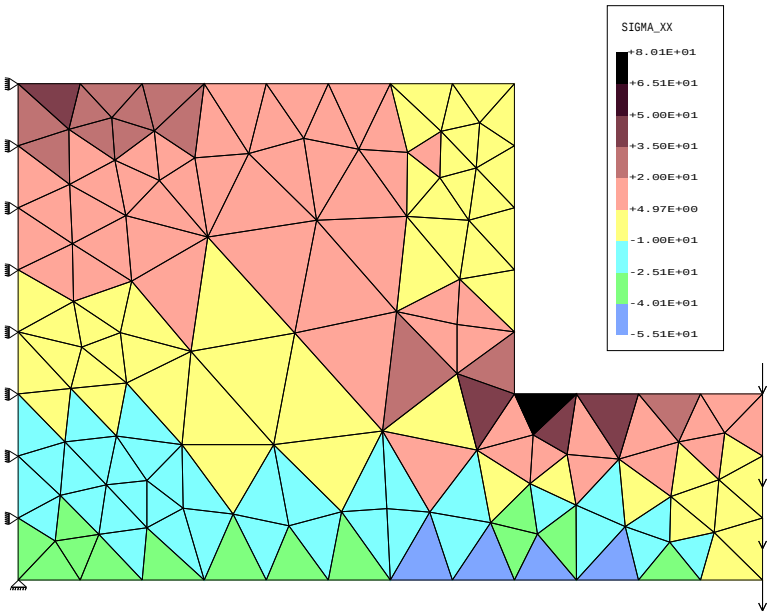


Figura 6.4: Representação de resultados nodais não suavizados.

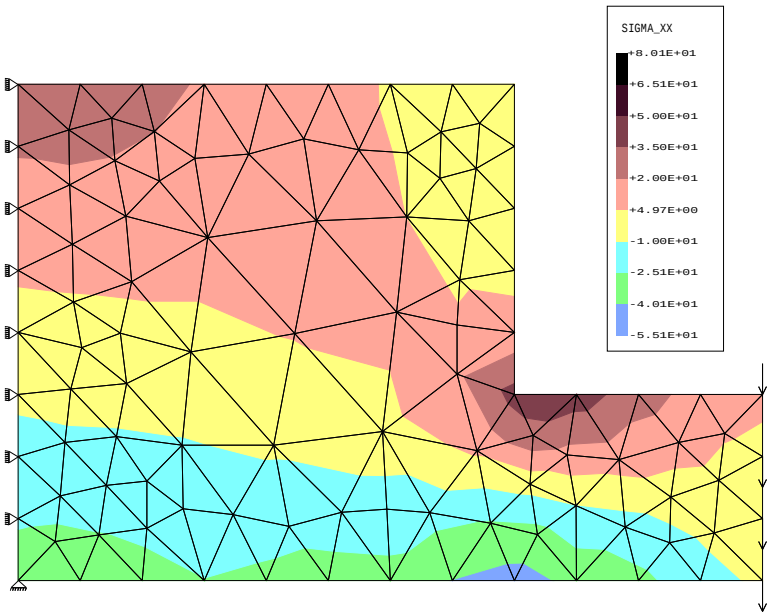


Figura 6.5: Representação de resultados nodais suavizados.

A representação dos resultados vetoriais é feita através da configuração deformada do modelo, por malha, mostrada na figura 6.6, ou por setas, mostrada na figura 6.7.

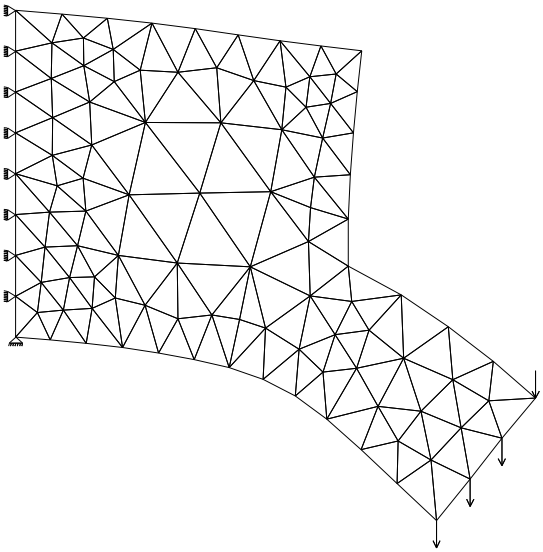


Figura 6.6: Representação do modelo deformado por malha.

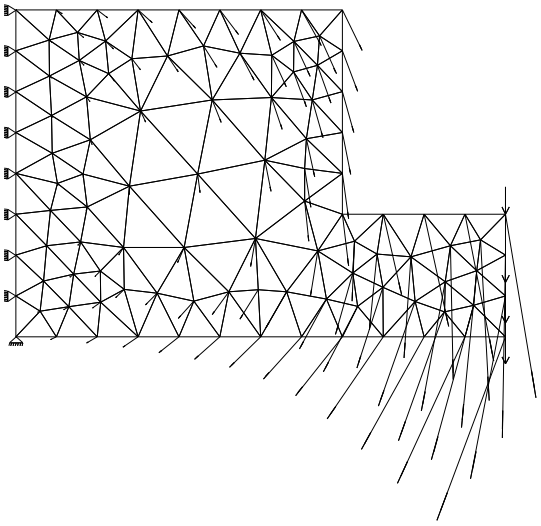


Figura 6.7: Representação do modelo deformado por setas.

Outra forma de representação disponível é a plotagem de vetores que indicam a direção e a intensidade da grandeza vetorial nos pontos de Gauss, como por exemplo a representação das tensões principais do modelo.

6.2 *Exemplos Numéricos*

Nesta seção, apresenta-se cinco exemplos de problemas lineares elásticos bidimensionais para demonstrar a eficácia da estratégia proposta no que diz respeito ao remalhamento. Deve-se considerar em todos os exemplos que o estimador de erro utilizado apresenta um baixo grau de precisão. Entretanto, a estratégia usada tende a compensar isso, pois as malhas geradas apresentam uma distribuição uniforme do erro estimado, o que caracteriza uma malha considerada "ótima" e atinge os níveis de erro desejados.

Os exemplos apresentam resultados para análise auto-adaptativa usando elementos lineares (T3) e quadráticos (T6). As comparações com resultados "exatos" são feitas através do conhecimento da solução analítica para os exemplos ou usando soluções aproximadas extrapoladas de método adaptativo *h-p* (Zienkiewicz et al., 1989), para validar a estratégia proposta. A efetividade do estimador de erro é dada pelo *índice de efetividade*

$$\theta \;=\; \frac{\|\overline{e}\|}{\|e\|}, \tag{6.1}$$

onde $\|\overline{e}\|$ é a norma de energia do erro estimada calculada pela equação (3.17) e $\|e\|$ é a norma de energia do erro "exata" dada pela relação (Zienkiewicz et al., 1989)

$$\|e\|^2 \;=\; \|u\|^2 - \|\hat{u}\|^2. \tag{6.2}$$

O erro é dito assintoticamente "exato" se θ converge para um à medida que o erro converge para zero. Demonstra-se que o estimador de erro dado pela norma de energia é assintoticamente exato supondo-se que o máximo tamanho de um elemento finito tenda para zero (Ainsworth et al., 1989). Na prática, esse tamanho não tende para zero; portanto é necessário que o estimador seja efetivo também dentro de uma faixa típica de análise de problemas de engenharia, especialmente porque o estimador é usado para determinar o critério de parada para a solução adaptativa. Normalmente se usa que o critério de parada dado pelo erro relativo

$$\eta \;=\; \frac{\|e\|}{\|u\|} \tag{6.3}$$

e aproximado por

$$\overline{\eta} \;=\; \frac{\|\overline{e}\|}{\sqrt{\|\hat{u}\|^2 + \|\overline{e}\|^2}} \tag{6.4}$$

(conforme descrito no capítulo 3) seja igual ou inferior a 10% para elementos lineares e 5% para elementos quadráticos.

6.2.1 *Exemplo 1 - Placa Quadrada com Furo Quadrado*

O primeiro problema analisado é uma placa quadrada com um furo quadrado, submetida a um carregamento lateral unitário. Pelas condições de simetria, apenas um quarto da placa é modelado. É considerado para solução numérica do problema estado plano de tensões, com módulo de elasticidade $E = 10^5$ e coeficiente de Poisson $\nu = 0.3$, e a energia de deformação analítica para o problema é 0.1556660, que corresponde a um valor para a $\|u\|^2$ de 0.311332. A figura 6.8 mostra o exemplo e seus atributos.

A análise do problema utilizando elementos lineares (T3) e quadráticos (T6) é mostrada nas figuras 6.9 e 6.10. Como critério de parada, utilizou-se um erro relativo máximo η^* de 10% para elementos lineares e 5% para os quadráticos. A análise do problema mostra que nos dois casos foram precisos apenas dois passos de auto-adaptação. No elemento linear, apenas um passo trouxe o erro para 13.45%, ou seja, quase no limite desejado; para os quadráticos, essa convergência se acentuou ainda mais, com o erro caindo para 5.7% no primeiro passo de auto-adaptação. As figuras 6.11 e 6.14 mostram as taxas de convergência para os elementos linear e quadrático onde se comparam com valores teóricos de 0.5 para elementos lineares (p=1) e 1.0 para quadráticos (p=2). As figuras 6.12 e 6.15 mostram o gráfico do indice de efetividade θ pelo número de graus de liberdade da malha, observando-se que o seu valor tende para um à medida que o erro no modelo decresce.

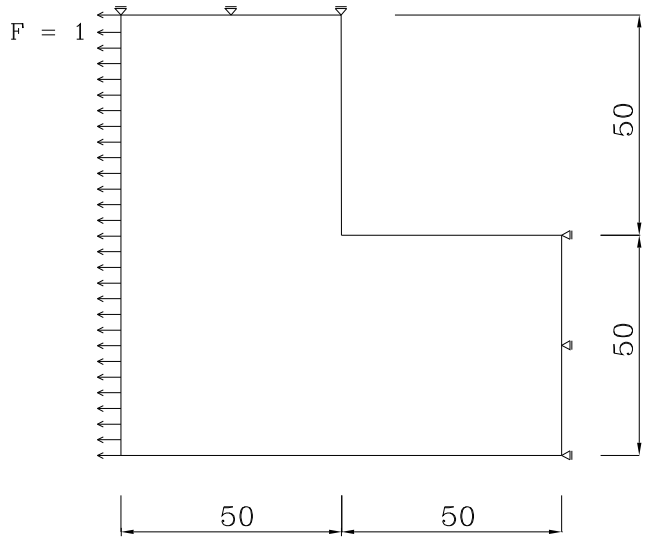


Figura 6.8: Placa quadrada com furo quadrado.

Para ilustrar a distribuição da norma de energia para o erro $\|\overline{e}\|$ através dos passos de auto-adaptação, são mostradas a distribuição de $\|\overline{e}\|$ através das malhas para o elemento quadrático na figura 6.17. Nota-se que essa norma vai se distribuindo igualmente pelos elementos, com o canto reentrante tendo os valores mais altos, como era de se esperar, com a malha final mostrando uma distribuição bem uniforme de $\|\overline{e}\|$.

Esse problema também foi analisado por Baehmann (1989) somente para elementos quadráticos, que também usa uma técnica de enumeração espacial recursiva, com a diferença para este trabalho que a adaptação da malha é toda ela baseada na técnica convencional de *quadtree* (sem considerar a triangulação da faixa entre a árvore no interior e o contorno por uma técnica de contração por Delaunay), e também foram necessários dois passos de adaptação. Outras técnicas para adaptação da malha foram usadas para solução deste problema, como a usada por Lee e Lo (1992), onde foram necessários três passos para elementos lineares e dois para os quadráticos. Santana (1993), que implementou uma técnica de adaptação da malha baseada em trabalho de Peraire et al. (1987), também analisou o problema, onde também foram necessários dois passos de adaptação para elementos lineares e quadráticos.

Gráficos do tempo computacional pelo número de graus de liberdade são mostrados nas figuras 6.13 e 6.16. Para efeito de tempo computacional, o processo de auto-adaptação foi divido em duas etapas: a realizada pelo módulo de análise numérica, que também calcula a estimativa do erro no modelo, e a realizada pelo módulo auto-adaptativo, encarregado da geração da nova malha, redistribuição dos atributos e reordenação nodal, sendo neste trabalho utilizada a técnica de Cuthill-McKee (1969).

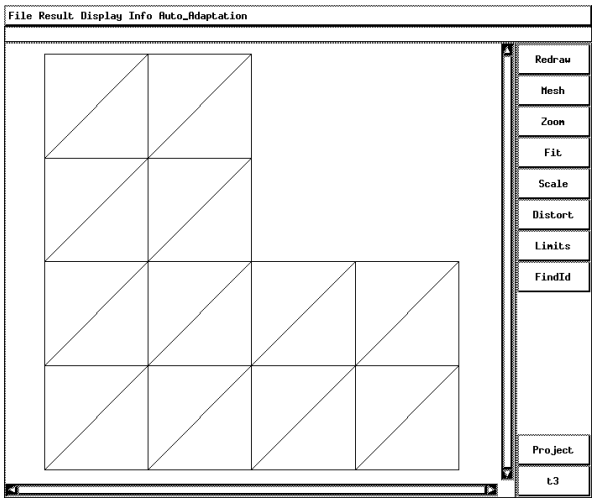
Os exemplos foram rodados em uma SUN Sparcstation 10 para efeito de medição de tempo de execução. Nota-se que para elementos quadráticos, por exemplo, o último passo de auto-adaptação, que possui logicamente mais graus de liberdade, levou menos de 4 segundos para a análise e menos de 2 segundos para a auto-adaptação, o que comprova a eficácia do método e a aplicabilidade a problemas práticos, pois o fator tempo é uma variável importante neste contexto. A análise de elementos lineares no último passo leva em torno de 6 segundos para análise e 12 segundos e meio para a auto-adaptação, sendo que a malha original tem 464 graus de liberdade e a malha final tem 1365.

O tempo computacional para este problema também foi analisado por Baehmann (1989) e foi dividido em quatro etapas: análise numérica, estimativa de erro, geração da nova malha e reordenação nodal. Apesar das análises terem sido feitas

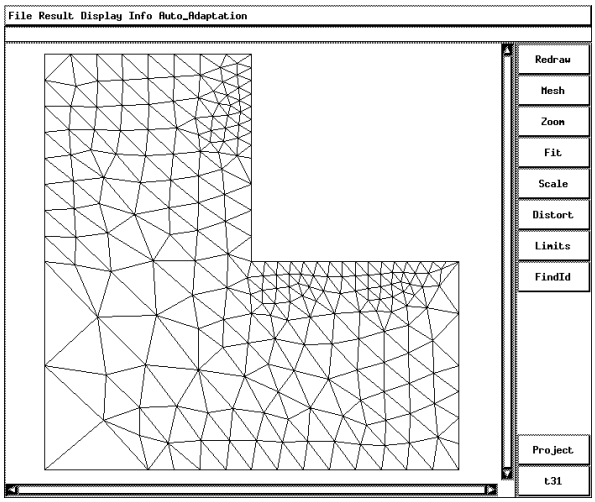
em uma plataforma diferente (VAX 785), se forem somados os tempos de geração de malha e reordenação nodal, que correspondem ao feito pelo módulo auto-adaptativo deste trabalho, também se obtém tempos da mesma ordem de grandeza, o que era de se esperar, pois técnicas de enumeração espacial recursiva percorrem árvores para geração da malha, o que as tornam muito rápidas e eficientes. Não foi possível avaliar a influência dos diferentes tratamentos para a discretização do contorno entre este trabalho e o trabalho de Baehmann no que se refere à eficiência computacional.

