PUC

LUIZ GIL SOLON GUIMARÃES

DISCIPLINA DE PROGRAMAÇÃO ORIENTADA A OBJETOS PARA ANÁLISE E VISUALIZAÇÃO BIDIMENSIONAL DE MODELOS DE ELEMENTOS FINITOS

DISSERTAÇÃO DE MESTRADO

Departamento de Engenharia Civil
PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

Rio de Janeiro, Agosto de 1992

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 – CEP 22453
RIO DE JANEIRO – BRASIL

Luiz Gil Solon Guimarães

DISCIPLINA DE PROGRAMAÇÃO ORIENTADA À OBJETOS PARA ANÁLISE E VISUALIZAÇÃO BIDIMENSIONAL DE MODELOS 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, 31 de Agosto de 1992

Agradecimentos

Aos meus pais, por tudo.

À Patrícia, minha esposa, pelo apoio e compreensão nas horas de ausência.

Ao professor e amigo Luiz Fernando pela orientação, apoio, confiança, incentivo e ensinamentos recebidos durante o desenvolvimento deste trabalho.

À todos amigos da PUC e em especial à Anselmo, Arlindo, Eduardo, João Batista e Remo pela grande amizade, apoio técnico e pessoal.

Aos amigos Ivan e Anselmo pela amizade e desenvolvimento conjunto ao longo deste trabalho.

Aos professores e funcionários do departamento e em especial ao Professor Marcelo Gattass que, indiretamente, colaborou na orientação deste trabalho.

Ao ICAD (Laboratório de CAD Inteligente), pela utlização das estações de trabalho.

Ao Peter pelas fotografias apresentadas na defesa e suporte das estações de trabalho.

Ao apoio financeiro dado pela PROMON Engenharia, CAPES e IBM (num convênio durante um período de elaboração deste trabalho, conseguido através do Prof. Sérgio Fontoura).

RESUMO

O desenvolvimento da Computação Gráfica na última década causou um grande impacto na forma de se projetar sistemas. Como os sistemas gráficos podem ser desenvolvidos para uma gama muito grande de atividades, torna-se natural a intenção de se reutilizar módulos específicos já existentes e testados como módulos independentes em novos sistemas. Na maioria dos casos, isto se torna bastante dispendioso, uma vez que sempre há a necessidade de modificação destes módulos para colocá-los em condições ideais de uso, seja alterando-os, expandindo-os ou até mesmo reduzindo-os.

Sob esta ótica, o potencial de alteração/expansão do sistema tornou-se um fator muito importante a ser levado em consideração quando do desenvolvimento de um sistema novo. A linguagem orientada a objetos passa, então, a ser uma opção natural dos programadores devido às marcantes vantagens que a sua filosofia introduz nestes quesitos.

Este trabalho aborda o desenvolvimento de um programa básico de análise linear elástica pelo método dos elementos finitos (formulação em deslocamentos) tanto para fins acadêmicos, como para servir de base para futuros desenvolvimentos que darão prosseguimento à linha de pesquisa.

Uma hierarquia de classes, dentro de uma filosofia orientada a objetos, cujo objetivo é a computação de matrizes de rigidez na solução numérica do problema da elasticidade pelo método dos elementos finitos é apresentado. A solução enfoca o problema da análise de modelos aporticados, planos,de placa e casca, e tridimensionais. Dentro deste contexto, estão descritas a seleção das classes de objetos necessárias para representar o problema e a especificação das mensagens, e os correspondentes métodos, para as classes.

Este trabalho também enfoca o pós-processamento bidimensional de elementos finitos através do uso da computação gráfica interativa. Aproveitou-se o ambiente orientado a objetos para servir de base para este desenvolvimento que utiliza diversas técnicas para visualização de resultados da análise. Estão disponíveis a visualização do modelo deformado e dos campos de tensão através da representação do fluxo de tensões principais, contorno de tensões nodais e em pontos de integração de Gauss e diagramas de tensões ao longo de linhas arbitrárias sobre o modelo.

ABSTRACT

Computer Graphics improvements over the past decade caused a great impact in system designing. Since graphics systems can be developed for a wide variety of activities, it becomes natural the reuse of specific existing modules as independent modules in new systems. In most cases, this task is very expensive, since there is always the need to modify those modules to put then in proper conditions for use, by altering, expanding or even reducing them.

Based on this, the capacity to alter/expand a system is a very important factor to be considered in its development. Object oriented programming is a natural option for programmers due to the outstanding advantages that its philosofy introduces in these subjects.

This work approaches the development of a basic program for linear elastic analysis using the finite element method (displacement formulation) which can be used as a academic tool or as a basis for future developments in the research group in PUC-Rio.

A class hierarchy, in the context of a object oriented philosofy, whose objective is the stiffness matrix computation in the numeric solution of the elasticity problem by finite element method, is introduced. The solution details the analysis problem of beam, plane, plate, shell and three-dimensional models. In this context, the selection of the object classes needed to represent the problem and the message specifications, and the corresponding methods, are described.

This work also considers the bi-dimensional postprocessing of finite element models through the use of interactive computer graphics. This development uses many technics to visualize analysis results and is also based on the object oriented environment. These technics comprise deformed model visualization, principal stresses flow representation, nodal and Gauss points stress contours, and stress diagrams along arbitrary lines across the model.

Sumário

1. Introdução 1

1.1 Escopo e Objetivos 2

1.2 Histórico 3

1.3 Organização da tese 4

2. Conceitos de Programação Orientada a Objetos 5

2.1 Relação Cliente-Fornecedor 5

2.2 Classes e Objetos 7

2.3 Hierarquia de Classes 7

2.4 Exemplo ilustrativo 9

2.5 Metodologia de desenvolvimento 10

3. Organização de Classes para Computação da Matriz de Rigidez 12

3.1 Definição do problema 12

3.2 Seleção das classes 13

3.2.1 Classe Elemento 14

3.2.2 Classe Material 17

3.2.3 Classe Modelo de Análise 17

3.2.4 Classe Gauss 18

3.3 Especificação das Classes 18

3.4 Exemplo de adequação das classes 22

4. Implementação da Disciplina Orientada a Objetos 25

4.1 Encapsulação 25

4.2 Classes e Métodos Genéricos 25

4.3 Invocação de métodos 27

4.4 Herança 28

5. Representação de Dados para Pós-processamento Bidimensional 30
5.1 Classe Elemento para pós-processamento 30
5.2 Estruturas de dados 32
5.2.1 Representação explícita do contorno do modelo 32
5.2.2 Estrutura de dados para informação de elementos adjacentes a nós 34
5.3 Estratégia de utilização da memória 36

6. Técnicas de Visualização utilizadas 37
6.1 O modelo 37
6.2 A deformada 39
6.3 O fluxo de tensões 41
6.4 O contorno de tensões nodais 43
6.5 O diagrama de tensões 47
6.6 O contorno de tensões em pontos de integração de Gauss 50
6.7 Balanço das técnicas de contorno de tensões utilizadas 56

7. Conclusões e Sugestões 57

Referências Bibliográficas 60

Lista de Figuras

Figura 2.1 Relação Cliente-Fornecedor em ambiente convencional. 5

Figura 2.2 Relação Cliente-Fornecedor em ambiente de POO..... 6

Figura 2.3 Exemplo de uma organização hierárquica de classes..... 8

Figura 2.4 Algoritmos gerais do cliente e fornecedores..... 9

Figura 2.5 Trecho em pseudo-código do algoritmo do cliente em programação convencional..... 9

Figura 2.6 Trecho em pseudo-código do algoritmo do cliente em programação orientada para objetos..... 10

Figura 3.1 Organização hierárquica da classe Elemento..... 15

Figura 3.2 Subclasses implementadas da classe Elemento..... 16

Figura 3.3 Subclasses implementadas da classe Modelo de Análise..... 18

Figura 3.4 Protocolo e descrição dos métodos da classe Elemento..... 19

Figura 3.5 Protocolo e relação das classes que possuem métodos para responder às mensagens..... 20

Figura 3.6 Protocolo e descrição dos métodos da classe Material..... 21

Figura 3.7 Protocolo e descrição dos métodos da classe Modelo de Análise..... 21

Figura 3.8 Protocolo e descrição dos métodos da classe Gauss..... 22

Figura 3.9 Descrição do objeto da classe elemento..... 22

Figura 3.10 Algoritmo de computação da matriz [B]..... 23

Figura 3.11 Algoritmo de computação da matriz de rigidez [K]..... 24

Figura 4.1 Declaração de alguns campos da estrutura da classe Elemento..... 26

Figura 4.2 Exemplo de inicialização de uma subclasse..... 27

Figura 4.3 Declaração de macros de mensagens para alguns métodos da classe Elemento..... 28

Figura 5.1 Organização hierárquica da classe Elemento..... 31

Figura 5.2 Protocolo e descrição dos métodos da classe Elemento. 31

Figura 5.3 Estrutura de dados para representar a fronteira do modelo..... 33

Figura 5.4 Representação do modelo com a malha..... 34

Figura 5.5 Representação do contorno do modelo com a presença de furos..... 34

Figura 5.6 Estrutura de dados para armazenar elementos adjacentes aos nós.... 35

Figura 6.1 Modelo com as características iniciais adotadas pelo pós-processador. 38

Figura 6.2 Modelo com a representação da malha de elementos finitos..... 38

Figura 6.3 Modelo sem a representação das restrições de apoio..... 39

Figura 6.4 Modelo com a configuração deformada..... 40

Figura 6.5 Modelo com as configurações deformada e indeformada..... 40

Figura 6.6 Modelo com as configurações indeformada e deformada ampliada.... 41

Figura 6.7 Representação do fluxo de tensão principal máxima..... 41

Figura 6.8 Visualização do fluxo das tensões princpais máximas e mínimas 42

Figura 6.9 Visualização do fluxo das tensões princpais de tração..... 42

Figura 6.10 Visualização do fluxo das tensões princpais de compressão..... 43

Figura 6.11 Visualização do contorno de tensões nodais suavizado..... 43

Figura 6.12 Visualização do contorno de tensões nodais não suavizado..... 44

Figura 6.13 Detalhe do contorno não suavizado em região de concentração de tensões..... 45

Figura 6.14 Detalhe do contorno de tensões suavizado em região de concentração de tensões..... 46

Figura 6.15 Interseção do plano de corte arbitrário com os planos de extrapolação de tensões..... 47

Figura 6.16 Interseção da linha fornecida com as arestas do modelo..... 48

Figura 6.17 Diagrama de tensões referente ao campo suavisado visualizado..... 49

Figura 6.18 Diagrama de tensões referente ao campo não suavisado visualizado. 49

Figura 6.19 Polígonos internos aos elementos..... 51

Figura 6.20 Polígonos em torno dos nós internos..... 51

Figura 6.21 Polígonos referentes a cada aresta interna..... 52

Figura 6.22 Visualização do contorno de tensões em pontos de Gauss com a representação da malha auxiliar gerada..... 52

Figura 6.23 Polígonos em torno de cada nó da fronteira..... 53

Figura 6.24 Polígonos referentes a cada aresta da fronteira..... 53

Figura 6.25 Representação do contorno de tensões em pontos de Gauss..... 54

Figura 6.26 Representação do detalhe do contorno de tensões em pontos de Gauss. 55

Figura 6.27 Representação do contorno de tensões em pontos de Gauss com aproximação na fronteira..... 55

Figura 6.28 Representação do contorno de tensões em pontos de Gauss com aproximação na fronteira..... 56

1

Introdução

O desenvolvimento da Computação Gráfica na última década foi responsável por um grande impacto na forma de se projetar sistemas. Como os sistemas gráficos podem ser desenvolvidos para uma gama muito grande de atividades, torna-se natural a intenção de se reutilizar módulos específicos já existentes e testados como módulos independentes em novos sistemas. Na maioria dos casos, isto se torna bastante dispendioso, uma vez que sempre há a necessidade de modificação destes módulos para colocá-los em condições ideais de uso, seja alterando-os, expandindo-os ou até mesmo reduzindo-os.

Sob esta ótica, o potencial de alteração ou expansão do sistema torna-se um fator muito importante a ser levado em consideração quando do desenvolvimento de um sistema novo. A programação orientada a objetos passa a ser uma opção natural dos programadores devido às marcantes vantagens que a sua filosofia introduz nestes quesitos (Goldberg et. al., 1983).

A maioria dos enfoques para desenvolvimento de *software* para Engenharia enfatizam a metodologia para resolver o problema em questão, quer pelo método numérico, estratégia de solução numérica ou lógica utilizada, em detrimento da representação de dados. Não que se queira dizer que um ponto é mais ou menos importante que o outro, mas quer-se enfatizar que a representação de dados é igualmente importante no desenvolvimento de um *software* (Fenves, 1990).

Este trabalho aborda o desenvolvimento de um programa básico de análise linear elástica pelo método dos elementos finitos (formulação em deslocamentos) representando os dados sob o enfoque da orientação a objetos. Este programa serve como base tanto para fins acadêmicos como para futuros desenvolvimentos que darão prosseguimento à linha de pesquisa no qual o trabalho está inserido na PUC-Rio.

1.1 *Escopo e objetivos*

O desenvolvimento de um sistema de análise numérica pelo método dos elementos finitos é normalmente feito de uma forma incremental. O programa parte de uma implementação simples, com uma pequena família de elementos e com um número limitado de modelos e procedimentos de análise, e sua funcionalidade é acrescida durante o processo de desenvolvimento. Técnicas de programação orientada a objetos são ideais para este tipo de desenvolvimento incremental e parece natural que sejam adotadas neste tipo de ferramenta numérica. Como conseqüência imediata, tem-se um produto com vida útil mais longa.

No entanto, ainda é limitado o uso de linguagens orientadas a objetos em programas de análise numérica. Isto porque, além de serem recentes, elas podem comprometer a eficiência do processamento numérico. Quando se quer avaliar a eficiência de uma plataforma a ser adotada no projeto de um sistema tem-se que levar em consideração as fases de desenvolvimento, manutenção e utilização. O paradigma da programação orientada a objetos garante uma grande eficiência nas duas primeiras etapas citadas em detrimento da performance (Fenves, 1990). Este comprometimento se deve principalmente ao processo de busca, em tempo de execução, de um procedimento a ser executado. Já nas linguagens convencionais esta decisão é tomada em tempo de compilação, definida pelo nome da subrotina ou função.

Na fase inicial deste desenvolvimento, o trabalho descreve uma hierarquia de classes, dentro de uma filosofia orientada para objetos, cujo objetivo é a computação das matrizes de rigidez dos elementos finitos na solução numérica do problema da elasticidade. A solução aborda o problema da análise de modelos aporticados, planos, de placa, de casca e tridimensionais.

Pelos motivos citados acima, adotou-se para este programa, em fase de prototipagem, uma plataforma de desenvolvimento baseada na linguagem C, padrão ANSI, onde, através de uma disciplina de programação, conseguiu-se unir a portabilidade conferida pela padronização com o potencial de expansão garantido pelos mecanismos implementados. Deve-se ressaltar que, nesta fase, o mais importante é a organização de um sistema baseado no método dos elementos finitos para problemas de elasticidade segundo esta filosofia de programação.

Este trabalho também enfoca o desenvolvimento de um sistema para pós-processamento bidimensional de elementos finitos através do uso da computação gráfica interativa. Aproveitou-se o ambiente orientado a objetos criado para a análise numérica

para servir de base para este desenvolvimento que utiliza diversas técnicas para visualização de resultados da análise. A adoção de uma estrutura de dados complementar à típica de elementos finitos possibilitou a utilização de algumas técnicas de visualização interativamente, como contorno de tensões em pontos de integração de Gauss e diagrama de tensões ao longo de uma reta definida arbitrariamente pelo usuário. Também estão disponíveis a visualização do modelo deformado e dos campos de tensão através da representação do fluxo de tensões principais e contorno de tensões nodais suavizadas ou não.

Os dois programas desenvolvidos são independentes, porém integrados por uma única interface. Eles fazem parte de um projeto que também envolve um pré-processador gráfico interativo que conta com um modelador geométrico para geração de modelos de elementos finitos (Vianna, 1992). Este projeto tem o intuito de criar um ambiente poderoso para manipular modelos de elementos finitos, já que elimina as fases de criação manual da malha e de consultas em extensas listagens. Isto propicia um aprimoramento do modelo, importante não só pelo resultado em si, mas também como ferramenta didática, pois permite muitas análises num curto período de tempo.

1.2 *Histórico*

Alguns trabalhos que utilizam linguagens orientadas a objetos para análise numérica têm sido publicados. O trabalho de Forde et. al. (1990) contem uma descrição detalhada desta técnica aplicada à análise por elementos finitos. No entanto, os autores focalizam o problema somente para estado plano de tensão e para elementos isoparamétricos. No presente trabalho, procura-se resolver o problema para diversos tipos de elementos (barra, membrana, placa, casca, sólido) e diversos tipos de análise (pórtico, estado plano de tensão, estado axissimétrico, etc.). Isto é alcançado através da criação de uma nova classe para gerenciar esta diversificação de tipos de análises. Outros trabalhos em áreas correlatas que devem ser mencionados são os de Alves Filho e Devloo (1991) e o de Fenves (1990).

No que diz respeito a pós-processamento, trabalhos anteriores (Bailey, 1986; Panthaki, 1987; Wawrzynek, 1987) comprovam a ineficiência da estrutura de dados convencional de elementos finitos para implementação das funções básicas que dão suporte às técnicas de visualização de resultados. Portanto deve-se estudar que tipo de estrutura de dados adicionais devem ser implementadas de acordo com cada caso.

Dentro da mesma linha de pesquisa deste trabalho, na PUC-Rio, o trabalho de Celes Filho (1990) abordou o problema da visualização de resultados de modelos tridimensionais de elementos finitos utilizando estruturas de dados mais completas. Os resultados foram bastante animadores, o que motivou a busca de novas técnicas de visualização, sugeridas neste trabalho.

1.3 *Organização da tese*

A apresentação dos principais conceitos da filosofia de orientação a objetos é feita no capítulo 2. Para um melhor entendimento, é descrito um exemplo simples onde são comparadas a filosofia adotada com a técnica convencional. Também é colocada a mudança na metodologia de desenvolvimento de um sistema que adote o paradigma de orientação a objetos.

O capítulo 3 define o problema do método dos elementos finitos no que diz respeito a implementação de um procedimento para montagem de matrizes de rigidez e descreve como ele foi organizado para atender a filosofia de desenvolvimento adotada.

O ambiente de desenvolvimento que serviu de base para todo o trabalho é apresentado no capítulo 4. Todos os conceitos apresentados no capítulo 2 são enfocados segundo a ótica da implementação adotada.

O capítulo 5 aborda o problema do pós-processamento bidimensional de elementos finitos que é avaliado pelo ponto de vista da filosofia adotada para desenvolvimento. Também são apresentadas as estruturas de dados adicionais criadas para melhorar o desempenho de interação. Por fim descreve-se a estratégia de utilização da memória do computador.

As diversas técnicas de visualização utilizadas no pós-processamento são descritas no capítulo 6. Cada uma delas é discutida e apresentada através de figuras. Este capítulo também apresenta exemplos genéricos das diversas possibilidade oferecidas para pós-processamento. Todos os exemplos foram gerados pelo pré-processador já mencionado e analisados pelo programa de análise por elementos finitos desenvolvido neste trabalho.

Finalmente, no capítulo 7 são apresentadas as conclusões e sugestões para trabalhos futuros que darão prosseguimento à linha de pesquisa.

2

Conceitos de Programação Orientada a Objetos

O objetivo deste capítulo é caracterizar os conceitos básicos de programação orientada a objetos (POO). Isto serve como uma base para os capítulos subseqüentes. Os conceitos apresentados estão baseados principalmente no trabalho de Cox (1987), que define dois princípios fundamentais neste ambiente de programação: *encapsulação* e *herança*.

2.1 *Relação Cliente-Fornecedor*

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

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* e os demais são os *fornecedores*.

Na programação convencional, a relação entre cliente e fornecedor se materializa através da transferência de dados, onde as duas partes possuem o conhecimento das estruturas de dados e dos operadores, como mostra a Figura 2.1.

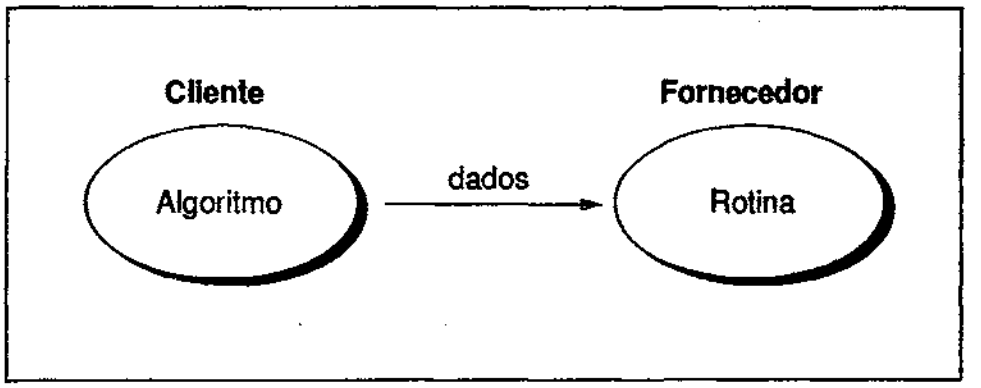

Figura 2.1: Relação Cliente-Fornecedor em ambiente convencional.