O tempo computacional também foi medido no trabalho de Santana (1993) para o último passo de adaptação, e foram atingidos no seu trabalho valores bem superiores: cerca de 244 segundos para adaptação da malha, se somados os tempos de geração da malha e aprimoramento da mesma, para elementos lineares e cerca de 122 segundos para elementos quadráticos. Não foi especificado no trabalho a plataforma utilizada, mas estes tempos indicam uma ordem de complexidade bem maior para o algoritmo por ele utilizado.

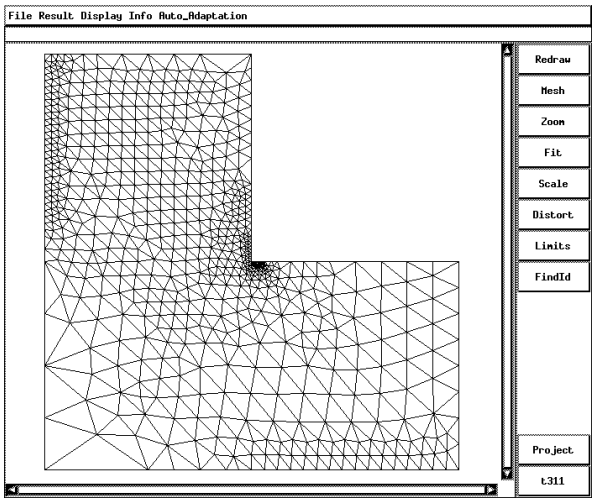
Observa-se que existe uma diferença entre os aspectos das malhas geradas para elementos lineares e quadráticos, sendo que para os lineares (T3) a malha apresenta regiões fora da concentração de tensões com alto grau de refinamento. Os outros trabalhos da literatura também mostram este mesmo aspecto. Isto pode ser atribuído à uma má qualidade do elemento linear e também à pouca precisão do estimador de erro adotado.



a - NN = 21, NE = 24
DOF = 36, $\overline{\eta} = 33.27\%$



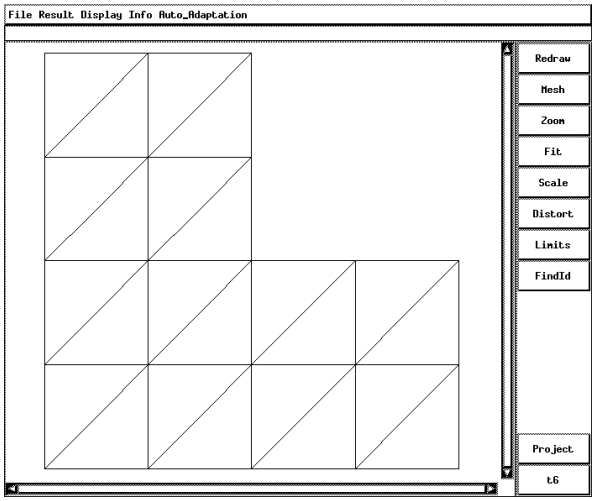
b - NN = 240, NE = 413
DOF = 464, $\overline{\eta} = 13.45\%$



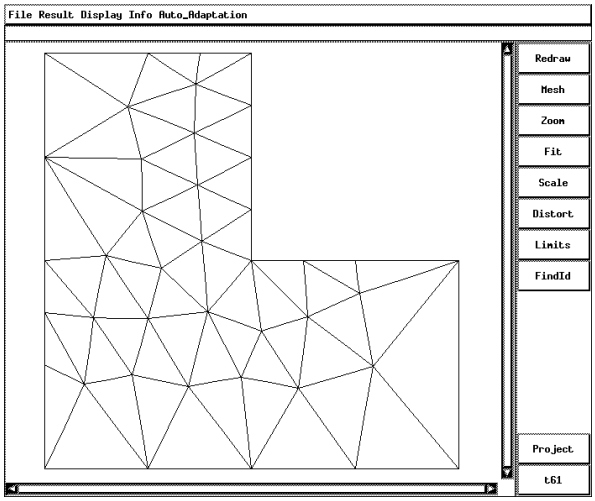
c - NN = 693, NE = 1265
DOF = 1365, $\overline{\eta} = 7.77\%$

Figura 6.9: Exemplo 1 com T3: malha inicial (a), malhas seguintes (b-c).

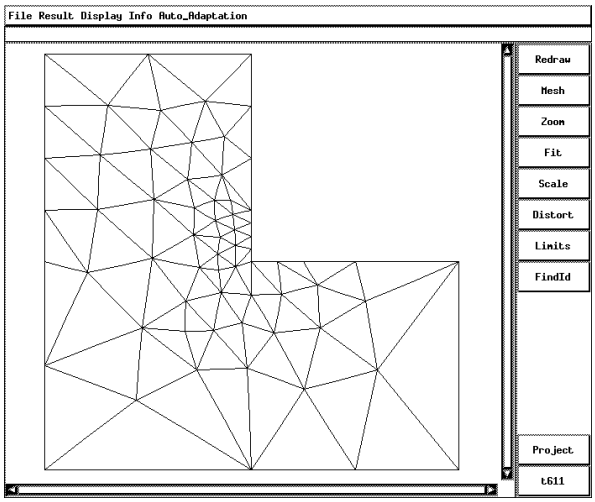
NN = número de nós, NE = número de elementos, DOF = número de graus de liberdade



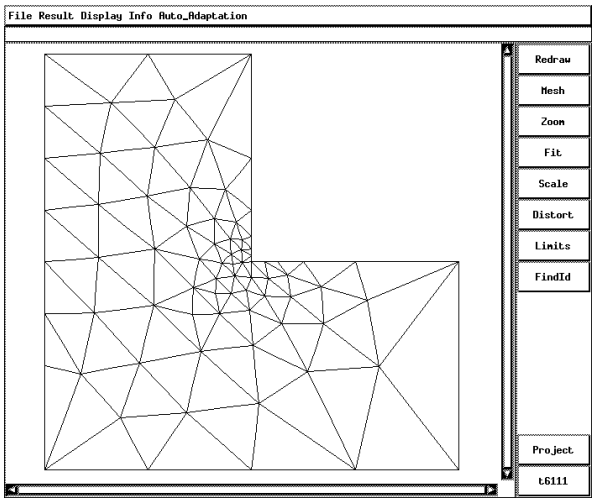
a - NN = 65, NE = 24
DOF = 120, $\overline{\eta} = 9.02\%$



b - NN = 141, NE = 60
DOF = 272, $\overline{\eta} = 5.77\%$



c - NN = 238, NE = 107
DOF = 468, $\overline{\eta} = 3.81\%$



d - NN = 280, NE = 127
DOF = 552, $\overline{\eta} = 2.66\%$

Figura 6.10: Exemplo 1 com T6: malha inicial (a), malhas seguintes (b-d).

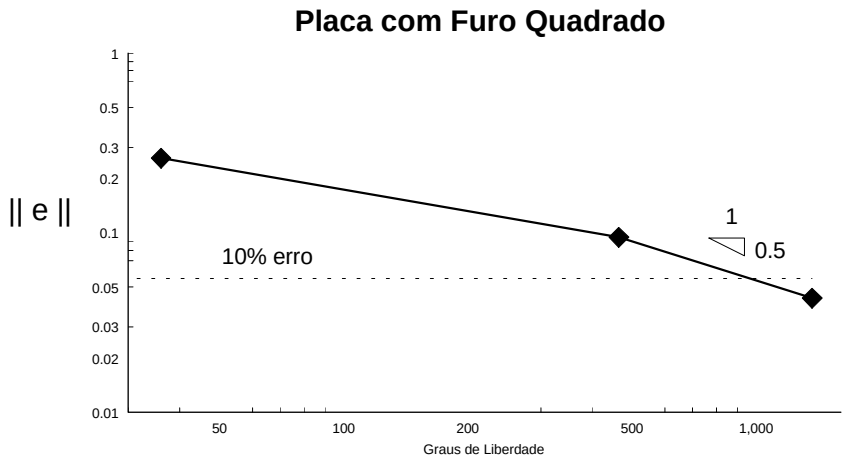


Figura 6.11: Taxa de convergência do exemplo 1 para T3.

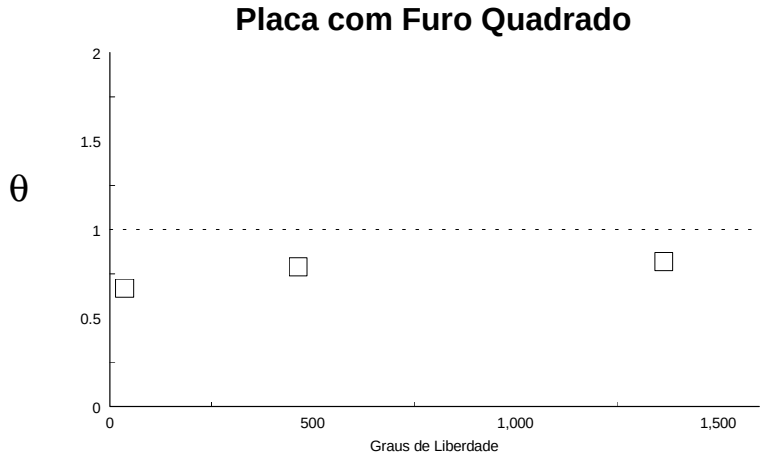


Figura 6.12: Índice de efetividade do exemplo 1 para T3.

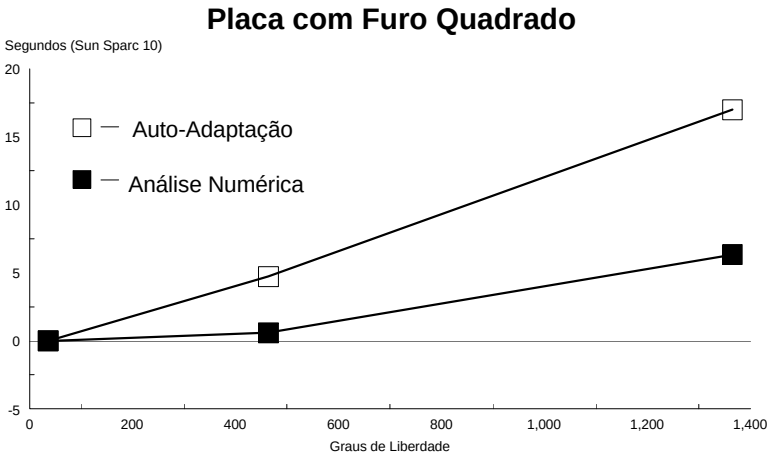


Figura 6.13: Tempo computacional do exemplo 1 para T3.

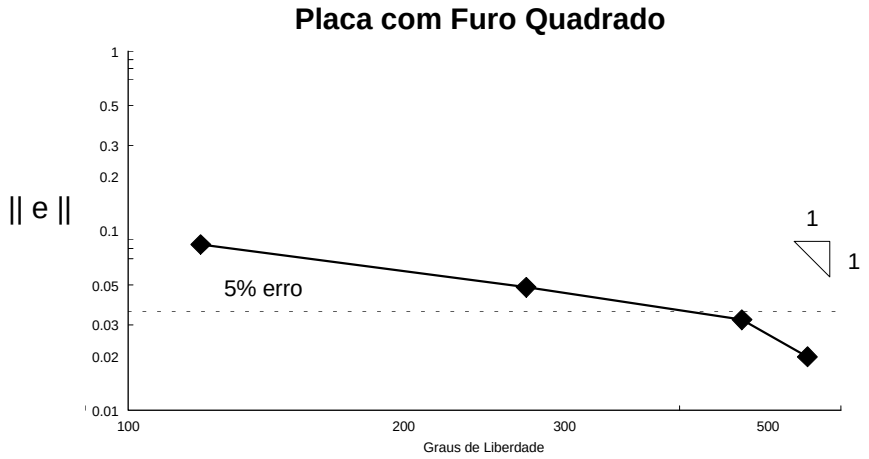


Figura 6.14: Taxa de convergência do exemplo 1 para T6.

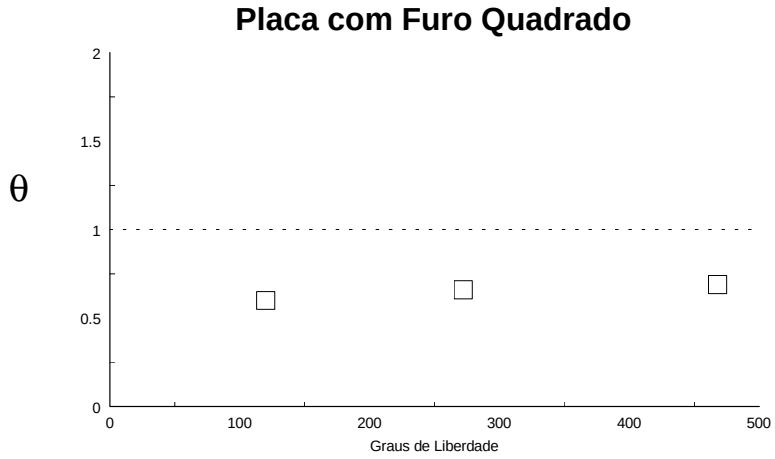


Figura 6.15: Índice de efetividade do exemplo 1 para T6.

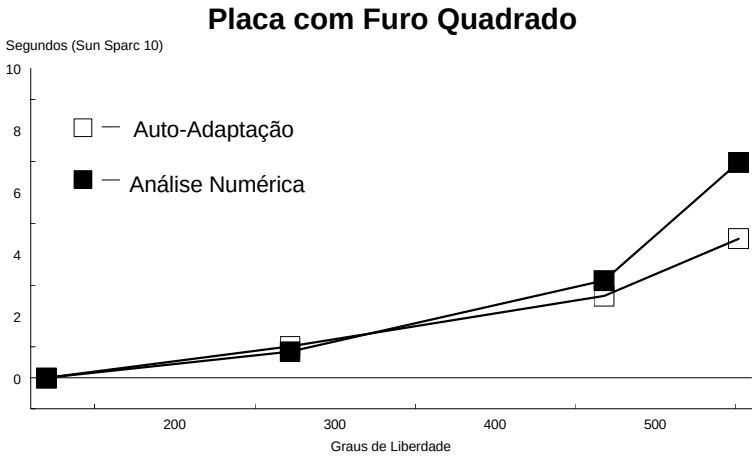
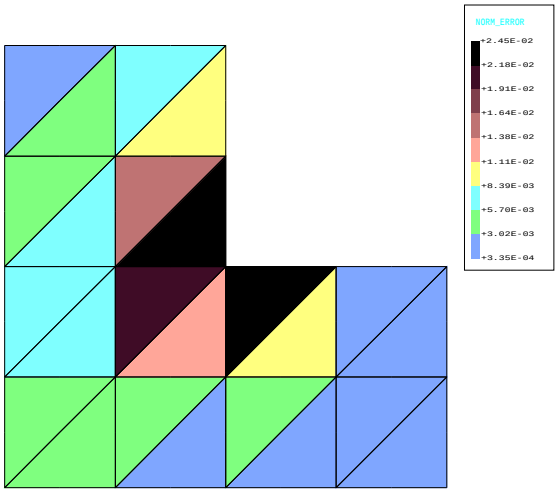
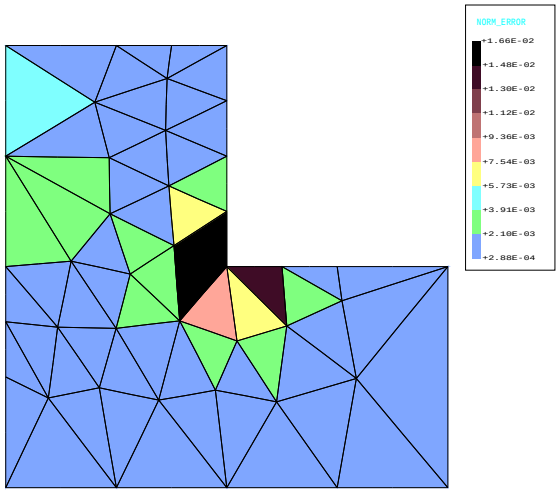


Figura 6.16: Tempo computacional do exemplo 1 para T6.



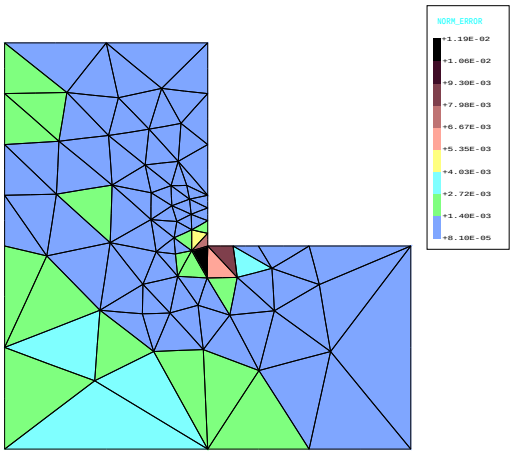
a - NN = 65, NE = 24

DOF = 120, $\overline{\eta} = 9.02\%$



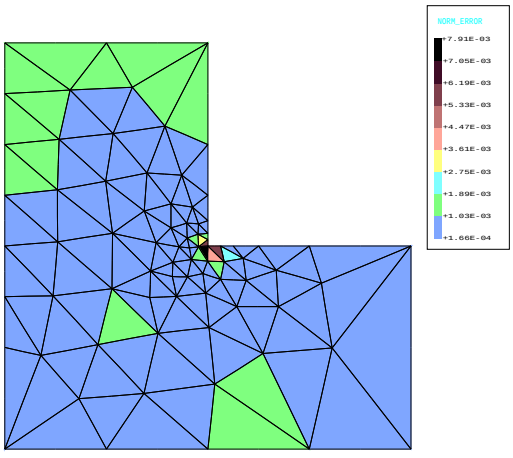
b - NN = 141, NE = 60

DOF = 272, $\overline{\eta} = 5.77\%$



c - NN = 238, NE = 107

DOF = 468, $\overline{\eta} = 3.81\%$



d - NN = 280, NE = 127

DOF = 552, $\overline{\eta} = 2.66\%$

Figura 6.17: Norma de energia para o erro nas malhas com elemento quadrático para o exemplo 1.

6.2.2 *Exemplo 2 - Viga Curta em Balanço*

O segundo exemplo é uma viga curta em balanço sob um carregamento distribuído unitário no seu bordo superior. Considera-se para a solução deste problema estado plano de deformação, com $E = 1.0$ e $\nu = 0.3$. A energia de deformação da solução analítica resulta em um valor para $\|u\|^2$ de 1.903697. A figura 6.19 mostra o exemplo e seus atributos.

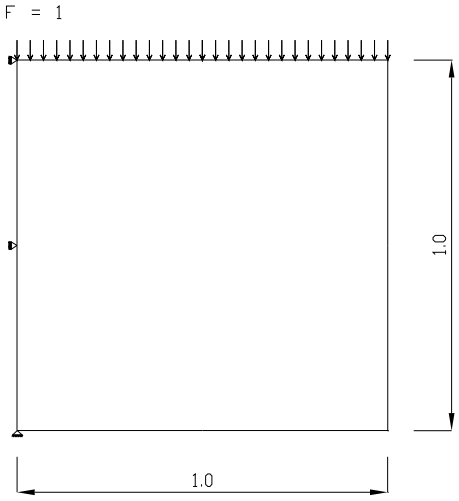


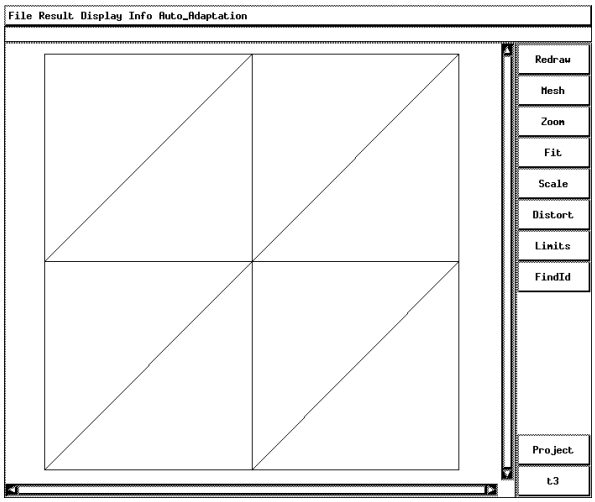
Figura 6.18: Viga curta em balanço.

As malhas geradas pelo processo auto-adaptativo estão mostradas nas figuras 6.19 e 6.20. Como critério de parada, usou-se um erro relativo máximo permitido de 8% para elementos lineares e 5% para quadráticos. Foram precisos dois passos de auto-adaptação para os dois tipos de elementos. Mais uma vez, apenas um passo traz o erro para bem próximo do desejado, com 11.80% para os lineares e 6.15% para os quadráticos. Nestes elementos, as análises foram levadas adiante e atingiu-se 2.74% na última malha. As figuras 6.21 e 6.24 mostram as taxas de convergência para o exemplo e as figuras 6.22 e 6.25 os gráficos dos índices de efetividade.

Na figura 6.27 mostra-se a distribuição da razão de erro ξ_i definida na equação 3.22 para os elementos quadráticos, onde a faixa cinza mais escura corresponde a elementos com ξ_i abaixo de 1, que representaria que não precisariam ser mais refinados, e mostra claramente que ξ_i maior que 1 indica a região onde se necessita refinar, ou seja, próximos aos cantos da face com suporte da viga, como era de se esperar.

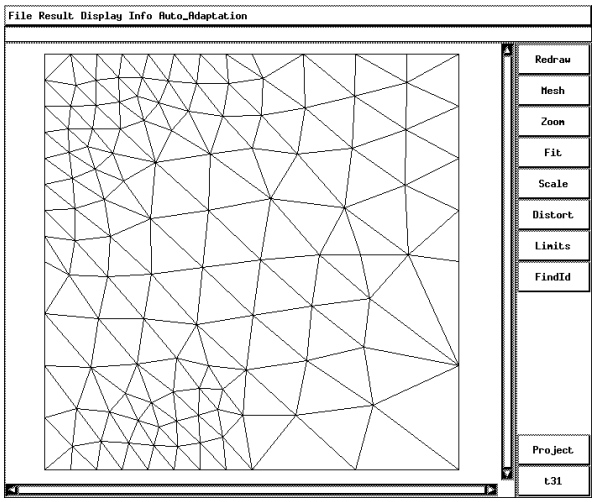
Baehmann (1989) analisou exemplo semelhante e três passos de adaptação foram necessários para trazer o erro para 3% no caso de elementos quadráticos. Ainsworth et al. (1989) também analisaram o problema através de uma técnica de adaptação por enriquecimento da malha e obtiveram uma solução abaixo de 5% após seis passos de adaptação. Lee and Lo (1992) também analisaram o problema, e partindo da mesma malha inicial usada neste trabalho, obtiveram soluções inferiores a 10% para elementos lineares (T3) e 5% para elementos quadráticos (T6) em três e dois passos de adaptação, respectivamente. Santana (1993) também analisou este exemplo e foram necessários três passos de auto-adaptação para os elementos lineares e dois para os quadráticos.

As figuras 6.23 e 6.26 mostram os gráficos dos tempos computacionais medidos para o exemplo. Mais uma vez se verificaram tempos muitos baixos para a auto-adaptação, inferiores até aos obtidos para o exemplo 1. O elemento quadrático, por exemplo, levou apenas 2.05 segundos no último passo de auto-adaptação, enquanto que o linear, 6.70 segundos. Baehmann (1989), apesar de analizar exemplo semelhante, não mostra os tempos obtidos. Santana (1993) obteve tempos ainda bem altos, cerca de 182 segundos para o elemento linear e 29 segundos para os quadráticos no último passo de adaptação (sempre se levando em consideração que não se conhece a plataforma por ele usada, mas apenas como um paramêtro de comparação do tempo do algoritmo).



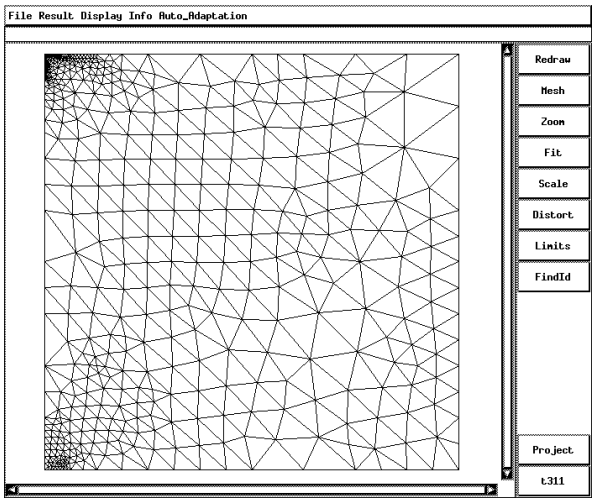
a - NN = 9, NE = 8

DOF = 12, $\overline{\eta} = 35.19\%$



b - NN = 128, NE = 214

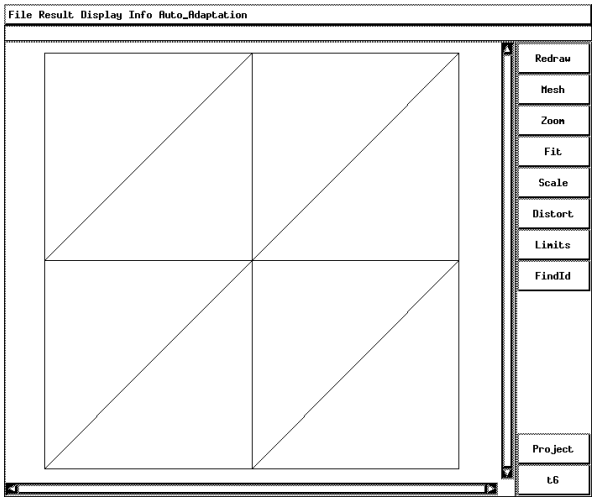
DOF = 230, $\overline{\eta} = 11.80\%$



c - NN = 525, NE = 940

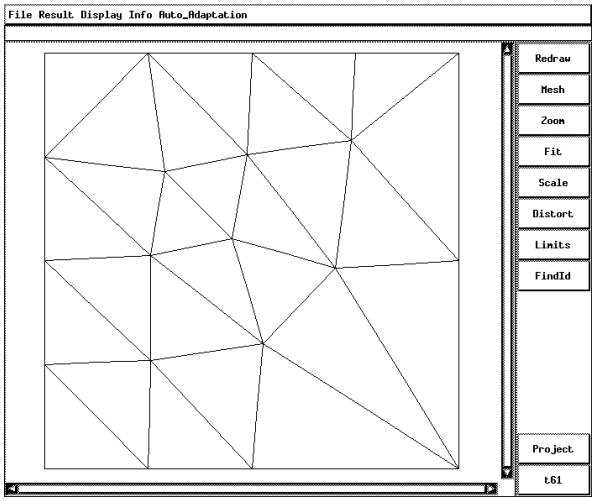
DOF = 970, $\overline{\eta} = 5.64\%$

Figura 6.19: Exemplo 2 com T3: malha inicial (a), malhas seguintes (b-c).



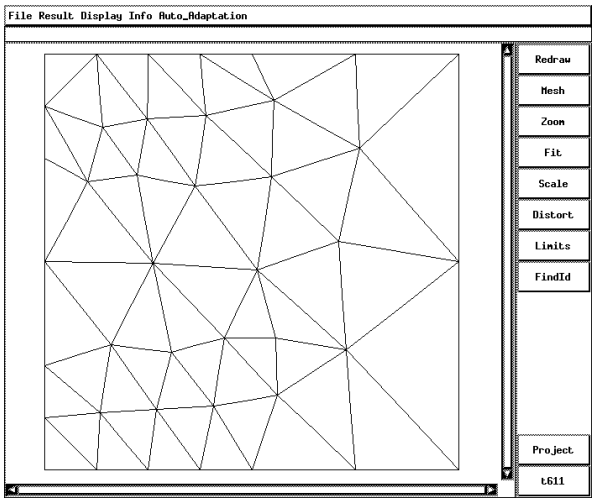
a - NN = 25, NE = 8

DOF = 40, $\overline{\eta} = 11.08\%$



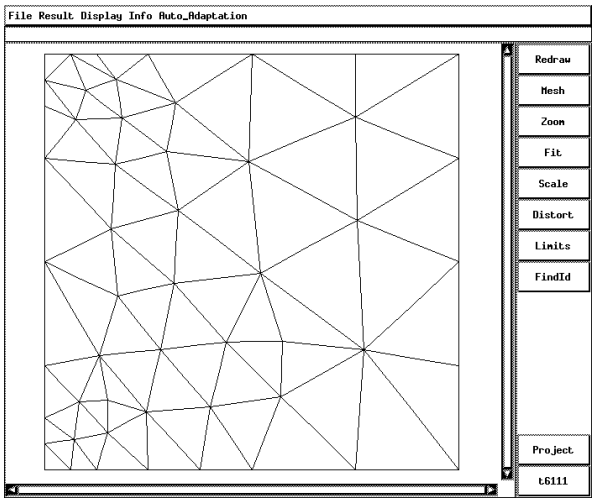
b - NN = 68, NE = 27

DOF = 118, $\overline{\eta} = 6.15\%$



c - NN = 141, NE = 60

DOF = 256, $\overline{\eta} = 3.83\%$



d - NN = 180, NE = 77

DOF = 326, $\overline{\eta} = 2.74\%$

Figura 6.20: Exemplo 2 com T6: malha inicial (a), malhas seguintes (b-d).

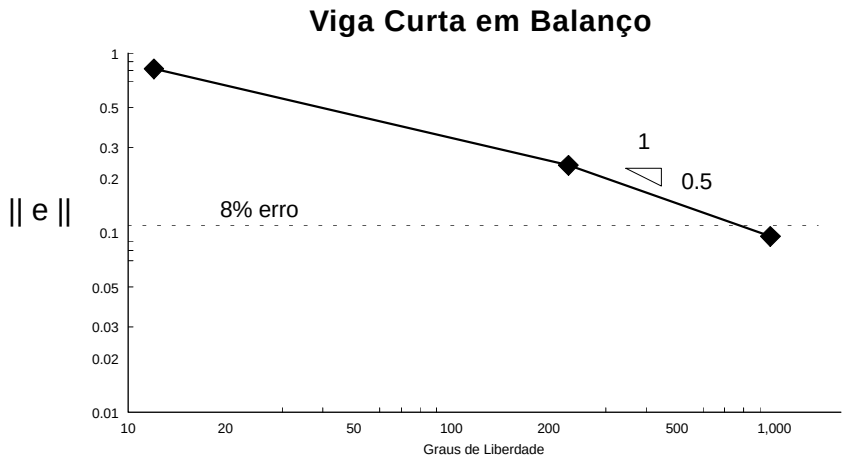


Figura 6.21: Taxa de convergência do exemplo 2 para T3.

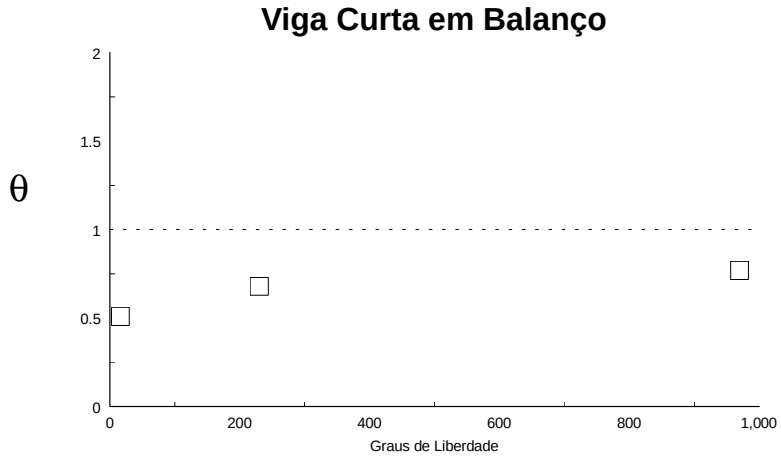


Figura 6.22: Índice de efetividade do exemplo 2 para T3.