O cliente decide como cada operação deve ser executada, chamando a rotina específica de cada fornecedor. Quando vai-se 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 fornecedor que apenas definem a tarefa. Isto é, o cliente apenas decide qual a operação que deve ser executada, acionando genericamente (através de mensagens) o procedimento que a processa, como mostra a Figura 2.2.

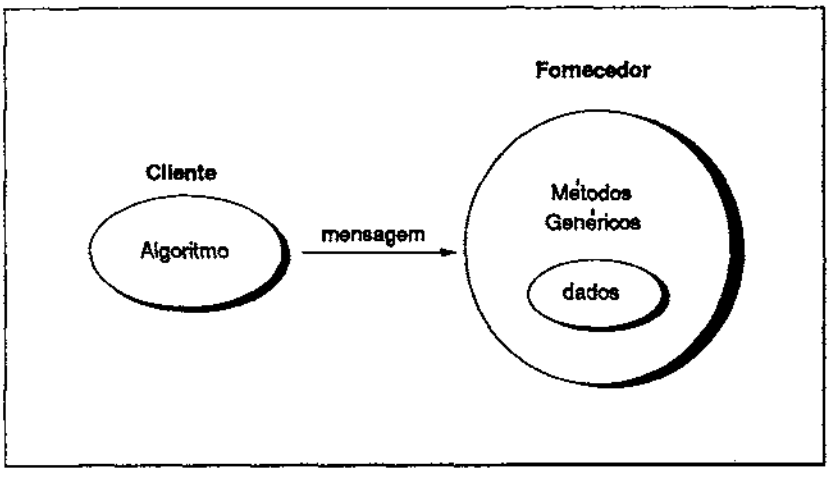

Figura 2.2: Relação Cliente-Fornecedor em ambiente de POO.

Para que isto se torne possível, cada fornecedor cria uma fronteira envolvendo e escondendo seus procedimentos, chamados de *métodos*, e seus dados. Isto é conhecido como princípio da *encapsulação*. Portanto, pela abordagem da filosofia orientada a objetos, o cliente somente possui conhecimento da existência dos métodos genéricos que, por sua vez, acessam os procedimentos e os dados específicos de cada fornecedor.

A encapsulação transfere o conhecimento e a responsabilidade da execução das tarefas para cada fornecedor. Com isto, o cliente lida apenas com algoritmos globais que acionam os métodos genéricos através de mensagens, cabendo aos fornecedores manipular seus procedimentos e dados privados. Pode-se fazer uma associação entre a emissão de uma mensagem pelo cliente e uma chamada convencional de subrotina ou função. A diferença fundamental é que, além de não estar explícito qual a subrotina (função) que será executada, são passados apenas parâmetros genéricos, já que a encapsulação esconde a estrutura de dados dos fornecedores.

2.2 *Classes e Objetos*

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

Os *objetos*, neste tipo de programação, podem ser vistos como elementos de uma classe e definidos como suas *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 seus procedimentos e área de dados privada. Como nenhum detalhe deste processo é transparente ao cliente, qualquer alteração nos procedimentos de um objeto não implica em conseqüências fora do seu domínio, ou seja, toda mudança é localizada.

Um programa orientado a objetos consiste de objetos passando mensagens entre si. O programa transforma o estado dos objetos quando estes respondem às mensagens enviadas. A mudança de estado se reflete na mudança dos valores das variáveis locais dos objetos, e os objetos finais representam a solução do problema (Goldberg, et. al., 1986).

2.3 *Hierarquia de Classes*

Como todo conjunto, uma classe pode ser composta de *subclasses*, que seriam seus subconjuntos. Da relação inversa, nasce a *superclasse*. A idéia de se criar subclasses é a de se aumentar o grau de especificidade de cada uma a ponto de, ao se chegar à base da organização, elas contemplem somente métodos específicos dos seus objetos. A Figura 2.3 mostra uma organização hierárquica de classes.

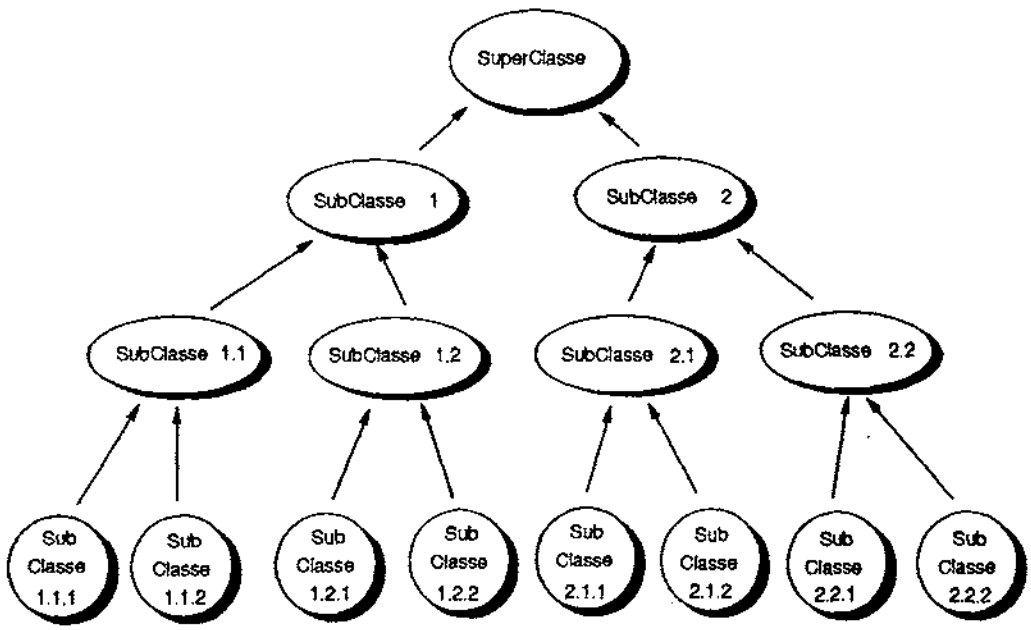

Figura 2.3: Exemplo de uma organização hierárquica de classes.

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, herda os métodos da sua superclasse. Isto significa que eles estão 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 conhecido como *overwriting*.

Conforme mencionado anteriormente, a comunicação entre cliente e fornecedores 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. A seleção do método depende da classe do objeto que recebe a mensagem e da sua posição na hierarquia de classes. Quando uma mensagem é enviada para um objeto, o sistema primeiro verifica se o método está definido para a classe receptora. Se estiver, ele é executado, caso contrário, a procura procede para cima na hierarquia de classes (ver sentido das setas na Figura 2.3) até que se encontre, em uma classe superior, o método correspondente à mensagem enviada; neste ponto o método é executado. Se o topo da hierarquia é atingido, o sistema assinala um erro.

2.4 *Exemplo ilustrativo*

Buscando elucidar um pouco mais estes conceitos, esta seção descreve um exemplo simples. Seja um programa que tem por objetivo desenhar linhas geométricas para formar um modelo, tal como mostrado na Figura 2.4.

Segue-se com trechos em pseudo-código do algoritmo geral do cliente, primeiramente para o ambiente de programação convencional (Figura 2.5) e em seguida para a programação orientada a objetos (Figura 2.6).

```
Algoritmo geral do cliente:
        Percorre todas as linhas do modelo e solicita que cada uma seja desenhada.

Rotinas específicas dos fornecedores:
        Desenha a linha para um determinado tipo de geometria.
```

Figura 2.4: Algoritmos gerais do cliente e fornecedores.

Na programação convencional o cliente solicita a execução de uma tarefa através da chamada para uma subrotina, transferindo os dados necessários.

```
Para todas as linhas do modelo faça
        Conforme ( tipo de geometria )
            caso Reta:
                DesenhaReta ( dadosReta )
            caso Círculo:
                DesenhaCirculo ( dadosCirculo )
            caso Bezier:
                DesenhaBezier ( dadosBezier )
            caso Spline:
                DesenhaSpline ( dadosSpline )
        Fim-Conforme
    Fim-Para
```

Figura 2.5: Trecho em pseudo-código do algoritmo do cliente em programação convencional.

Em POO, no pseudo-código da Figura 2.3, o cliente emite uma mensagem através de uma função identificada como `Mensageiro()` onde o primeiro argumento é o identificador do objeto e o segundo o identificador da mensagem.

```
Para todas as linhas do modelo faça
    Mensageiro ( linha, DesenhaLinha )
Fim-Para
```

Figura 2.6: Trecho em pseudo-código do algoritmo do cliente em programação orientada para objetos.

Através deste exemplo vê-se claramente a transferência de responsabilidades e conhecimento do cliente para os fornecedores. Caso fosse necessário incluir mais um tipo de linha geométrica no programa, o algoritmo do cliente na programação convencional deveria ser modificado, o que não ocorre com o do orientado a objetos. Obviamente, em um sistema complexo estas dependências estariam espalhadas por todo o código, dificultando muito a programação incremental. No exemplo de POO apresentado, se uma rotina específica de um novo tipo de linha fosse inserida, o algoritmo do cliente continuaria funcionando sem problemas. Isto é, não seria preciso que houvesse qualquer alteração no restante do código. Esta facilidade é tanta que esta qualidade é comparada com a facilidade que o computador possui de mudar de configuração (Cox, 1987). Para que isto ocorra, basta que se instale uma determinada placa numa posição específica que tudo passa a funcionar da maneira desejada. Na POO, seria como se o novo módulo fosse simplesmente acoplado ao programa.

2.5 *Metodologia de desenvolvimento*

A fase de desenvolvimento de um programa envolve a especificação e a implementação da representação de seus dados. Em POO, 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 cada mensagem, ou seu método associado, faz; a especificação contem 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.

Vale ressaltar que as duas primeiras etapas independem da linguagem adotada para implementação.

3

Organização de Classes para Computação da Matriz de Rigidez

Como foi descrito no capítulo anterior, a criação das classes e suas hierarquias é fruto de uma análise global do problema a ser resolvido. Na análise linear elástica pelo método dos elementos finitos, o procedimento mais importante a ser avaliado é o da montagem da matriz de rigidez de um elemento. Por esta razão este processo foi o primeiro a ser estudado dentro da linha de pesquisa.