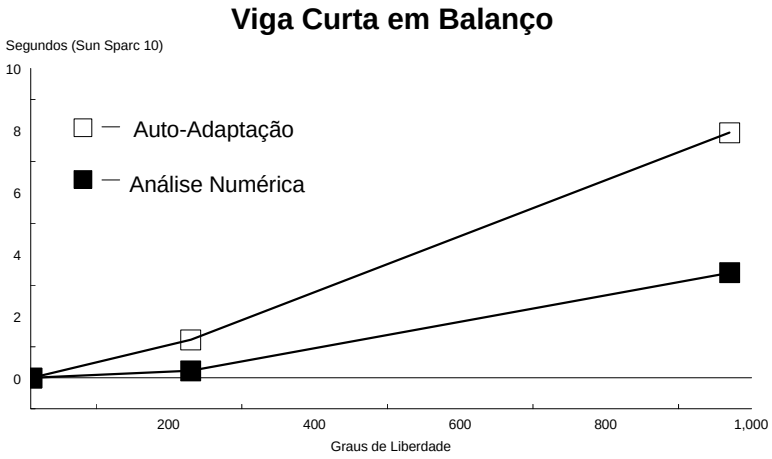


Figura 6.23: Tempo computacional do exemplo 2 para T3.

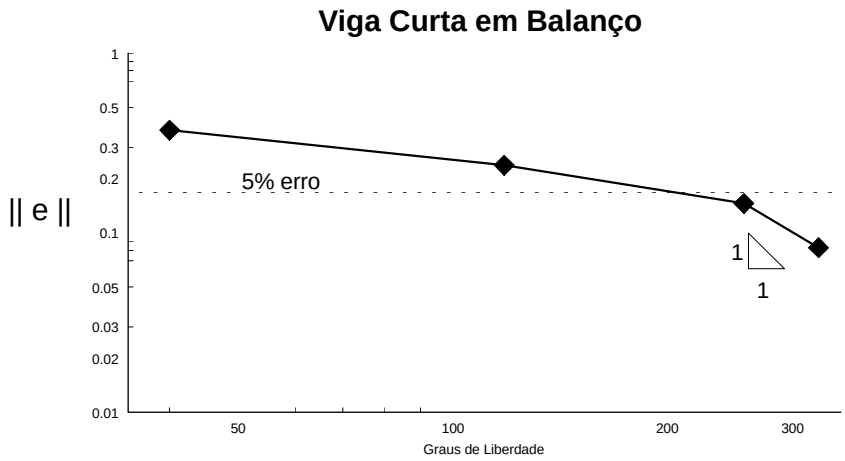


Figura 6.24: Taxa de convergência do exemplo 2 para T6.

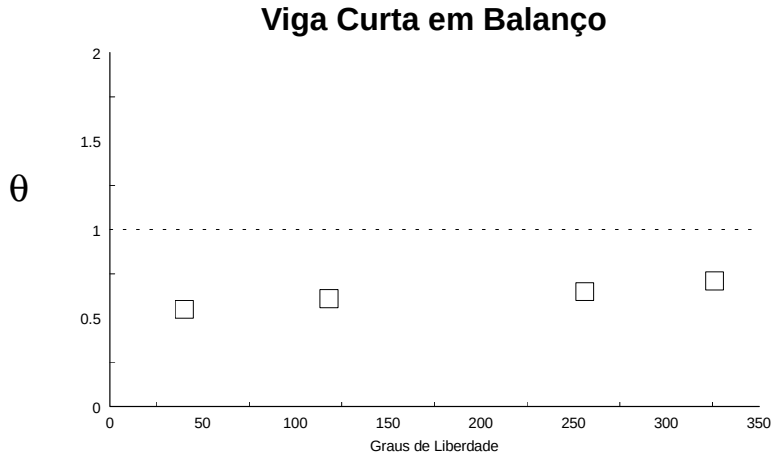


Figura 6.25: Índice de efetividade do exemplo 2 para T6.

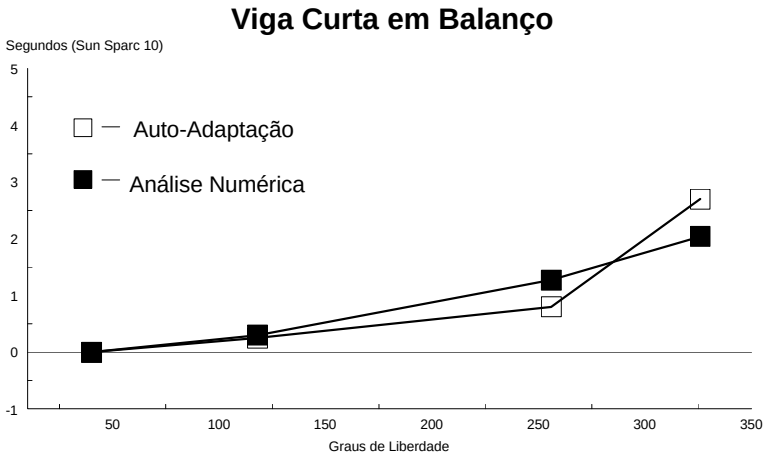
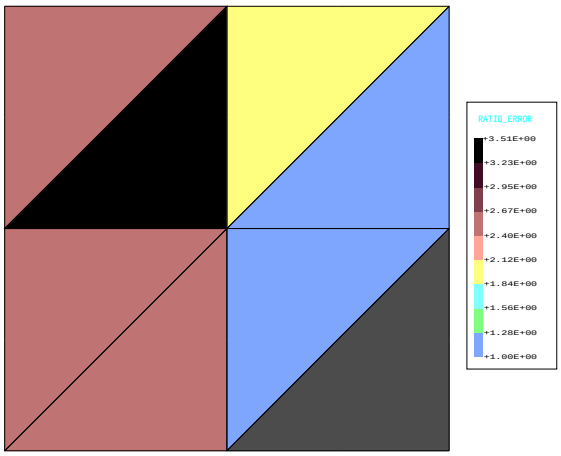
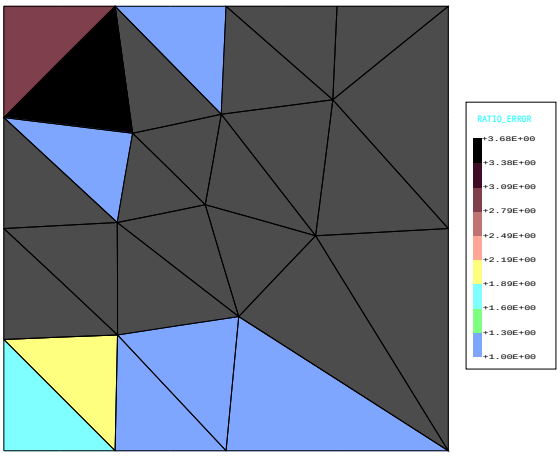


Figura 6.26: Tempo computacional do exemplo 2 para T6.



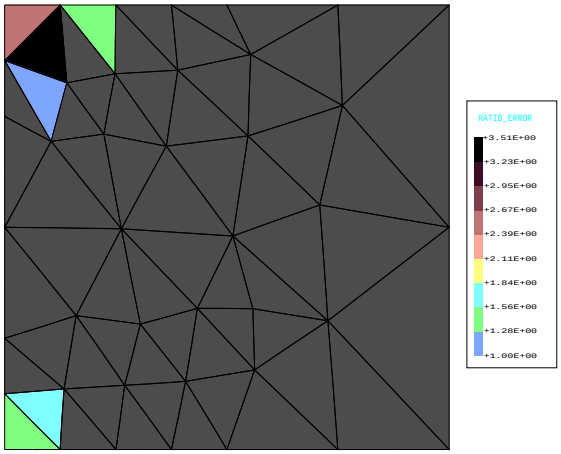
a - NN = 25, NE = 8

DOF = 40, $\overline{\eta} = 11.08\%$



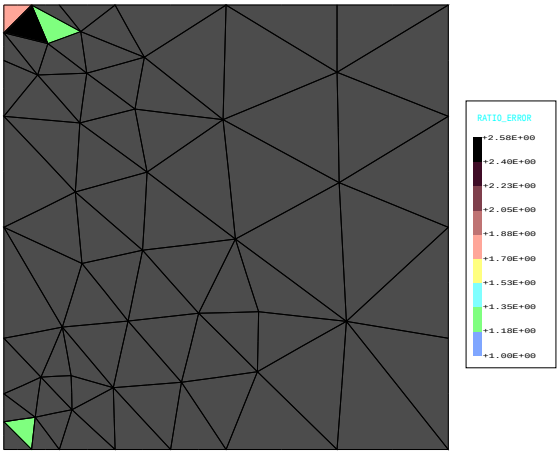
b - NN = 68, NE = 27

DOF = 118, $\overline{\eta} = 6.15\%$



c - NN = 141, NE = 60

DOF = 256, $\overline{\eta} = 3.83\%$



d - NN = 180, NE = 77

DOF = 326, $\overline{\eta} = 2.74\%$

Figura 6.27: Razão de erro nas malhas com elemento quadrático para o exemplo 2. A cor cinza mais escura corresponde a razões de erro menores que 1, que não são representados na escala de cores.

6.2.3 *Exemplo 3 - Cilindro com Pressão Interna*

O terceiro exemplo é a análise de um cilindro espesso com pressão interna unitária. Pelas condições de simetria, apenas um quarto do cilindro é analisado. A energia de deformação analítica conhecida corresponde a uma norma $\|u\|^2$ 55.815629. A figura 6.28 mostra o modelo e seus atributos.

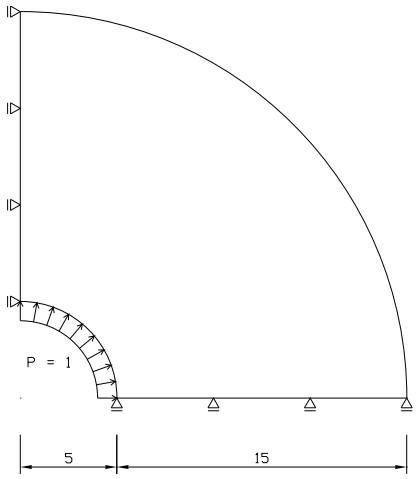
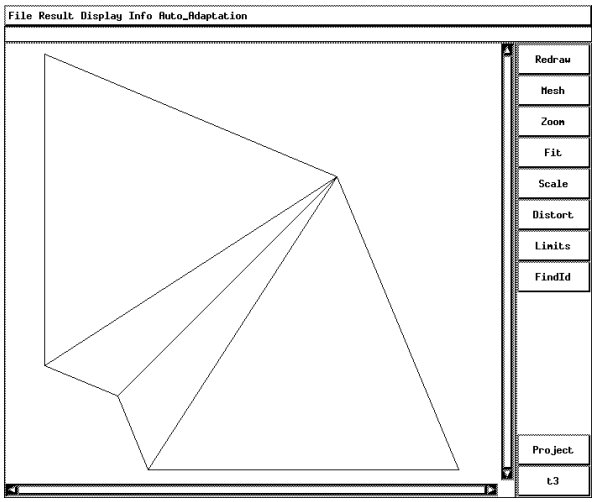


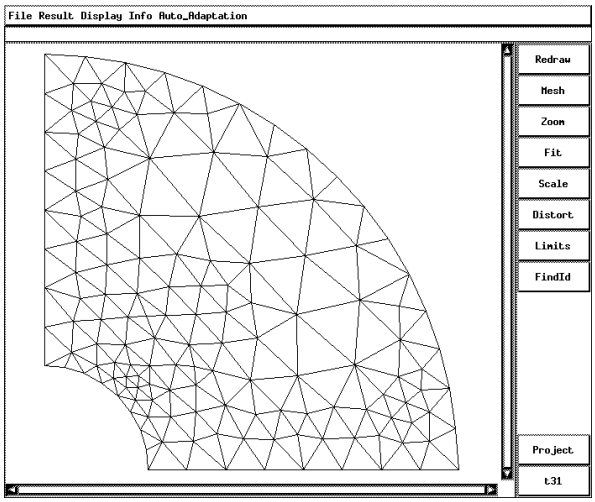
Figura 6.28: Cilindro com pressão interna.

Este exemplo é apresentado para demonstrar a eficácia da estratégia proposta neste trabalho no tratamento de curvas genéricas, no caso um arco de círculo, além das linhas retas já mostradas. Além disso, verifica-se o comportamento da estratégia em situações sem presença de singularidades. As malhas obtidas são mostradas nas figuras 6.29 e 6.30. O erro relativo máximo η^* desejado de 5% para elementos quadráticos e 10% para lineares foi obtido em um e em dois passos de auto-adaptação, respectivamente. Assim como Lee e Lo (1992), que também analisaram este problema, se iniciou de uma malha bem grosseira de apenas quatro elementos para se verificar a eficiência da estratégia, e ela se comportou muito bem, com o erro estimado caindo de 57.77% para 11.44% para elementos lineares só no primeiro passo e de 31.75% para 2.78% para elementos quadráticos. As figuras 6.31 e 6.34 mostram as taxas de convergência para o exemplo. Os gráficos do índice de efetividade são mostrados nas figuras 6.32 e 6.35. Como o valor para a energia de deformação analítica para o problema não é "exata", quando a norma $\|e\|$ calculada decresce para valores muito baixos, θ ultrapassa 1.0. Esta é a explicação para o ponto acima de 1.0 no caso do elemento quadrático. As figuras 6.33 e 6.36 mostram as taxas computacionais envolvidas.



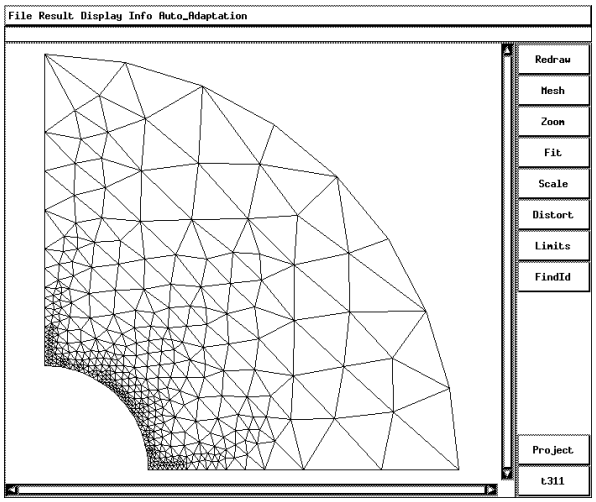
a - NN = 6, NE = 4

DOF = 8, $\overline{\eta} = 57.77\%$



b - NN = 149, NE = 256

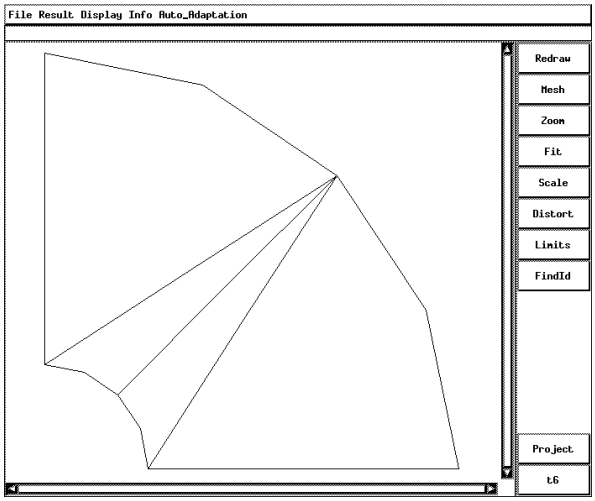
DOF = 280, $\overline{\eta} = 11.44\%$



c - NN = 444, NE = 811

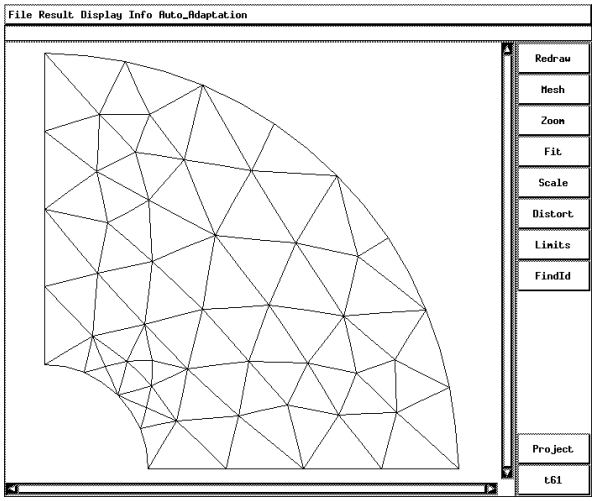
DOF = 851, $\overline{\eta} = 5.63\%$

Figura 6.29: Exemplo 3 com T3: malha inicial (a), malhas seguintes (b-c).



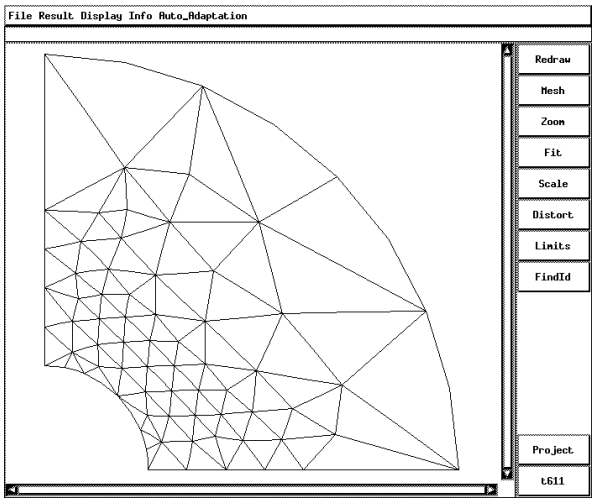
a - NN = 15, NE = 4

DOF = 24, $\overline{\eta} = 31.75\%$



b - NN = 189, NE = 84

DOF = 360, $\overline{\eta} = 2.78\%$



c - NN = 253, NE = 116

DOF = 484, $\overline{\eta} = 1.43\%$

Figura 6.30: Exemplo 3 com T6: malha inicial (a), malhas seguintes (b-c).

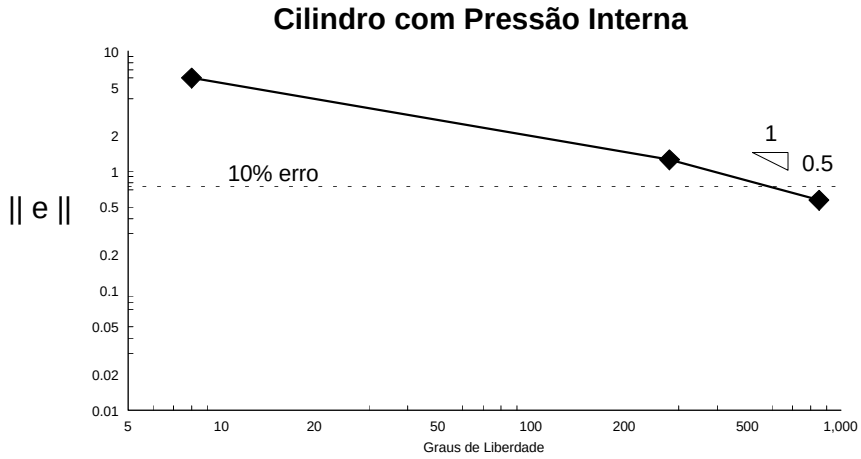


Figura 6.31: Taxa de convergência do exemplo 3 para T3.

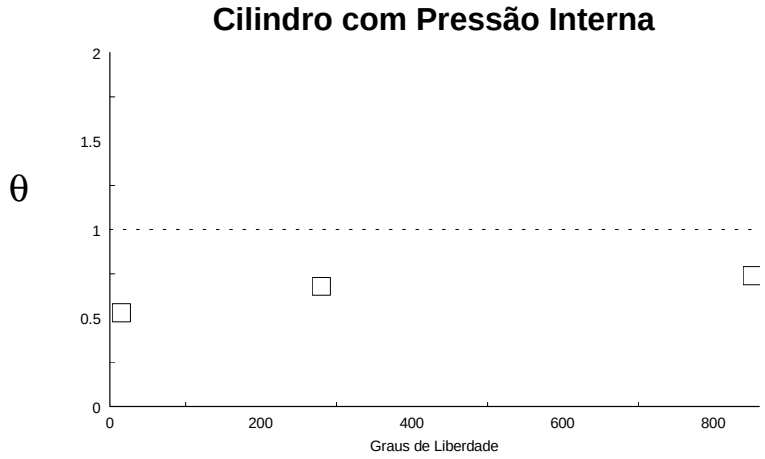


Figura 6.32: Índice de efetividade do exemplo 3 para T3.

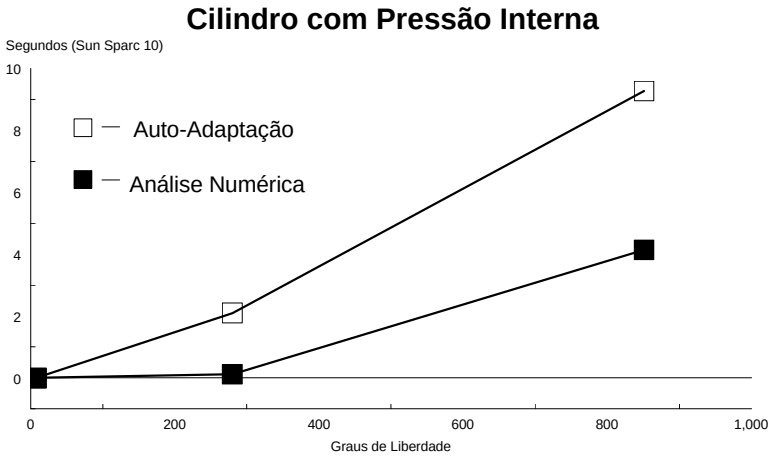


Figura 6.33: Tempo computacional do exemplo 3 para T3.

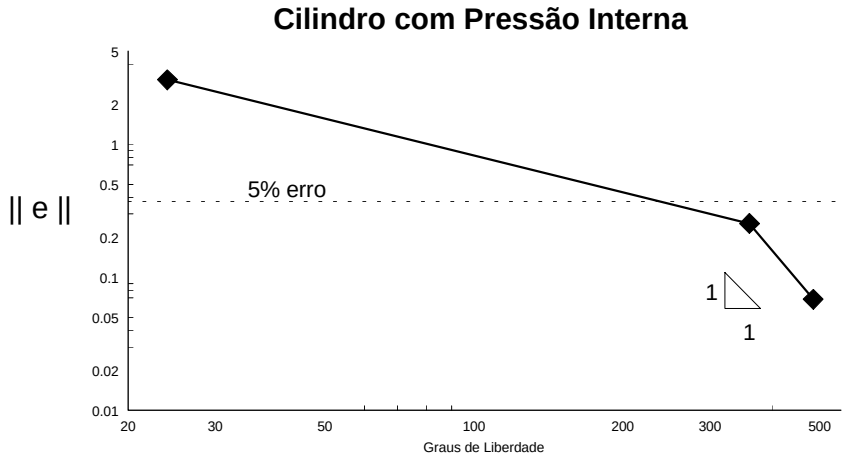


Figura 6.34: Taxa de convergência do exemplo 3 para T6.

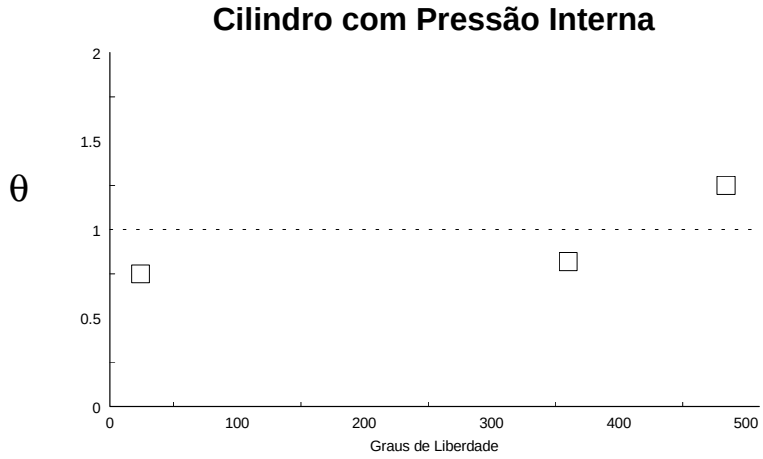


Figura 6.35: Índice de efetividade do exemplo 3 para T6.

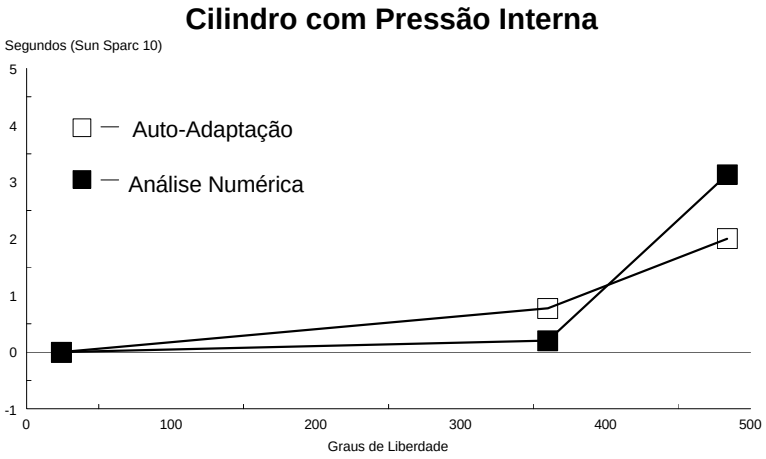


Figura 6.36: Tempo computacional do exemplo 3 para T6.

6.2.4 *Exemplo 4 - Placa com Furo com Dois Materiais*

O quarto exemplo é a mesma placa do exemplo 1, mas considerando dois materiais distintos no quarto de placa modelado. O objetivo deste exemplo é demonstrar a aplicabilidade da estratégia para regiões de materiais distintos. O módulo de elasticidade para uma região é o triplo do considerado para a outra, ou seja, tem-se $E_1 = 10^5$ e $E_2 = 3 \times 10^5$ com o coeficiente de Poisson mantido em 0.3. A energia de deformação calculada para o problema resulta em um valor para a norma $\|u\|^2$ de 0.20466. A figura 6.37 mostra o quarto de placa com os dois materiais considerados.

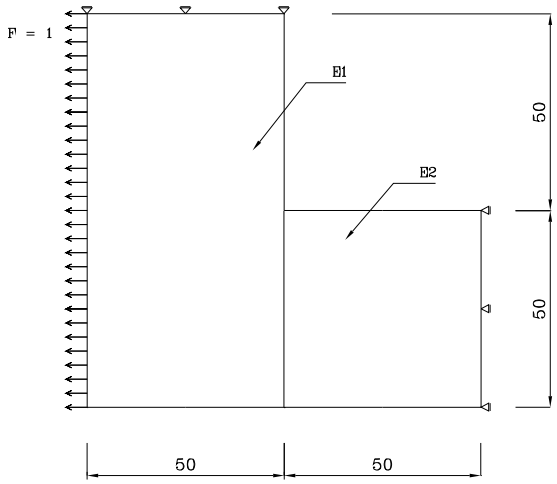


Figura 6.37: Placa com furo com dois materiais.

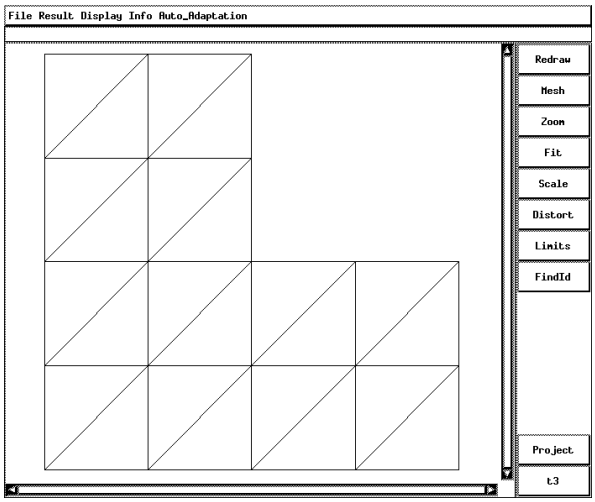
Para o erro relativo η^* máximo desejado de 10% para os elementos lineares e 5% para os quadráticos, foram precisos dois passos de adaptação para ambos os casos. As figuras 6.40 e 6.43 mostram as taxas de convergência e as figuras 6.41 e 6.44 os índices de efetividade para ambos os tipos de elementos. Esse problema foi analisado por Ainsworth et al. (1989) para elementos quadráticos e também foram necessários dois passos para a obtenção dos resultados esperados.

Para ilustrar a distribuição da norma de energia de erro $\|\overline{e}\|$ são mostradas, nas figuras 6.46 e 6.47, as distribuições de $\|\overline{e}\|$ para as última malhas usando elementos lineares e quadráticos. As figuras apresentam uma ampliação do canto reentrante do modelo. Verifica-se que a norma distribui-se de maneira praticamente uniforme pelo modelo, sendo que no canto reentrante se obtém valores superiores como era de se

esperar, indicando que um posssível prosseguimento na auto-adaptação refinaria mais aquela região.

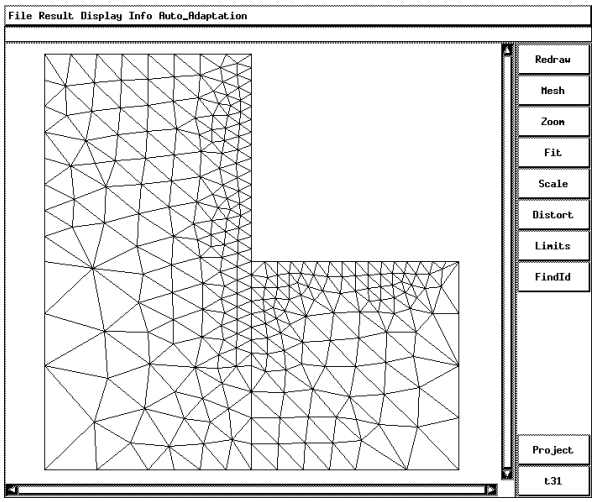
Os gráficos dos tempos computacionais são mostrados nas figuras 6.42 e 6.45. Obtiveram-se tempos um pouco superiores para os elementos lineares que o exemplo 1, porque para cada região é montada uma árvore diferente para geração da malha. Mesmo assim, isso não acarreta em tempos muito mais elevados na auto-adaptação, como se pode notar para os elementos quadráticos, por exemplo, em que se obteve tempos não muito superiores.

Este exemplo é importante também porque demonstra que a técnica de gerar uma árvore para cada região pode ser usada para fazer um refinamento local e não necessariamente se refazer a malha inteira como se faz atualmente, apesar disto não acarretar em um tempo muito maior de geração e provavelmente assegurar uma transição melhor na nova malha.