Este capítulo descreve os dois primeiros passos do desenvolvimento deste processo, isto é, a seleção das classes e suas especificações.

3.1 *Definição do problema*

O problema básico a ser resolvido é a computação da matriz de rigidez de um elemento finito. A obtenção canônica desta matriz está mostrada na expressão (3.1) (Zienkiewicz, 1989).

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

Onde,

[K] – Matriz de rigidez do elemento.

[B] – Matriz de compatibilidade de deslocamentos (relaciona deformações internas com deslocamentos nodais).

[E] – 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).

$$\left[K\right] = \sum_{i=1}^{n} \left[B\right]^T \left[E\right] \left[B\right] \left|J\right| \omega_i \tag{3.2}$$

Onde,

n – Número de pontos de integração de Gauss.

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

ω_i – Peso associado ao ponto de integração.

3.2 *Seleção das classes*

O primeiro passo do estudo deste processo é uma avaliação das entidades envolvidas para que se possa determinar o fluxo de informações e quem as detém. É bastante claro que são três os eventos a serem avaliados: 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 número de nós, e do tipo de análise 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 do elemento e do tipo de análise que o solicita. 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 e faz surgir a figura da matriz Jacobiana que depende unicamente da forma do elemento finito.

Com os passos do processo da montagem da matriz de rigidez do elemento finito organizados desta forma, definiu-se quatro classes que estruturam a resolução do problema dentro do contexto da programação orientada a objetos. São elas a classe Elemento (*Element*), a classe Material, a classe Modelo de Análise (***Analysis Model***) e a classe Gauss.

A comprovação de que as classes escolhidas para o processo de montagem da matriz de rigidez do elemento são adequadas é feita na seção 3.4, onde é exemplificado como estas classes se relacionam.

3.2.1 *Classe Elemento*

Talvez a mais intuitiva de todas as classes, ela é responsável por fornecer ao cliente (algoritmo que utiliza a classe) todas as informações necessárias à resolução do problema que dependam exclusivamente do elemento finito. Ela armazena todas as informações geométricas do elemento e detem o conhecimento das funções de forma e suas derivadas. A abrangência desta classe é muito ampla pois existem vários tipos de elementos diferentes. Portanto, foram criadas subclasses hierarquicamente inferiores, obtendo uma especificidade de conhecimentos maior. A Figura 3.1 mostra a organização hierárquica das subclasses da classe Elemento.

A primeira explosão desta classe gerou quatro sub-classes definidas de acordo com a forma do elemento e a sua localização no espaço. São elas a classe de elementos de barra (Bar), a classe de elementos planos (Plane), a classe de elementos de casca (Shell) e a classe de elementos sólidos (Solid).

Em uma segunda explosão, a classe Plane é subdividida nas classes de elementos de membrana (Membrane), formada por elementos de continuidade C0, e de elementos de placa (Plate). Esta última ainda é subdividida nas classes de elementos de placa delgada (ThinPlate) e de placa espessa (ThickPlate).

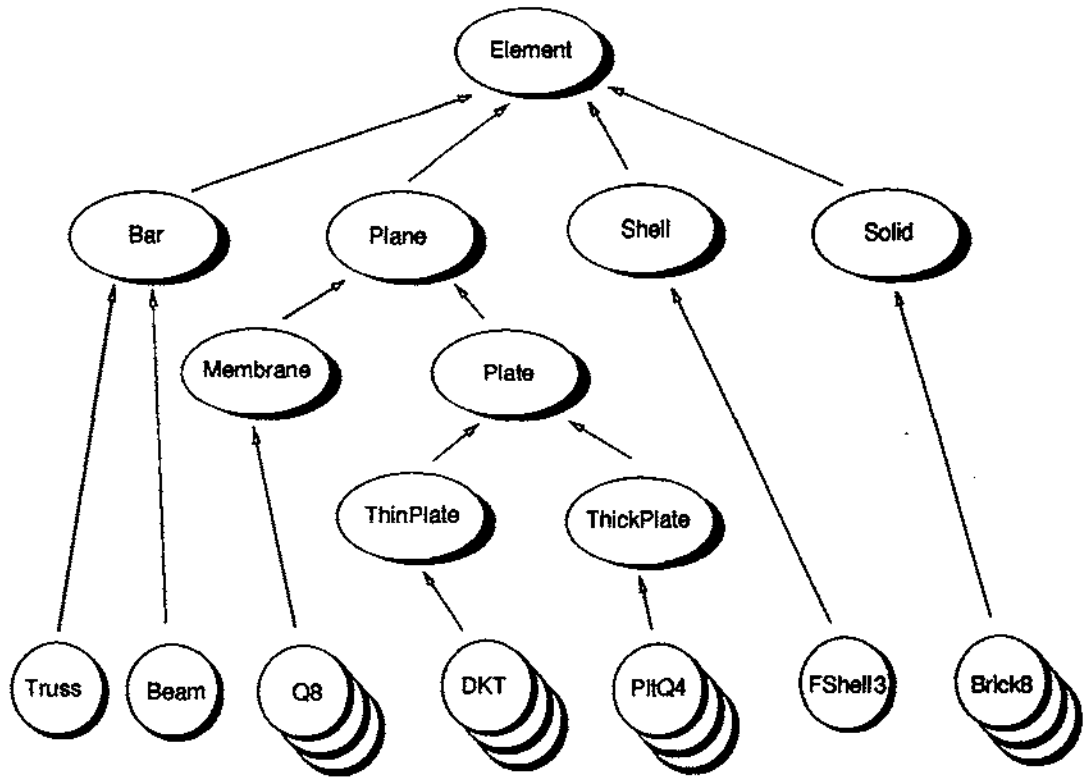

Figura 3.1: Organização hierárquica da classe Elemento.

Com isto, a superclasse Elemento passa a ser responsável apenas por métodos globais, ou seja, que atendam a todos os tipos de elementos, e transfere todo o conhecimento específico a cada uma das subclasses criadas. A hierarquia fica caracterizada através da herança que cada subclasse recebe da superclasse, isto é, a possibilidade de utilizar seus métodos.

O último nível de classes criado representa cada tipo de elemento implementado. Tomando como exemplo a classe de Membrane, vê-se que ela somente deve responder por procedimentos comuns a todos os elementos que se enquadram nela. A explosão desta classe gera, por exemplo, as classes Q8, Q4, T3, T6, etc., cujas instâncias são os próprios elementos finitos.

A Figura 3.1 corresponde ao estado atual da implementação, onde somente um elemento de casca está implementado. Provavelmente a classe Shell será subdividida em várias subclasses que serão necessárias para a implementação de outros elementos desta classe.

A principal vantagem desta organização de classes é o desenvolvimento incremental de uma família de elementos. Para a implantação de um novo elemento finito no sistema, tem-se que, primeiramente, avaliar dentro da organização, aonde ele se encaixa. A relação hierárquica entre as classes, caracterizada pela herança de métodos, diminui

consideravelmente o custo da expansão. O primeiro passo é a avaliação da possibilidade de encaixar o novo elemento como "folha" na "árvore", o que irá ocorrer na maioria dos casos. Este fato indica que ele já herdará automaticamente todos os métodos das classes hierarquicamente superiores. Apenas os seus métodos específicos devem ser desenvolvidos. Isto sem mencionar o fato de que, como novo fornecedor, não causará nenhum impacto sobre o cliente, isto é, estará disponível para todos os algoritmos globais. As subclasses implementadas até o momento se encontram na Figura 3.2.

Bar			
		TRUSS	barra de treliça tri-dimensional com 2 nós
		BEAM	barra de quadro tri-dimensional com 2 nós
Plane			
	Membrane		
		T3	triângulo com 3 nós
		T6	triângulo com 6 nós
		Q4	quadrilátero com 4 nós
		Q5	quadrilátero subparamétrico com 5 nós
		Q6	quadrilátero subparamétrico com 6 nós
		Q6B	quadrilátero isoparamétrico com 6 nós
		Q8	quadrilátero com 8 nós
		Q9	quadrilátero com 9 nós
		INFL4	infinito Lagrangeano com 4 nós
		INFS3	infinito Serendipity com 3 nós
		INFS5	infinito Serendipity com 5 nós
	Plate		
		ThinPlate	
		DKT	triângulo de Kirchhoff discreto com 3 nós
		DKQ	quadrilátero de Kirchhoff discreto com 4 nós
		ThickPlate	
		PLTQ4	quadrilátero com 4 nós
		PLTQ8	quadrilátero com 8 nós
Shell			
		FSHELL3	triângulo com 3 nós (T3 + DKT)
Solid			
		BRICK8	paralelepípedo com 8 nós
		BRICK20	paralelepípedo com 20 nós

Figura 3.2: Subclasses implementadas da classe Elemento.

3.2.2 *Classe Material*

O material é encarado como uma entidade sólida, ou seja, tridimensional. O tipo de análise é que define se ele é tratado como uni, bi ou tridimensional quando da resolução do problema. A classe armazena todas as grandezas que caracterizam o material, ou seja, módulos de elasticidade, coeficientes de Poisson, massa específica, etc.

Ao analisar-se os métodos que ela deveria possuir para atender aos clientes, viu-se que ela também deveria ser subdividida. Criou-se duas subclasses, a de material Isotrópico e a de material Ortotrópico. Estas subclasses detêm o conhecimento específico de cada um dos tipos de materiais.

3.2.3 *Classe Modelo de Análise*

Certamente a mais abstrata de todas as classes, a classe Modelo de Análise permitiu que todas as outras classes ficassem "limpas", ou seja, sem nenhum tipo de dependência. Também evita que o cliente possua qualquer conhecimento específico sobre as características do elemento ou particularidades do tipo de análise efetuada. A necessidade da sua criação pode ser exemplificada através da análise de um elemento plano em estado plano de tensão comparada com uma análise axissimétrica do mesmo elemento. Caso esta classe não existisse, quando o cliente solicitasse a criação da matriz [B], o conhecimento da classe elemento deveria ser tal que, para a mesma mensagem, houvessem dois métodos diferentes para serem ativados, pois as matrizes são diferentes. Este fato se repetiria no caso da montagem da matriz [E]. Portanto, sem a classe modelo de análise, os princípios da filosofia adotada seriam feridos porque, ou o cliente seria responsável pela seleção onde dependesse do tipo de análise, ou todas as classes deveriam conhecer os tipos de análise e fazer a seleção. Este tipo de seleção deve ser evitado a todo custo pois este é um dos pontos fracos de um sistema em termos de expansão, na medida em que a criação de um novo tipo de análise implicaria em mudanças espalhadas por todo o código.

A principal tarefa desta classe é gerenciar a montagem das matrizes [B] e [E]. Como ocorreu nas demais classes, o modelo de análise também sofreu uma explosão para que cada tipo de análise se transformasse em subclasse. A Figura 3.3 esclarece a identidade de cada subclasse.

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

Figura 3.3: Subclasses implementadas da classe Modelo de Análise.

3.2.4 *Classe Gauss*

Esta classe foi criada para organizar o processo de integração de Gauss. Como o conhecimento desta classe também é vasto, optou-se por subdividí-la em cinco subclasses. São elas a Linear, a Triangular, a Quadrilateral, a Cúbica e a Tetraédrica, onde cada uma delas possui os conhecimentos específicos capazes de proceder a integração numérica de acordo com o domínio do elemento finito.

3.3 *Especificação das Classes*

A especificação de uma classe é composta por seu protocolo (lista de mensagens) e uma descrição sumária e precisa de cada método invocado pelas mensagens.

Por ser a mais complexa das classes, a especificação da classe Elemento é apresentada de uma forma especial. A Figura 3.4 mostra o protocolo da classe com a respectiva descrição dos métodos invocados e, complementando, a Figura 3.5 esclarece, dentro da organização hierárquica da classe, quais as subclasses que respondem a cada uma das mensagens.

Métodos para a classe:

Init	Inicializa a classe.
New	Cria uma instância da classe.

Métodos gerais para objetos:

Anmodel	Verifica a consistência do objeto com o tipo de análise global.
BMatrix	Retorna a matriz deformação-deslocamento do objeto avaliada no ponto fornecido.
Mass	Retorna a matriz de massa do objeto.
NMatrix	Retorna a matriz das funções de forma do objeto avaliada no ponto fornecido.
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.
Stiff	Retorna a matriz de rigidez do objeto.
Stress	Retorna as tensões nos pontos de Gauss e nodais do objeto.

Métodos de forma 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.
Jacobian	Retorna a matriz Jacobiana do objeto, sua inversa e seu determinante no ponto fornecido.
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.
RespOrder	Retorna a ordem de integração ótima (número de pontos em cada direção) para avaliação da resposta (tensões).
ShpFunc	Retorna as funções de forma avaliadas no ponto fornecido.
Thickness	Retorna a espessura dos elementos planos ou de casca e 1.0, se for sólido.
TRMatrix	Retorna a matriz de transformação utilizada para extrapolar as tensões dos pontos de integração de Gauss para os nós do objeto.

Métodos de restrição nodal para objetos:

ContrNode	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 3.4: Protocolo e descrição dos métodos da classe Elemento.

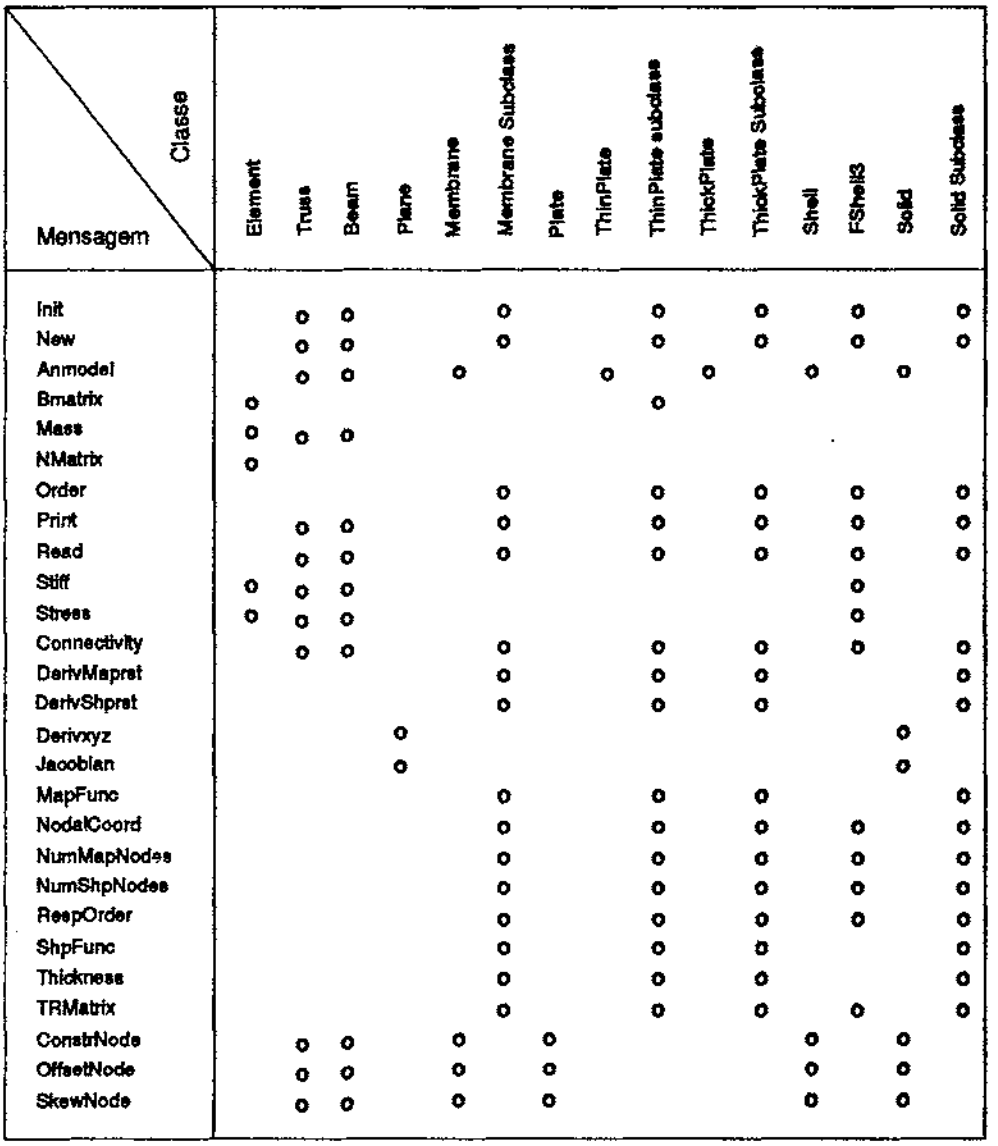

Figura 3.5: Protocolo e relação das classes que possuem métodos para responder às mensagens.

Na Figura 3.5, quando uma subclasse não responde a uma mensagem, significa que ou a subclasse herda o método correspondente de uma superclasse ou a mensagem não se aplica a subclasse. Observe, nesta figura, que praticamente todas as subclasses dos elementos implementados herdam os métodos de computação da matriz [B] (BMatrix) e da matriz de rigidez (Stiff), entre outros. Já os métodos mais especializados, como os ligados às funções de forma (MapFunc, DerivMaprst, ShpFunc e DerivShprst), ficam no nível mais baixo da "árvore", o que também ocorre com os métodos de consulta aos dados dos objetos (NumMapNodes, NumShpNodes, Connectivity, NodalCoord, etc.).

O mecanismo orientado a objetos propiciou a consideração de casos especiais no que se refere à obtenção da matriz de rigidez. Quando não interessa a sua obtenção de forma canônica, a subclasse tem o seu próprio método para montar a matriz de

rigidez. Nota-se, na Figura 3.5, que as subclasses (Beam, por exemplo) que possuem a suas matrizes formuladas, simplesmente não herdam os métodos envolvidos no processo canônico.

As demais classes possuem somente um nível de hierarquia, tornando o entendimento mais fácil. Portanto, as sua especificações ficam resumidas à apresentação das Figuras 3.6, 3.7 e 3.8.

Métodos para a classe:	
Init	Inicializa a classe.
New	Cria uma instância (objeto) na classe.
Métodos para objetos:	
Cmatrix	Retorna a matriz constitutiva do objeto (tri-dimensional).
Print	Imprime os dados do objeto.
Rho	Retorna a densidade do objeto.

Figura 3.6: Protocolo e descrição dos métodos da classe Material.

Métodos para a classe:	
Init	Inicializa a classe.
Métodos para objetos:	
EMatrix	Retorna a matriz tensão-deformação para uma dada matriz constitutiva tri-dimensional do material.
DimBMatrix	Retorna o número de linhas da matriz [B].
DofGlobDir	Retorna as direções globais consistentes com o tipo de análise.
MountBMatrix	Gerencia a montagem da matriz [B].
NodalStress	Retorna as tensões nodais avaliadas à 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.

Figura 3.7: Protocolo e descrição dos métodos da classe Modelo de Análise.

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 3.8: Protocolo e descrição dos métodos da classe Gauss.

3.4 *Exemplo de adequação das classes*

A adequação das classes criadas pode ser comprovada através dos algoritmos genéricos para computação da matriz [B] e da matriz de rigidez. As Figuras 3.10 e 3.11 apresentam os algoritmos em pseudo-código, de forma simplificada, correspondentes aos métodos invocados pelas mensagens BMatrix e Stiff e utilizados pela maioria dos objetos das sub-classes da classe Elemento que implementam elementos isoparamétricos e subparamétricos. Na apresentação dos algoritmos, a invocação de um método é feita através da utilização da função `msg()`, cujo primeiro argumento é o identificador do objeto e o segundo é o identificador da mensagem.

O objeto da classe Elemento, identificado nas Figuras 3.10 e 3.11 por `elem`, tem a descrição de seus dados apresentada na Figura 3.9.

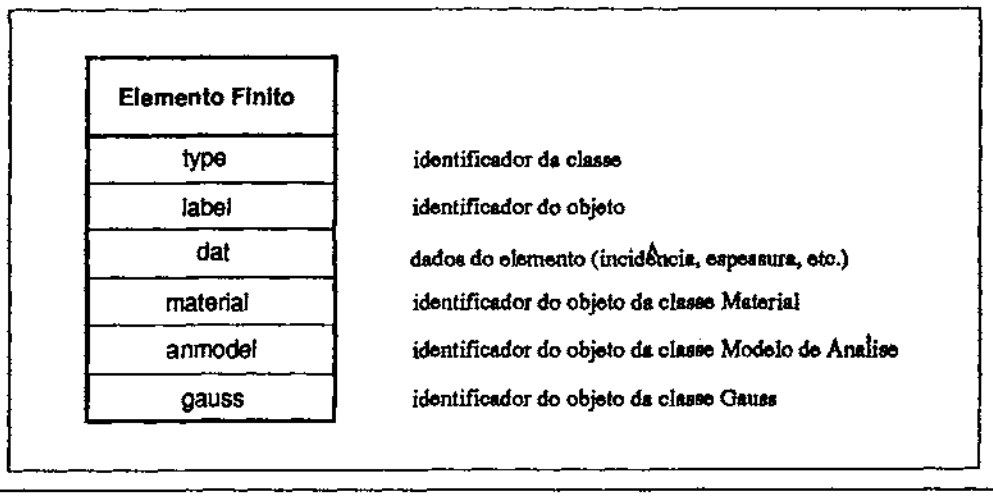

Figura 3.9: Descrição do objeto da classe elemento.