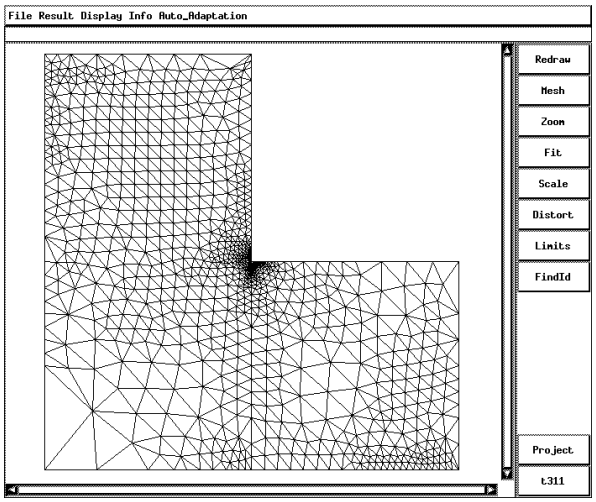
a - NN = 21, NE = 24

DOF = 36, $\overline{\eta} = 35.76\%$



b - NN = 286, NE = 503

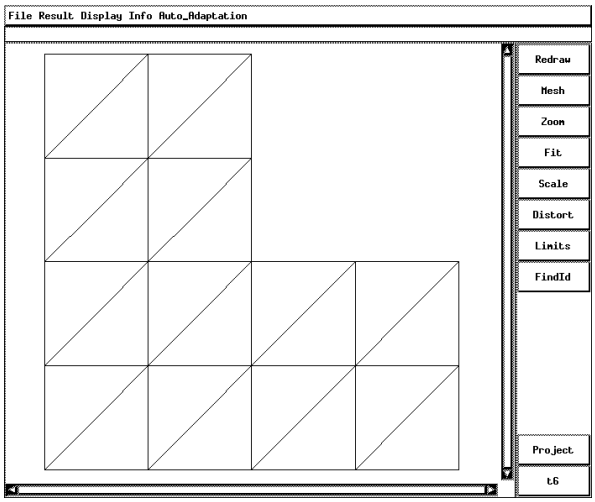
DOF = 558, $\overline{\eta} = 14.41\%$



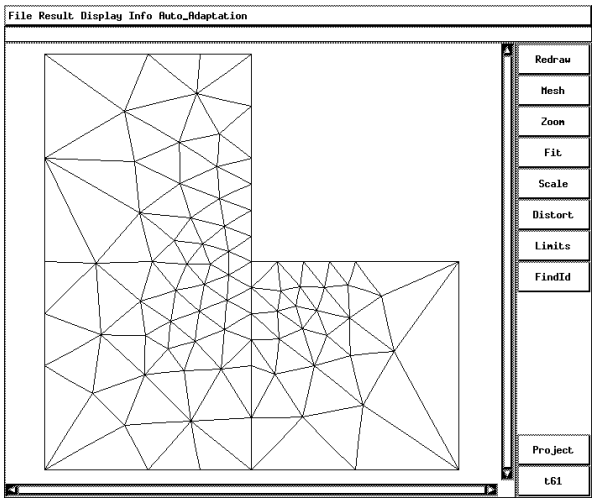
c - NN = 1077, NE = 1999

DOF = 2135, $\overline{\eta} = 7.33\%$

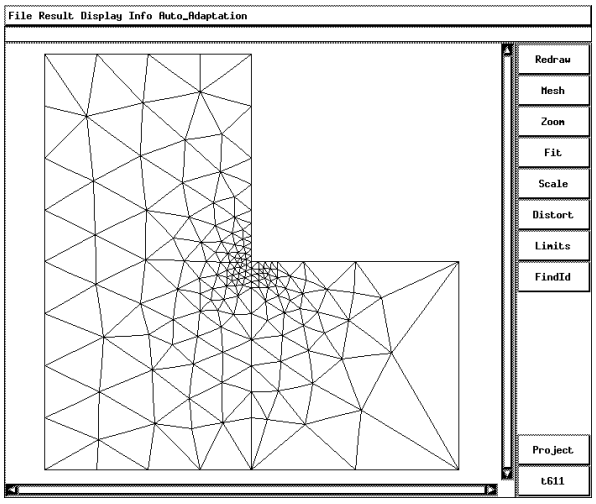
Figura 6.38: Exemplo 4 com T3: malha inicial (a), malhas seguintes (b-c).



a - NN = 65, NE = 24
DOF = 120, $\overline{\eta} = 19.37\%$



b - NN = 308, NE = 141
DOF = 606, $\overline{\eta} = 6.57\%$



c - NN = 629, NE = 296
DOF = 1246, $\overline{\eta} = 3.26\%$

Figura 6.39: Exemplo 4 com T6: malha inicial (a), malhas seguintes (b-c).

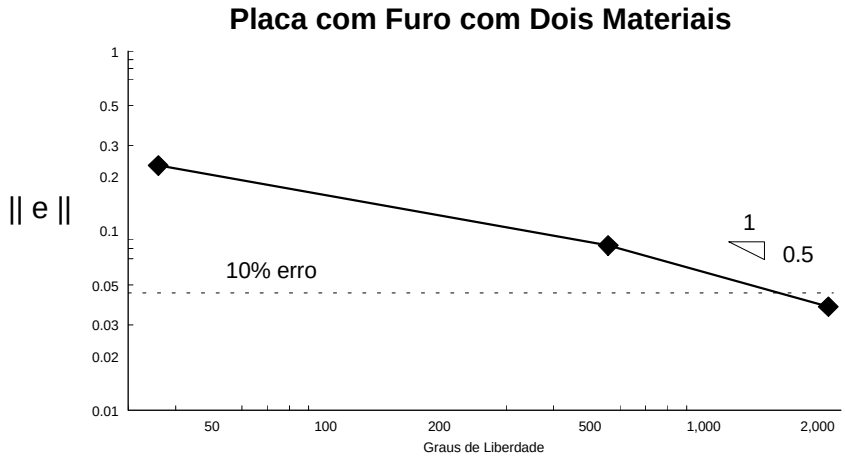


Figura 6.40: Taxa de convergência do exemplo 4 para T3.

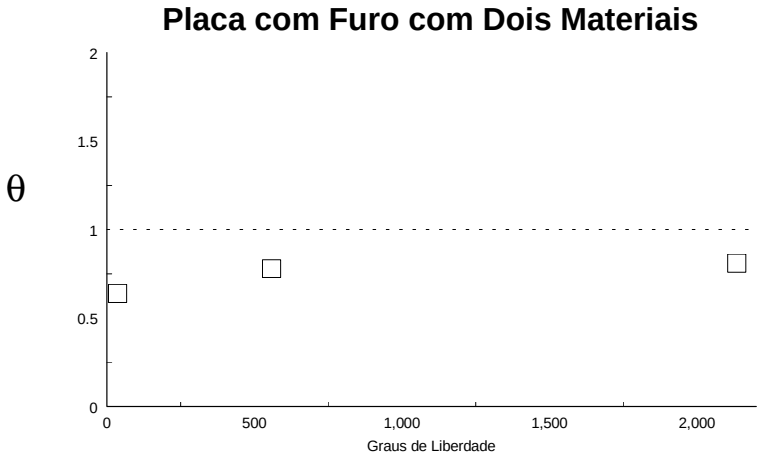


Figura 6.41: Índice de efetividade do exemplo 4 para T3.

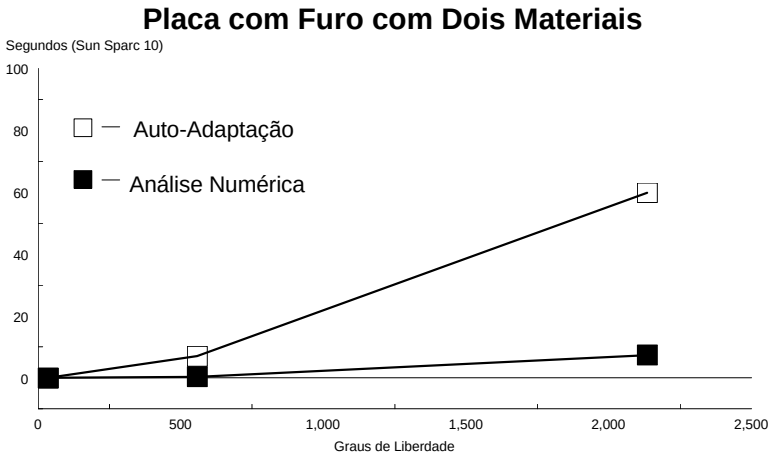


Figura 6.42: Tempo computacional do exemplo 4 para T3.

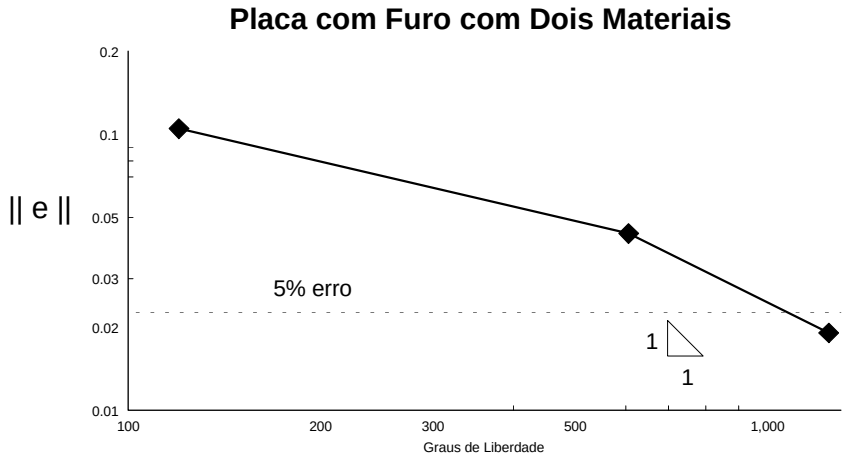


Figura 6.43: Taxa de convergência do exemplo 4 para T6.

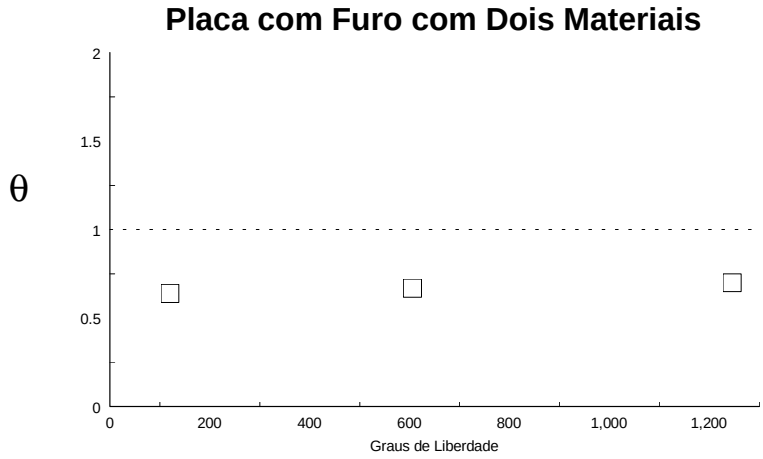


Figura 6.44: Índice de efetividade do exemplo 4 para T6.

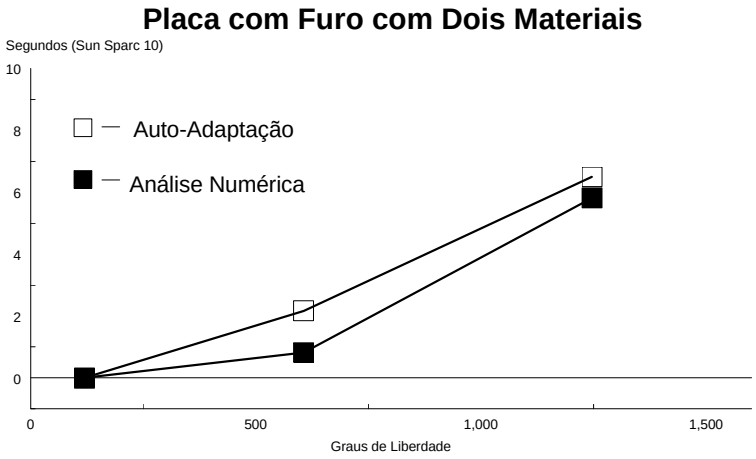


Figura 6.45: Tempo computacional do exemplo 4 para T6.

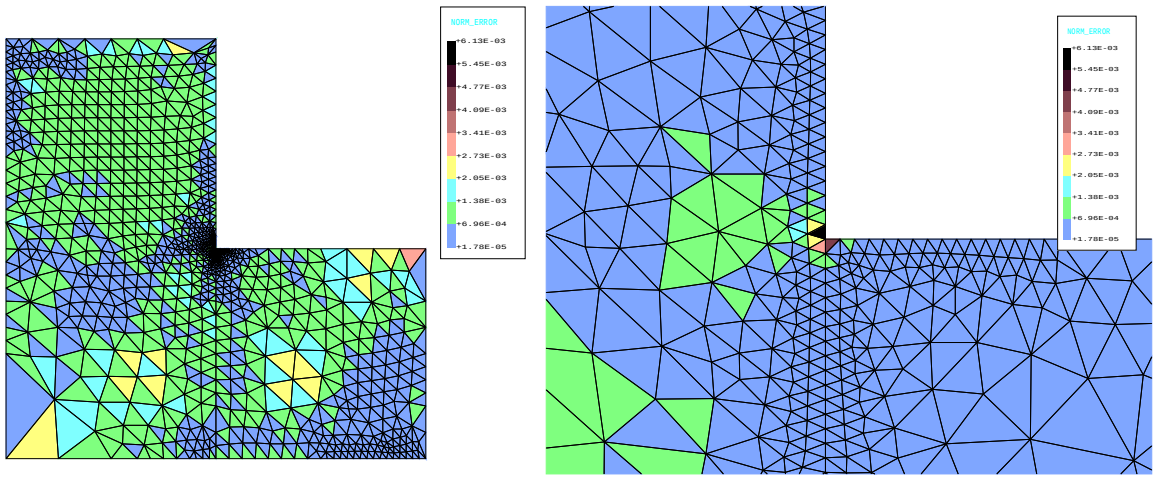


Figura 6.46: Norma de energia para o erro na última malha do exemplo 4 para o T3.

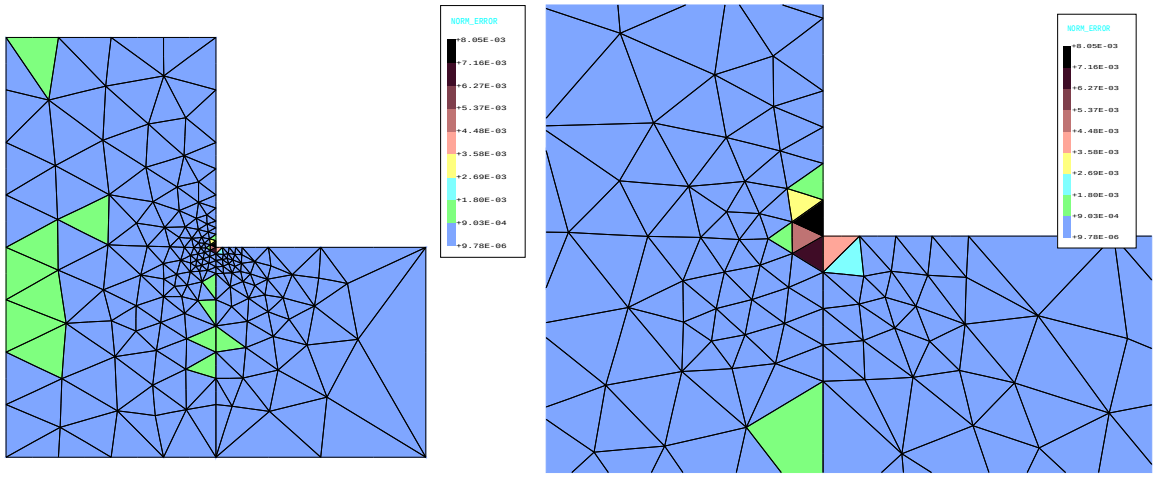


Figura 6.47: Norma de energia para o erro na última malha do exemplo 4 para o T6.