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

Figura 3.10: Algoritmo de computação da matriz [B].

Nota-se que o algoritmo descrito na Figura 3.11 invoca o método descrito pelo algoritmo da Figura 3.10 seguidas vezes dentro de uma estrutura em laço. O algoritmo que calcula a matriz [B], o faz para cada ponto de integração, por isso recebe como parâmetro de entrada os dados (coordenadas e peso) do ponto de Gauss corrente.

```
// obtem número de graus de liberdade por nó
numdofnode = msg( anmodel, NumDofNode );
// obtem o número de nós do elemento corrente
numnodes = msg( elem, NumMapNodes );
// calcula o número de graus de liberdade do elemento
ndof = numdofnode * numnodes;
// obtem as coordenadas dos nós do elemento
msg( elem, NodalCoord, elemcoord );
// 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 );
// obtem a matriz [C] do material
msg( mat, CMatrix, cmatrix );
// obtem o número de componentes de tensão
numstrcmps = msg( anmodel, DimBMatrix );
// monta a matriz [E] para o tipo de análise
msg( anmodel, EMatrix, cmatrix, ematrix );

foreach gauss point [i], i = 1..numgausspts
    begin
    // obtem a matriz [B]
    msg( elem, BMatrix, gaussrst[i], elemcoord, detjac, bmatrix );
    // obtem as funções de forma avaliadas no ponto de Gauss corrente
    msg( elem, MapFunc, gaussrst[i], mapfunc );
    // 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];
    // calcula o triplo produto e acumula na matriz de rigidez
    TripProdBtEB( numstrcmps, ndof, bmatrix, ematrix, coeff, elmstiff );
    end
```

Figura 3.11: Algoritmo de computação da matriz de rigidez [K].

4

Implementação da Disciplina Orientada a Objetos

Segundo Fenves (1990), quanto maior for a encapsulação dos dados em um programa desenvolvido de forma convencional, mais próximo da orientação a objetos ele estará. Além da encapsulação, que suporta a criação de classes e objetos, outros conceitos básicos foram enfocados. Implementou-se mecanismos capazes de simular a invocação de métodos através de mensagens, sejam eles da classe do objeto ou herdados de uma superclasse. Com isso, a organização hierárquica das classes fica assegurada, pois tem suporte no mecanismo de herança disponível. Estes recursos, descritos nas seções deste capítulo, formam o ambiente orientado a objetos criado para dar suporte a este trabalho.

4.1 *Encapsulação*

A encapsulação dos dados foi tratada da mesma forma para todas as classes de objetos. Cada sub-classe é representada por um arquivo que contem os códigos fontes de seus métodos e declaração dos seus objetos. Os métodos são implementados através de funções declaradas com a cláusula `static`, da linguagem C, o que os torna privativos da classe. Os dados dos objetos de uma classe também só são visíveis dentro dela, já que toda a alocação de memória para armazenar seus dados ocorre no arquivo que representa a classe. Isto implementa a fronteira que cada fornecedor cria em torno de seus procedimentos e dados mencionada no capítulo 2.

4.2 *Classes e Métodos Genéricos*

Em um programa formal orientado a objetos, o método invocado por uma mensagem é definido em tempo de execução, como descrito no capítulo 2.

Neste trabalho, a implementação do mecanismo que simula este comportamento foi feita através da utilização de variáveis "ponteiro de função" da linguagem C, que contêm endereços da memória onde se localizam funções.

Para, através de uma mensagem, invocar-se um método, algumas etapas devem ser cumpridas a nível de implementação. A declaração de uma classe e sua posterior inicialização são os procedimentos fundamentais da implementação.

Define-se um tipo de estrutura que representa a classe, utilizando-se a cláusula `struct` da linguagem C. Cada campo da estrutura que define a classe está associado a uma mensagem. Isto é, cada campo é uma variável ponteiro de função que recebe o endereço de cada método, como mostra a Figura 4.1, que apresenta a declaração da classe Elemento de forma simplificada.

```
typedef struct elementclass {

    void   (*new)           (int, FemElm **);
    void   (*bmatrix)       (FemElm *, NatCoord *, NodeCoord *, double *, double **);
    void   (*connectivity)  (FemElm *, int *);
    void   (*derivshprst)   (FemElm *, NatCoord *, DerivNat *);
    int    (*nummapnodes)   (FemElm *);

} ElementClass;
```

Figura 4.1: Declaração de alguns campos da estrutura da classe Elemento.

A declaração formal da classe e de suas subclasses ocorre quando é criado um vetor da estrutura declarada onde cada elemento está associado a uma subclasse. Conceitualmente, este vetor é o protocolo da classe e, por isso, é referenciado como vetor protocolo. O identificador da sub-classe, nesta implementação, é um índice para o vetor protocolo. A inicialização de uma subclasse nada mais é do que a atribuição do endereço de cada um dos seus métodos (funções genéricas) aos campos do elemento correspondente no vetor protocolo, como mostra a Figura 4.2, onde vê-se alguns campos do vetor protocolo recebendo os ponteiros das funções que implementam os métodos da subclsse Q4.

```
void InitQ4( void )
{
      elemclass[Q4].new              =     NewQ4;
      elemclass[Q4].bmatrix          =     elcbmatrix;
      elemclass[Q4].connectivity     =     ConnectQ4;
      elemclass[Q4].derivshprst      =     DerivMapQ4rst;
      elemclass[Q4].nummapnodes      =     NumNodesQ4;
}
```

Figura 4.2: Exemplo de inicialização de uma subclasse.

4.3 *Invocação de métodos*

A associação da mensagem ao método, nesta implementação, é feita através de definições do tipo "macro", utilizando a cláusula `define`, que escondem do cliente o mecanismo implementado, como mostra a Figura 4.3. Isto permite que, no futuro, se possa substituir este mecanismo por outro, talvez, até, dentro de um contexto formal de orientação a objetos.

Quando o cliente emite uma "mensagem" via macro que, obrigatoriamente, contem o identificador do objeto, tem-se os seguintes passos:

- O mecanismo reconhece, através da definição macro, o campo no elemento do vetor protocolo associado à mensagem.
- Através do objeto que recebeu a mensagem, determina-se qual a sub-classe que deve executar o método. Isto é feito através do campo "type" do objeto, que é um índice para o vetor protocolo.
- De posse do campo (variável ponteiro de função) do vetor e do índice (identificador da sub-classe), basta executar a função que estiver no endereço de memória que é o conteúdo da variável.

```
#define FemNew(type,lb,elm)                                          \
        (*elemclass[type].new)(lb,elm)

#define FemBMatrix(elm,gp,cd,d,bm)                                   \
        (*elemclass[elm->type].bmatrix)(elm,gp,cd,d,bm)

#define FemConnectivity(elm,cn)                                      \
        (*elemclass[elm->type].connectivity)(elm,cn)

#define FemDerivShprst(elm,gr,dr)                                    \
        (*elemclass[elm->type].derivshprst)(elm,gr,dr)

#define FemNumNodes(elm)                                             \
        (*elemclass[elm->type].nummapnodes)(elm)
```

Figura 4.3: Declaração de macros de mensagens para alguns métodos da classe Elemento.

Vale ressaltar que o acesso ao método é tão eficiente quanto a chamada convencional de uma função. No código compilado, a emissão de uma mensagem corresponde a mandar executar uma função que está num determinado endereço de memória que é o conteúdo de um campo num vetor de estruturas. Resumindo, é como se existisse uma tabela onde o número de linhas é igual ao número de sub-classes, o número de colunas igual ao número de mensagens e todos os elementos são ponteiros para função. Para executar um método, a mensagem identifica a coluna e a sub-classe, a linha.

4.4 *Herança*

A implementação da herança torna-se bastante simples após o entendimento de como uma mensagem invoca um método nesta implementação. Ela se dá em tempo de compilação. Como já foi mencionado, cada classe está contida em um arquivo. Portanto, um método herdado por uma subclasse é um procedimento cujo código fonte pertence ao arquivo de uma classe hierarquicamente superior. Ao inicializar seus métodos, a classe atribui o endereço de uma função a uma variável associada à mensagem, indexada pelo identificador da classe, no vetor protocolo. Para herdar um método, basta que a classe atribua à variável, o endereço do procedimento da superclasse. Vê-se, na Figura 4.2, que o método associado ao cálculo e montagem da matriz[B] está sendo herdado. Se

uma das subclasses não se interessar pela herança, ela atribui o endereço de uma outra função, implementando, assim, o conceito de "overwriting".

5

Representação de Dados para Pós-processamento Bidimensional

Dentro do paradigma da orientação a objetos, o caso de pós-processamento é menos complexo do que o da análise numérica pelo método dos elementos finitos, já que se resume em manipular os resultados gerados e tratá-los de forma a possibilitar a sua visualização através de recursos da computação gráfica.

No que diz respeito à representação de dados para pós-processamento, a estrutura de dados utilizada em análise numérica não satisfaz. A visualização do modelo e a representação dos resultados exige, muitas vezes, informações de adjacência que não estão explícitas na estrutura de dados típica de elementos finitos. Por se tratar de um sistema gráfico interativo, o tempo consumido entre a solicitação do usuário e a resposta do programa é fator fundamental na sua qualidade. Quanto mais informações topológicas forem armazenadas, mais rápida será a interação, contudo, tem-se que minimizar o consumo de memória. Este balanço deve ser bem estudado para que a estrutura de dados seja a mínima possível para assegurar uma resposta de interação satisfatória.

5.1 *Classe Elemento para pós-processamento*

Para o pós-processamento bidimensional desenvolvido neste trabalho, manteve-se ativada a plataforma orientada a objetos construida para o programa de elementos finitos e reaproveitou-se apenas a organização da classe Elemento, já que as outras perdem o sentido no ambiente de pós-processamento. A organização hierárquica da classe, mostrada na Figura 5.1, foi adaptada para a nova situação onde somente são tratados elementos de membrana. Com isto, os conhecimentos das classes Plane e Membrane transferiram-se para a classe Element e manteve-se as subclasses referentes aos elementos T3, Q4, T6 e Q8, os únicos tratados a nível de pós-processamento na implementação atual. Não existe limitação alguma para a consideração de outros elementos de membrana ou de placa. Apenas optou-se por deixar esta consideração para futuras implementações.

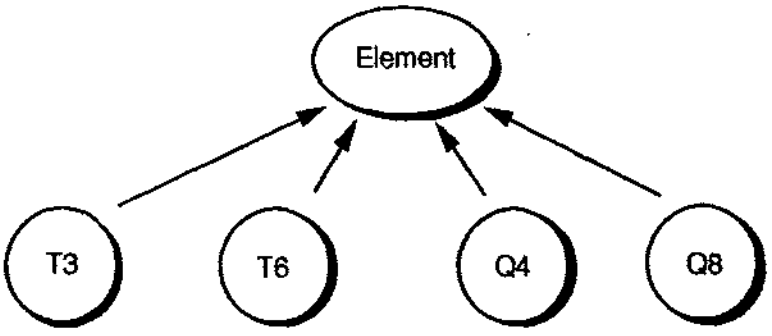

Figura 5.1: Organização hierárquica da classe Elemento.

As especificações das classes foram reavaliadas pois o cliente mudou e, com ele, suas necessidades. Com isto, novas operações devem ser realizadas sobre os objetos. Portanto, muitos métodos foram eliminados, alguns reaproveitados e outros, novos, criados. De uma maneira geral, os métodos descartados foram os ligados à computação da matriz de rigidez e os criados executam operações sobre os objetos com o objetivo de representá-los graficamente. A Figura 5.2 mostra a especificação da classe.

Métodos para a classe:

Init	Inicializa a classe.
New	Cria uma instância da classe.

Métodos gerais para objetos:

Read	Lê os dados do objeto no arquivo fornecido e os armazena.

Métodos de forma para objetos:

Connectivity	Retorna a incidência nodal do objeto.
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.

Métodos de pós-processamento dos objetos:

ReadStress	Lê as tensões do objeto e as armazena.
StressOrder	Retorna a ordem de integração para avaliação das tensões.
NodalDispl	Retorna os deslocamentos dos nós do objeto.
NodeVert	Verifica se um dado nó é vértice de canto do objeto.
VertConnect	Retorna a incidência do objeto desprezando os nós do meio das arestas.
GaussPolygon	Retorna os polígonos internos para geração da malha auxiliar.
GaussPolInf	Retorna o número de polígonos internos e de quantos pontos cada um é formado.

Figura 5.2: Protocolo e descrição dos métodos da classe Elemento.

5.2 *Estruturas de dados*

A representação do modelo e dos resultados de uma análise por elementos finitos possui algumas características que fazem com que determinadas informações sejam fundamentais. Por exemplo, ao se representar o modelo sem desenhar a malha, torna-se necessário conhecer as arestas do contorno. Já na representação dos resultados, é comum o procedimento de cálculo da média das tensões nodais para a simulação de um campo contínuo de tensões. Para obter estas informações sem uma estrutura de dados auxiliar, tem-se que promover uma série de consultas que dificultariam a utilização do sistema de forma interativa.

Optou-se pela utilização da estrutura de dados convencional de elementos finitos (incidência nodal) complementada por uma nova entidade que representa o contorno do modelo e uma informação adicional de adjacência que suporta a lista de elementos adjacentes a um nó. Todas estas informações são constantemente requisitadas para a visualização dos resultados da análise numérica do modelo e permanecem na memória enquanto o modelo estiver sendo processado.

5.2.1 *Representação explícita do contorno do modelo*

A estrutura de dados que guarda os contornos tem capacidade para armazenar um número indefinido de fronteiras, possibilitando a representação de furos. Como não sabe-se de antemão quantos contornos o modelo possui e nem quantos nós formam cada polígono que representa um contorno, optou-se pela utilização de listas simplesmente encadeadas para armazenar estes dados. Cada contorno é armazenado em uma lista simplesmente encadeada que guarda os números dos nós que formam a incidência do polígono de fronteira. Cada uma destas listas (tantas quanto forem o número de fronteiras), fica ligada a uma outra lista simplesmente encadeada, que gerencia todos os contornos, como mostra a Figura 5.3.

A criação desta estrutura de dados auxiliar é feita em tempo de inicialização para cada modelo em duas etapas: a criação de uma lista temporária de arestas do contorno e a sua transformação numa lista de nós ordenada, formando a incidência do polígono que representa o contorno.

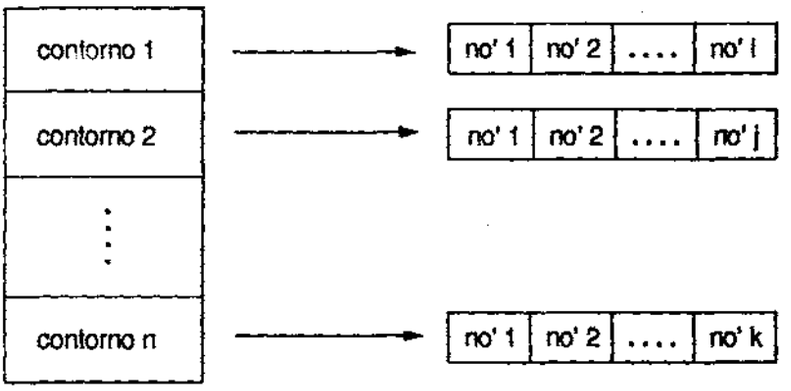

Figura 5.3: Estrutura de dados para representar a fronteira do modelo.

O procedimento de obtenção das arestas do contorno baseia-se num passeio por cada elemento do modelo quando, para cada uma de suas arestas, é verificado se ela já pertence à lista temporária. Se não pertencer, ela é armazenada, do contrário, ela é retirada. Ao final desta operação, só constam da lista as arestas pertencentes a apenas um elemento, ou seja, as do contorno. Note que, além de desordenadas, podem existir arestas de mais de um contorno, caso o modelo possua furos. Este algoritmo é de ordem linear com o número de elementos do modelo.

Neste momento, cria-se a estrutura de dados definitiva, tal como mostra a Figura 5.3. Consulta-se a lista temporária de arestas e retira-se a primeira aresta. Seus dois nós são armazenados na estrutura de dados. O primeiro é guardado para marcar o final do polígono e o segundo serve para procurar na lista qual a próxima aresta que o possui. Ao ser encontrada, ela é retirada da lista, e um novo nó é armazenado na estrutura que representa o polígono de contorno. Esta operação se repete até que o primeiro nó seja encontrado, o que significa que o polígono se fechou. Caso ainda reste alguma aresta na lista temporária, significa que o modelo possui outro contorno. Neste caso é criado outro elemento na lista que gerencia os contornos, apontando para mais um contorno. O processo de preenchimento da incidência do polígono é repetido. Este procedimento é executado tantas vezes quantos forem os contornos, ou seja, até que a lista de arestas temporária se esvazie. Neste momento a lista de arestas é eliminada da memória e a estrutura de dados que representa o contorno está pronta. Este algoritmo é quadrático com relação ao número de nós da fronteira do modelo.

A representação explícita desta nova entidade simplifica bastante a representação do modelo com ou sem a malha de elementos finitos, o que motivou a sua criação, como mostram as Figuras 5.4 e 5.5.

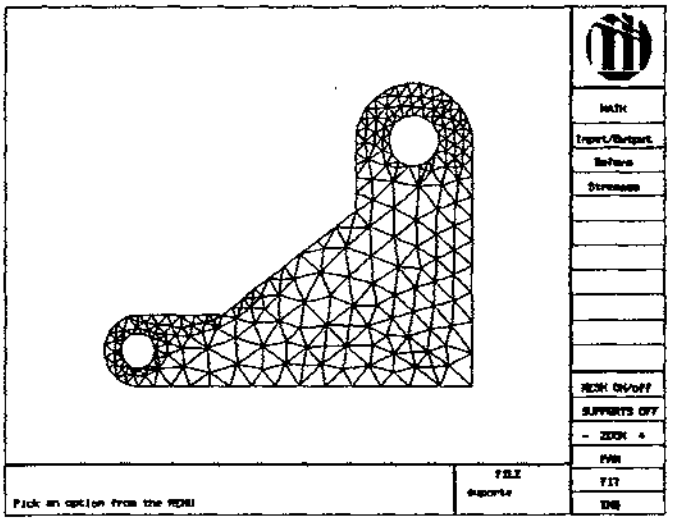

Figura 5.4: Representação do modelo com a malha.

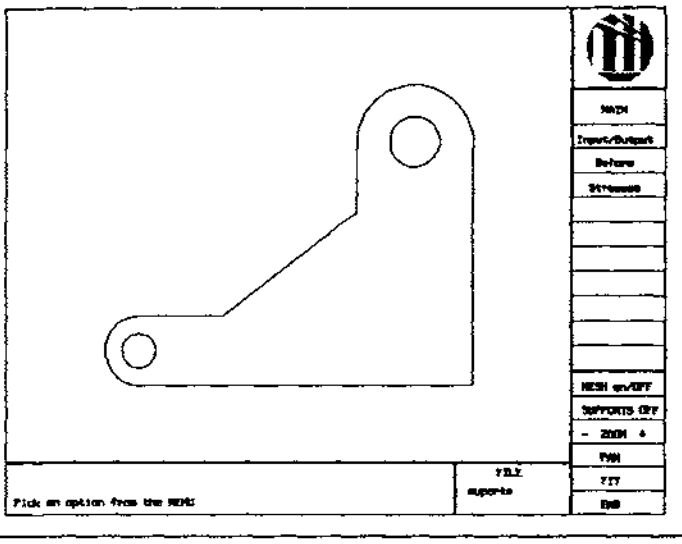

Figura 5.5: Representação do contorno do modelo com a presença de furos.

5.2.2 *Estrutura de dados para informação de elementos adjacentes a nós*

O resultado mais comum de uma análise pelo método dos elementos finitos é o valor nodal das tensões por elemento. Como o resultado é descontínuo, ou seja, para um

mesmo nó, tem-se resultados diferentes nos elementos adjacentes a ele, um procedimento comum em termos de visualização de resultados é o processo de suavização de tensões através da média aritimética das tensões em cada nó, objetivando simular um campo de tensões contínuo.