6.2.5 *Exemplo 5 - Placa Complexa com Vários Furos*

O quinto exemplo é a análise de uma placa sob um carregamento lateral unitário como mostrado na figura 6.48. Para a análise, é assumido estado plano de deformação com $\nu = 0.3$ e $E = 10^5$.

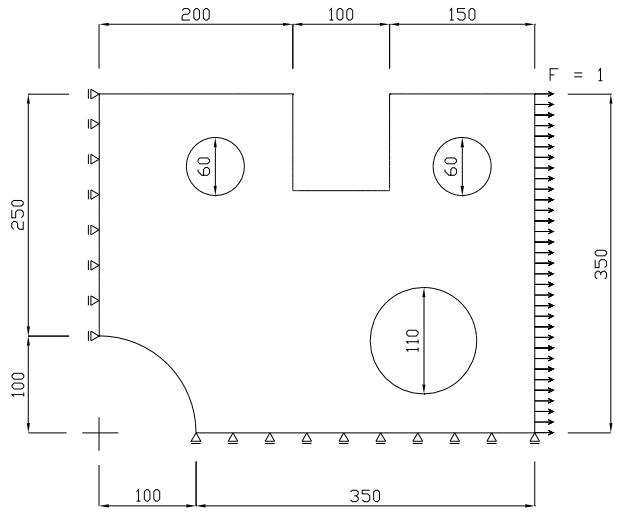
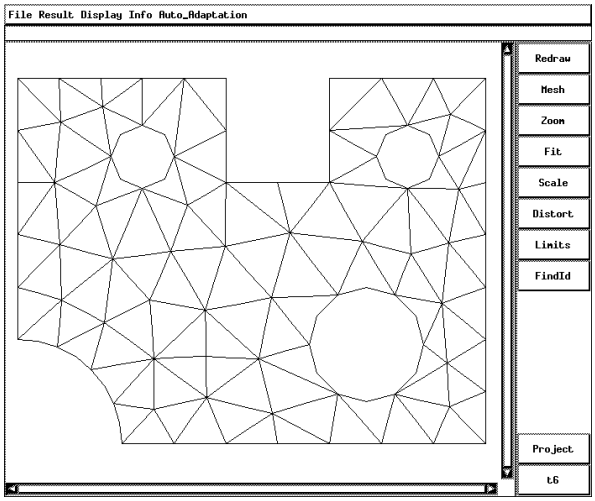


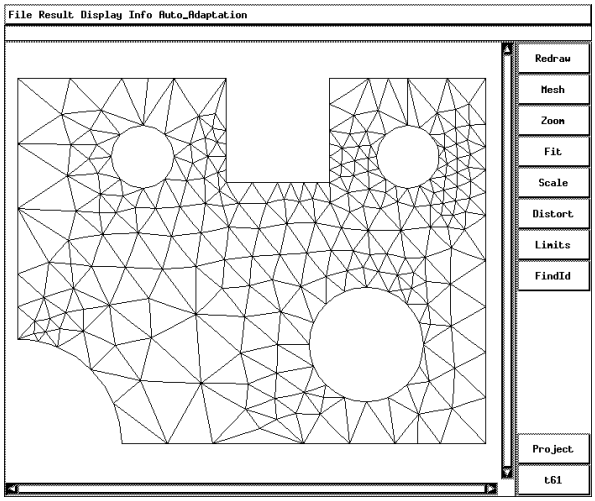
Figura 6.48: Placa complexa com vários furos.

Este exemplo foi incluído para demonstar a eficácia da estratégia para problemas de geometria complexa, com vários furos. Este exemplo só foi analisado para elementos quadráticos, com o máximo erro relativo permitido de 5%. Foram necessários dois passos de auto-adaptação para se atingir este erro, sendo a análise levada um passo adiante para se verificar o comportamento posterior ao atingir esse erro, e parou na malha seguinte onde se obteve um erro para $\overline{\eta}$ de 1.91%. A figura 6.49 mostra as malhas obtidas neste processo. Como não se conhece a solução analítica para a energia de deformação da placa, plotou-se apenas o gráfico de $\overline{\eta}$ por graus de liberdade para o modelo, para se verificar a convergência do mesmo (figura 6.50).

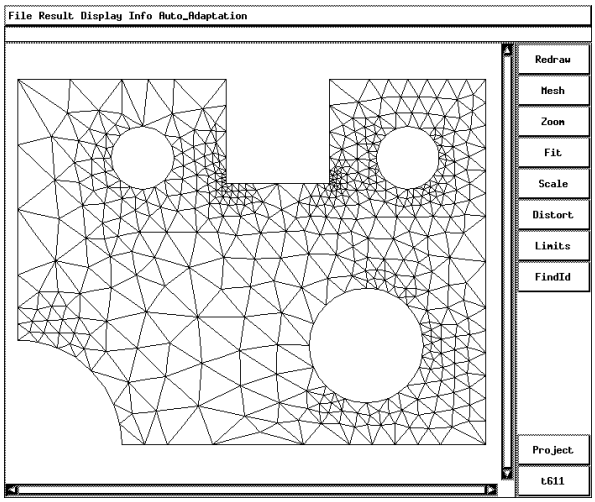
Baehmann (1989) analisou exemplo semelhante, e após quatro passos de auto-adaptação, atingiu em torno de 2% de erro. Vale ressaltar que não se conhece as dimensões da placa no seu trabalho, mas somente a forma da geometria que aqui foi reproduzida, prejudicando um pouco a comparação dos resultados feitos. O tempo computacional é mostrado na figura 6.51, onde se verifica que mesmo para este problema complexo, foram atingidos tempos bem rápidos para a auto-adaptação.



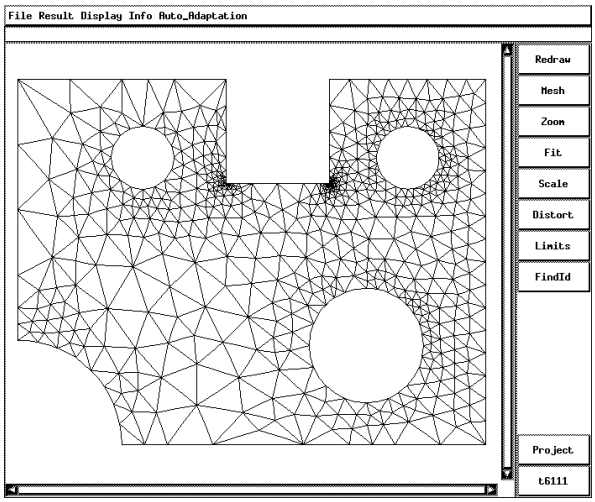
a - NN = 275, NE = 113
DOF = 524, $\overline{\eta} = 12.47\%$



b - NN = 953, NE = 431
DOF = 1878, $\overline{\eta} = 5.71\%$



c - NN = 1979, NE = 923
DOF = 3924, $\overline{\eta} = 2.83\%$



d - NN = 2476, NE = 1162
DOF = 4920, $\overline{\eta} = 1.91\%$

Figura 6.49: Exemplo 5 com T6: malha inicial (a), malhas seguinte (b-d).

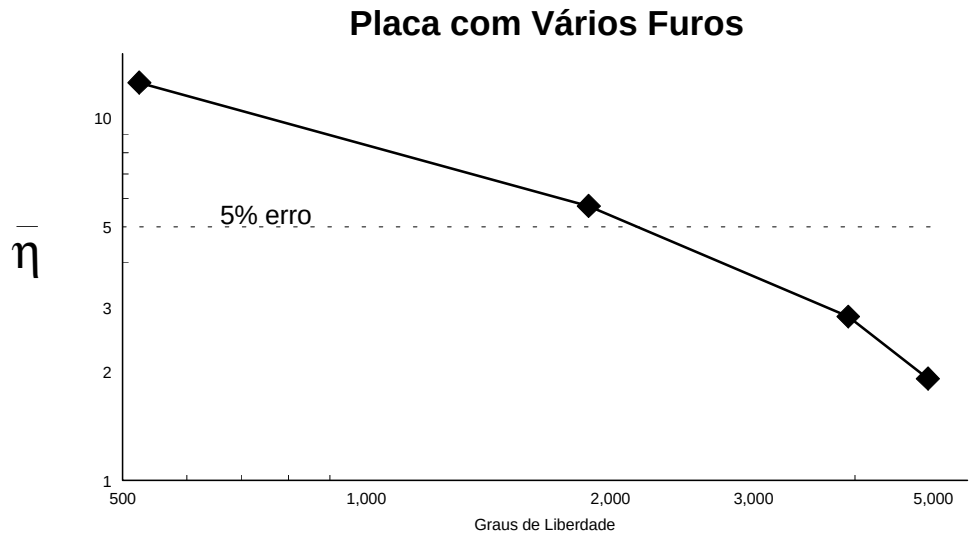


Figura 6.50: Taxa de convergência do exemplo 5 para T6.

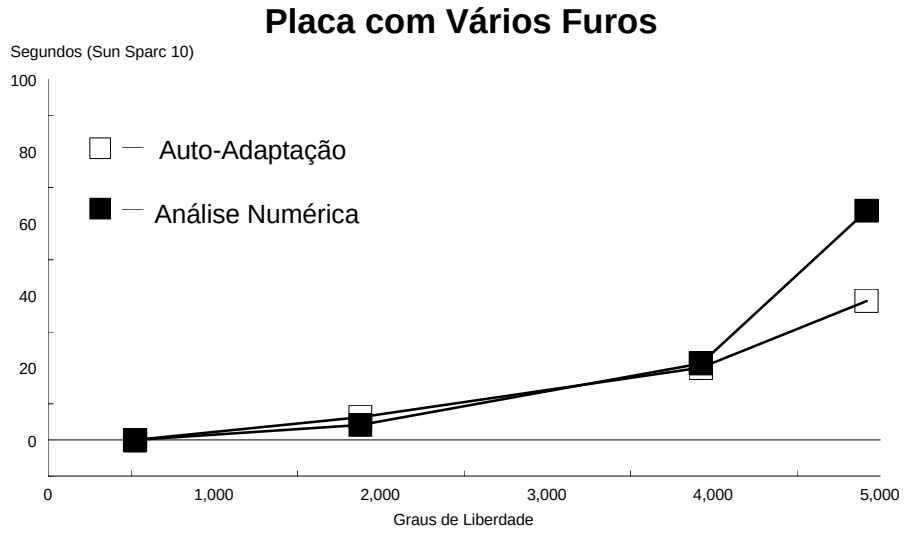


Figura 6.51: Tempo computacional do exemplo 5 para T6.

7

Conclusão

Este trabalho descreve um ambiente computacional gráfico interativo para análise integrada e auto-adaptativa de modelos bidimensionais de elementos finitos, com aplicação em problemas estruturais e mecânicos. O sistema é *integrado* porque envolve um modelador geométrico para criar a geometria do modelo, um pré-processador para geração da malha e aplicação de atributos, um módulo de análise para avaliar a solução do modelo e um pós-processador para vizualização dos resultados. O sistema integrado é *auto-adaptativo* porque, além de envolver todos esses conceitos, também tem a capacidade de, feita a criação do modelo com uma malha inicial e seus atributos, decidir onde refinar a malha, refazer a análise e repetir o procedimento até que um critério de convergência pré-estipulado seja atingido. A estratégia proposta envolve muito mais que simplesmente uma técnica de geração auto-adaptativa de malhas: também é considerada a maneira como o sistema gerencia o processo auto-adaptativo como um todo e como ele redistribui os atributos físicos e geométricos do modelo na nova malha criada. Além disso, a estratégia de auto-adaptação proposta é confiável, robusta e suficientemente rápida para possibilitar uma simulação interativa de problemas práticos de engenharia. Acredita-se que este trabalho é um passo decisivo na direção de uma estratégia de automação real em modelagem por elementos finitos.

7.1 *Principais Contribuições*

Tem-se pesquisado e proposto na literatura várias técnicas de auto-adaptação de malhas com características de robustez, rapidez e confiabilidade. A técnica de *Enumeração Espacial Recursiva*, através do uso de árvores quaternárias (*quadtree)* para modelos bidimensionais, se mostra como uma opção capaz de atender esses requisitos. Entretanto, o seu uso na forma original, como proposto em alguns trabalhos, muitas vezes gera malhas com elementos de forma ruim perto das fronteiras do modelo, já que a discretização das curvas do contorno é definida pela atuação da *quadtree* utilizada para discretizar o domínio. Dentro desse escopo, a principal contribuição deste trabalho é

uma estratégia de refinamento regular das curvas de fronteira, e entre regiões de materiais distintos, independente da discretização do domínio. O algoritmo de geração da malha no domínio parte da discretização do contorno e se conforma com esse refinamento. Tanto o refinamento do contorno como o do domínio são baseados na técnica de enumeração espacial recursiva. A discretização do contorno envolve uma estrutura de dados de árvore binária (*binary tree*) e a geração da malha no domínio é baseda na combinação das técnicas de *quadtree* e triangulação de Delaunay por contração de contorno, utilizando as informações de adjacência existentes na *quadtree* para otimização do algoritmo. O resultado desta estratégia é a manutenção das características de robustez, rapidez e confiabilidade da técnica de *quadtree* aliada a um refinamento regular e com boa transição da malha perto do contorno do modelo.

Além disso, criou-se uma plataforma que permite uma fácil expansão da estratégia para outros tipos de análise existentes. A disciplina de programação orientada a objetos usada neste trabalho se mostrou de fundamental importância neste aspecto. Ela permite a aplicação da estratégia a problemas genéricos no que diz respeito, por exemplo, ao tipo de elemento usado e ao tipo de geometria das curvas de contorno.

Foi desenvolvido também um sistema gráfico integrado, baseado em uma estratégia de janelas múltiplas, para análise e visualização de resultados. A plataforma utilizada (o sistema gráfico GKS/puc e o sistema de interface IUP/LED) permite a portabilidade do sistema para vários ambientes, como micro-computadores compatíveis com a linha IBM-PC e estações de trabalho Sun Sparcstation, IBM RS 6000.

7.2 *Sugestões para Trabalhos Futuros*

A estratégia proposta está calcada na existência de uma malha inicial juntamente com um arquivo de descrição de regiões, suas curvas e seus atributos. Este arquivo e a malha do modelo são gerados por um pré-processador existente (Mtool, 1992), mas o sistema funciona de forma independente, já que essa descrição é feita em formato padrão estabelecido (arquivo neutro para a malha e arquivo de descrição dos atributos). Uma sugestão para trabalho futuro é a inclusão dos recursos de modelagem geométrica do pré-processador no módulo auto-adaptativo. Além disso, sugere-se a inclusão de uma metodologia de especificação de atributos configurável pelo usuário dentro desse módulo, o que permitiria ao usuário definir ele próprio que atributos devem ser considerados na análise, que poderiam variar dependendo do problema em questão. Isso

seria feito através do uso da linguagem Lua desenvolvida dentro do mesmo grupo de pesquisa (Figueiredo et al., 1994). Tudo isso permitiria ao usuário trabalhar em um único ambiente, o módulo auto-adaptativo, sem necessitar do uso do pré-processador (a malha inicial também seria gerada dentro do módulo auto-adaptativo).

Outra sugestão é a expansão do sistema para abordar outros tipos de análises, não somente a linear elástica, baseada no paradigma de programação orientada a objetos. Essa expansão viria de encontro à sugestão de implementação de uma linguagem formal orientada a objetos, para todo o sistema, incluindo os módulos de análise e de auto-adaptação.

Verificou-se nesse trabalho que, apesar do estimador de erro utilizado apresentar um baixo grau de precisão, a estratégia de auto-adaptação proposta tende a compensar isso pois as malhas geradas apresentam uma distribuição uniforme do erro estimado. Uma busca por estimadores mais eficientes ou a melhoria do estimador usado, como por exemplo a implementação de um processo super-convergente de recuperação das tensões analíticas (Zienkiewicz, 1992a), e a consideração do efeito de concentração de tensão pelo estimador, seriam interessantes pois uma estimativa de erro mais eficiente geraria soluções ainda mais eficazes.

Na estratégia de auto-adaptação da malha dois aspectos são relevantes e podem ser melhorados. No uso da árvore binária para o refinamento das fronteiras, deve-se considerar o reconhecimento automático de características (*features*) geométricas das curvas para evitar em alguns casos um "desrefinamento" excessivo no bordo. Na geração da malha por *quadtree*, a fase de ajustes devido ao erro numérico dos elementos deve ser feita não apenas considerando cada elemento isoladamente, mas a influência de um conjunto de elementos adjacentes a cada elemento, já que o objetivo da malha "ótima" é se obter uma malha com o erro igualmente distribuído entre todos os elementos da malha do modelo. Ainda nesse algoritmo de geração da malha por *quadtree*, sugere-se sua complementação para geração de malha também utilizando elementos quadrilaterais.

A estratégia proposta baseada no uso da *quadtree* permite a implementação de um refinamento localizado da malha, ao invés de gerar uma malha global a cada passo. Sugere-se a implementação dessa metodologia a fim de se avaliar o desempenho do algoritmo e a qualidade da malha auto-adaptada com relação ao enfoque atual.

Finalmente sugere-se a expansão da estratégia para aplicação para problemas tridimensionais, pelo uso de enumeração espacial recursiva baseadas em *octree*.

Apêndice A

Formato do arquivo de Descrição das Condições de Contorno das Curvas e Atributos das Regiões

O arquivo de descrição segue as seguintes convenções:

region (:r)

line (:l)

polyline (:p)

arc (:a)

circle (:c)

bezier (:b)

A.1 *Arquivo de Descrição*

:r nl 'string region attribute'

1 nc orientation_1 id_1 . . . orientation_nc id_nc

2 nc orientation_1 id_1 . . . orientation_nc id_nc

.

.

nl nc orientation_1 id_1 . . . orientation_nc id_nc

:l x1 y1 x2 y2 'string curve attribute'

:p n 'string curve atribute'

x1 y1

x2 y2

.

.

xn yn

:a cx cy px1 py1 px2 py2 'string curve attribute'

:c cx cy px py 'string curve attribute'

:b px1 py1 cx1 cy1 cx2 cy2 px2 py2 'string curve attribute'

---

nl = número de ciclos (loops) em uma região, nc = número de curvas em um ciclo

Referências Bibliográficas

(Ainsworth et al., 1989)

Ainsworth, M.; Zhu, J.Z.; Craig, A.W. e Zienkiewcz, O.C. - "Analysis of the Zienkiewicz-Zhu *a-posteriori* Error Estimator in the Finite Element Method," *Int. J. Num. Meth. Engng.*, vol.28, pp. 2161-2174, 1989.

(Babuska, Rheimboldt, 1976)

Babuska, I. e Rheimboldt, W.C. - "Analysis of Optimal Finite Element Meshes in R^1," *Mathematics of Computations*, vol. 33, no. 146, pp. 435-463, 1976.

(Babuska, Rheimboldt, 1978)

Babuska, I. e Rheimboldt, W.C. - "*A-posteriori* Error Estimates for Finite Element Problems," *Int. J. Num. Meth. Engng.*, vol. 12, pp. 1597-1615, 1978.

(Babuska, Rheimboldt, 1979)

Babuska, I. e Rheimboldt, W.C. - "Adaptive Approaches and Reliability Estimation in Finite Element Analysis," *Comp. Meth. in Applied Mec. and Engn.*, vol. 18, pp. 519-540, 1979.

(Babuska, Szabó, 1991)

Babuska, I. e Szabó, B. - *Finite Element Analysis*, Jonh Wiley & Sons, 1991.

(Baehmann et al., 1987)

Baehmann, P.L., Wittchen, S.L., Shephard, M.S., et al. - "Robust Geometrically Based, Automatic Two-Dimensional Mesh Generation," *Int. J. Num. Meth. Engng.*, vol. 24, pp. 1043-1078, 1987.

(Baehmann, Shephard, 1989)

Baehmann, P.L. e Shephard, M.S. - "Adaptive Multiple-Level h-Refinement in Automated Finite Element Analysis," *Engng. with Comput.*, vol. 5, pp. 235-247, 1989.

(Baehmann, 1989)

Baehmann, P.L. - "Automated Finite Element Modeling e Simulation," Ph. D. Thesis, Rensselaer Polytechnic Institute, Troy - New York, 1989.

(Barbalho, 1994)

Barbalho, V.M.S. - "Triangulação Adaptativa de Sólidos CSG para Geração de Malhas de Elementos Finitos," Dissertação de Mestrado, COPPE/UFRJ, 1994.

(Barros, 1994)

Barros, J.C.P. - "Análise Térmica pelo Método dos Elementos Finitos - Uma Filosofia Orientada a Objetos," Dissertação de Mestrado, Departamento de Engenharia Civil, PUC-Rio, 1994.

(Cavalcante et al., 1993a)

Cavalcante Neto, J.B., Carvalho, M.T.M. e Martha, L.F. - "Combinação das Técnicas de Quadtree e Delaunay para Geração Automática de Malhas de Elementos Finitos," anais do VI SIBGRAPI, Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens, Recife - PE, pp. 285-291, 1993.

(Cavalcante et al., 1993b)

Cavalcante Neto, J.B., Carvalho, M.T.M. e Martha, L.F. - "Geração Auto-Adaptativa de Malhas de Elementos Finitos Utilizando uma Técnica de Enumeração Espacial Recursiva," anais do XIV CILAMCE, Congresso Ibero-Latino Americano sobre Métodos Computacionais para Engenharia, São Paulo - SP, pp. 1285-1294, 1993.

(Chew, 1989)

Chew, L.P. - "Constrained Delaunay Triangulation," *Algorithmica*, vol. 4, pp. 97-108, 1989.

(Cook, 1989)

Cook, R.D.; Malkus, D.S.; Plesha, M.E. - *Concepts e Applications of Finite Element Analysis*, John Wiley, New York - NY, 1989.

(Cox, 1991)

Cox, B.J.; Novobilsk, A.J. - *Object Oriented Programming - An Evolutionary Approach*, Addison-Wesley, 2nd. edition, Reading - MA, 1991.

(Cuthill, McKee, 1969)

Cuthill, E. e McKee, J. - "Reducing the Bandwidth of Sparse Symmetric Matrices," *ACM Proceedings of 24th National Conference*, pp. 157-172, 1969.

(Cyrino, 1989)

Cyrino, J.C.R. - "Convergência Acelerada pela Relocalização de Nós e Refinamento de Malhas de Elementos Finitos," Tese de Doutorado, COPPE/UFRJ, 1989.

(De Florani, Puppo, 1992)

De Florani, L. e Puppo E. - "An On-line Algorithm for Constrained Delaunay Triangulation," *Computer Vision, Graphics, e Image Processing*, vol. 54, pp. 290-300, 1992.

(Fenves, 1990)

Fenves,G.L. - "Object-Oriented Programming for Engineering Software Development", *Engineering with Computers*, 6, pp. 1-15.

(Field, 1986)

Field, D.A. - "Implementing Watson's Algoritm in Three Dimensions", Proc. 2nd ACM Symposium on Computational Geometry, pp. 246-259, 1986.

(Figueiredo et al., 1994)

Figueiredo, L.H.; Ierusalimschy, R. e Celes Filho, W. - "The Design and Implementation of a Language for Extending Applications," XXI Semish, Seminário Integrado de Software e Hardware, pp. 273-284, 1994.

(Guimarães, 1992)

Guimarães, L.G.S. - "Disciplina de Programação Orientada a Objetos para Análise e Visualização Bidimensional de Modelos de Elementos Finitos", Dissertação de Mestrado, Departamento de Engenharia Civil, PUC-Rio, 1992.

(Guimarães et al., 1992)

Guimarães, L.G.S., Menezes, I.F.M. e Martha, L.F. - "Disciplina de Programação Orientada a Objetos para Sistemas de Elementos Finitos," anais do XIII CILAMCE, Congresso Ibero-Latino Americano sobre Métodos Computacionais para Engenharia, Porto Alegre - RS, vol. 1, pp. 342-351, 1992.

(Haber et al., 1981)

Haber, R., Shephard, M.S., Abel, J.F., Gallagher, R.H., e Greenberg, D.P. - "A General Two-Dimensional, Graphical Finite Element Pre-processor Utilizing Discrete Transfinite Mappings," *Int. J. Num. Meth. Engng.*, **17**, pp. 1015-1044, 1981.

(Hinton, Campbell, 1974)

Hinton, E. e Campbell, J.S. - "Local and Global Smoothing of Discontinuous Finite Element Functions Using a Least Squares Method," *Int. J. Num. Meth. Engng.*, vol. 8, pp. 461-480, 1974.

(Joe, 1986)

Joe, B. - "Delaunay Triangular Meshes in Convex Polygons," *SIAM J. Sci. Stat. Comput.*, vol. 7, pp. 514-539, 1986.

(Las Casas, 1990)

Las Casas, E.B. - "Um Processo Adaptativo Misto Global para o Método dos Elementos Finitos," *RBE - Revista Brasileira de Engenharia - Caderno de Eng. Estrutural*, vol. 6, no. 2, pp. 41-51, 1990.

(Las Casas, 1991)

Las Casas, E. B. - "Mixed Adaptive Methods for Finite Elements," *RBCM - Revista Brasileira de Ciências Mecânicas*, vol XIII, pp. 205-216, 1991.

(Lee, Lo, 1992)

Lee, C.K. e Lo, S.H. - "An Automatic Adaptative Refinement Finite Element Procedure for 2D Elastostatic Analysis," *Int. J. Num. Meth. Engng.*, vol.35, pp. 1967-1989, 1992.

(Levy, 1993)

Levy, C.H. - "IUP/LED: Uma Ferramenta Portátil de Interface com Usuário," Dissertação de Mestrado, Departamento de Informática, PUC-Rio, 1993.

(Lo, 1989)

Lo, S.H. - "Delaunay Triangulation of Non-Convex Planar Domains," *Int. J. Num. Meth. Engng.*, vol. 28, pp. 2695-2707, 1989.

(Mtool, 1992)

Mtool - Bidimensional Mesh Tool, Manual do Usuário, versão 1.0, TeCGraf/PUC-Rio, 1992.

(Mview, 1993)

Mview - Bidimensional Mesh View, Manual do Usuário, versão 1.1, TeCGraf/PUC-Rio, 1993.

(Peraire et al., 1987)

Peraire J.; Vahdati, M.; Morgan K. e Zienkiewicz, O.C. - "Adaptive Remeshing for Compressive Flow Computations," *Journal of Computational Physics*, vol. 72, pp. 449-466, 1987.

(Potyondy, 1993)

Potyondy, D.O. - "A Software Framework for Simulating Curvilinear Crack Growth in Pressurized Thin Shells," Ph.D. Thesis, School of Civil Engineering, Cornell University, 1993.

(Ribeiro, 1991)

Ribeiro, L.F.B. - "Estratégia H-P de Refinamento para o Método de Elementos Finitos, " Tese de Doutorado, UFRJ, 1991.

(Samet, 1984)

Samet, H. - "The Quadtree e Related Hierarchical Data Structures," *ACM Computer Surveys*, vol. 16, no. 2, 1984.

(Samet, 1990)

Samet, H. - *Applications of Spatial Data Structures - Computer Graphics, Image Processing and GIS*, Addison-Wesley, 1st. edition, Reading - MA, 1990.

(Santana, 1993)

Santana, W.C. - "Método de Elementos Finitos h-Adaptativo para Análise de Problemas Elásticos Planos," Dissertação de Mestrado, Departamento de Engenharia Mecânica, PUC-Rio, 1993.

(Shaw, Pitchen, 1978)

Shaw, R.D. e Pitchen, R.G. - "Modifications to the Suhara-Fukuda Method of Network Generation," *Int. J. Num. Meth. Engng.*, vol. 12, pp. 93-99, 1978.

(Shephard, 1986)

Shephard, M.S. - "Adaptive Finite Elements e CAD, Accuracy Estimates and Adaptive Refinements in Finite Element Computations," I. Babuska et al. Edited, pp. 205-225, John Wiley & Sons, 1986.

(TeCGraf, 1989)

TeCGraf - Manual de Utilização do GKS/puc, versão 3.0, PUC-Rio, 1989.

(Vianna, 1992)

Vianna, A. C. - "Modelagem Geométrica completa para modelos bidimensionais de elementos finitos ", Dissertação de Mestrado, Departamento de Engenharia Civil, PUC-Rio, 1992.

(Von Herzen, Barr, 1987)

Von Herzen B. e Barr A. H. - "Accurate Triangulations of Deformed, Intersecting Surfaces", *Computer Graphics*, vol. 21, pp. 103-110, 1987.

(Watson, 1981)

Watson, D.F. - "Computing the n-dimensional Delaunay tesselation with application to Voronoi polytopes", *The Computer Journal*, vol. 24, no. 2, pp. 167-172, 1981.

(Yerry, Shephard, 1984)

Yerry, M.A. e Shephard, M.S. - "Automatic Three-Dimensional Mesh Generation by Modified-Octree Technique," *Int. J. Num. Meth. Engng.*, vol. 20, pp. 1965-1990, 1984.

(Zienkiewicz et al., 1982)

Zienkiewicz, O.C.; Kelly, D.W.; Gago, J. e Babuska, I. - "Hierarchical Finite Element Approaches, Error Estimador and Adaptive Refinement," in J. Whiteman (ed.), *Mathematics of Finite Elements and Applications (IV)*, Academic Press, New York - NY, 1982.

(Zienkiewicz, Zhu, 1987)

Zienkiewicz, O.C. e Zhu, J.Z. - "A Simple Error Estimator and Adaptive Procedure for Practical Engineering Analysis," *Int. J. Num. Meth. Engng.*, vol. 24, pp. 337-357, 1987.

(Zienkiewicz, 1989)

Zienkiewicz, O.C. e Taylor, R.L. - *The Finite Element Method - Volume 1, Basic Formulation and Linear Problems*, McGraw-Hill, New York - NY, 1989.

(Zienkiewicz et al., 1989)

Zienkiewicz, O.C.; Zhu, J.Z. e Gong, N.G. - "Effective and Pratical h-p-Version Adaptive Procedures for the Finite Element Method," *Int. J. Num. Meth. Engng.*, vol. 28, pp. 879-891, 1989.

(Zienkiewicz, Zhu, 1990)

Zienkiewicz, O.C. e Zhu, J.Z. - "The Three R's of Engineering Analysis an Error Estimation and Adaptivity," *Comp. Meth. in Appl. Mec. and Engng.*, vol. 28, pp. 879-891, 1989.

(Zienkiewicz, Zhu, 1992a)

Zienkiewicz, O.C. e Zhu, J.Z. - "The Superconvergent Patch Recovery and A Posteriori Error Estimates. Part 1: The Recovery Technique," *Int. J. Num. Meth. Engng.*, vol. 33, pp. 1331-1364, 1992.

(Zienkiewicz, Zhu, 1992b)

Zienkiewicz, O.C. e Zhu, J.Z. - "The Superconvergent Patch Recovery and A Posteriori Error Estimates. Part 2: Error Estimates e Adaptivity," *Int. J. Num. Meth. Engng.*, vol. 33, pp. 1365-1382, 1992.