Quando um determinado campo de tensão é carregado para a memória por elemento e a opção de contorno de tensões estiver ativada, o sistema procede a visualização do campo corrente de forma suavizada ou não de acordo com a solicitação do usuário. O algoritmo responsável por este processo passeia por cada elemento do modelo, obtém suas tensões e as envia ao procedimento que desenha o contorno de tensões de cada elemento (Gattass, et.al., 1990). Para a situação em que o contorno de tensões é visualizado com a suavização ativada, torna-se necessário o cálculo da média das tensões nodais antes de enviá-las para o processo de representação do contorno de tensões. Para proceder o cálculo da média nodal de tensões de forma a não prejudicar a performance do sistema, criou-se a estrutura de dados auxiliar que, dado um nó, sabe-se automaticamente quantos e quais são os elementos adjacentes a ele.

A estrutura é um vetor com dimensão igual ao número total de nós do modelo, onde cada elemento deste vetor aponta para uma lista simplesmente encadeada de elementos, pois não sabe-se de antemão quantos são os elementos adjacentes a cada nó. A Figura 5.6 mostra a estrutura de dados.

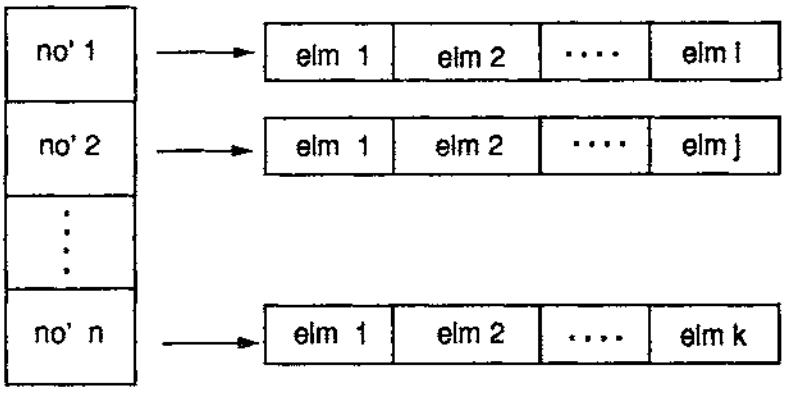

Figura 5.6: Estrutura de dados para armazenar elementos adjacentes aos nós.

O algoritmo que monta esta estrutura de dados cria um vetor onde cada elemento está associado a um nó. Feito isto, passeia por cada elemento finito do modelo e insere seu identificador nas listas correspondentes a cada um de seus nós. Ao final, a estrutura está totalmente preenchida e, portanto, apta a responder às consultas que

motivaram a sua criação. Este algoritmo é de ordem linear com respeito ao número de elementos do modelo.

5.3 *Estratégia de utilização da memória*

Ao ser inicializado, o sistema prepara seus dados para começar a interação, carregando para a memória os dados geométricos e topológicos, criando as estruturas de dados auxiliares e organizando todos os resultados da análise fornecidos (tensões e deslocamentos) em arquivos binários para possibilitar um acesso mais veloz aos dados, já que haveria um excessivo consumo de memória para manter todos os resultados na memória do computador simultaneamente. Portanto, somente um tipo de resultado estará presente na memória por vez, seja ele campo de tensão ou de deslocamento. Sempre que o usuário solicitar a visualização de um outro campo, o sistema libera totalmente a memória utilizada pelo campo anteriormente processado e procede a leitura do novo campo.

6

Técnicas de Visualização Utilizadas

As técnicas de visualização científica devem ser capazes de fornecer informações qualitativas e quantitativas das grandezas processadas. No caso da análise pelo método dos elementos finitos, as técnicas se desenvolvem no sentido de melhor representar o campo de deslocamentos e o de tensões.

O pós-processador bidimensional desenvolvido possibilita a visualização do modelo de elementos finitos e os resultados da análise numérica. Para o campo de deslocamentos, está disponível a deformada do modelo e para as tensões, o caminhamento ou fluxo de tensões nos pontos de integração de Gauss e o contorno de tensões nodais e em pontos de Gauss. Também é possível a visualização do diagrama de tensões ao longo de uma reta definida interativamente pelo usuário sobre o modelo.

Um fato que não pode deixar de ser mencionado é a utilização de um arquivo padronizado para a entrada de dados. A definição deste padrão é fruto de um esforço das pessoas ligadas à linha de pesquisa na PUC-Rio, no sentido de facilitar a utilização dos programas desenvolvidos na área de elementos finitos, sejam eles numéricos, ou gráficos. É conhecido como *arquivo neutro* e tenta ser genérico o suficiente para suportar todas as particularidades envolvidas.

6.1 *O modelo*

Ao se carregar o sistema com o arquivo neutro, o programa automaticamente desenha o modelo na sua condição inicial, ou seja, somente com o contorno desenhado e com as restrições de apoio ativadas, como mostra a Figura 6.1.

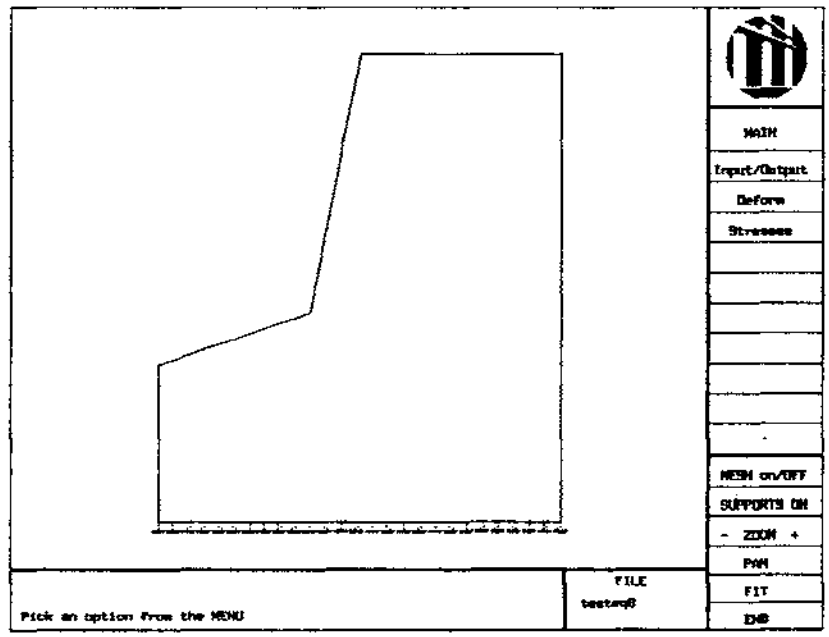

Figura 6.1: Modelo com as características iniciais adotadas pelo pós-processador.

O sistema permite a visualização do conjunto com ou sem a malha e com ou sem as restrições de apoio como mostram as Figuras 6.2 e 6.3.

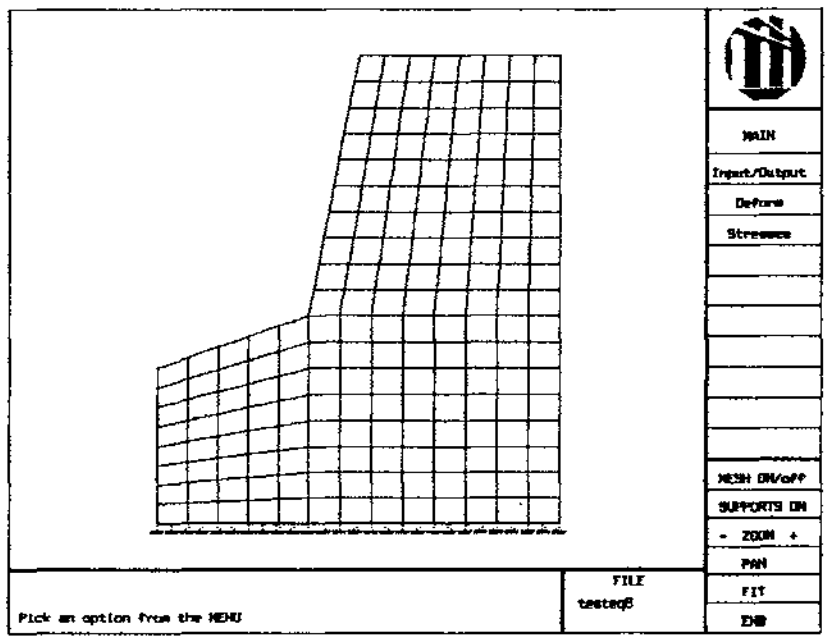

Figura 6.2: Modelo com a representação da malha de elementos finitos.

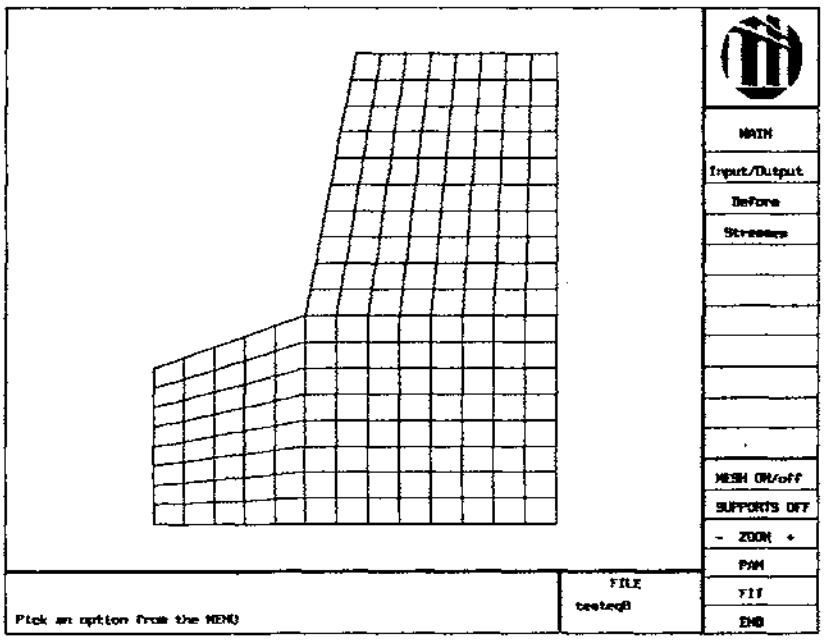

Figura 6.3: Modelo sem a representação das restrições de apoio.

A condição adotada para a visualização do modelo é mantida para representar os resultados da análise até que o usuário as altere.

Outras ferramentas de visualização estão disponíveis como ampliação (*zoom*), translação (*pan*) e enquadramento (*fit*), para facilitar a visualização de detalhes.

6.2 *A deformada*

Ao ter a opção da deformada selecionada, o sistema dispara um processo de limpeza de memória e carrega os deslocamentos nodais provenientes do arquivo binário que os armazena, conforme citado anteriormente. A visualização da deformada pode ser apresentada de duas formas: sozinha ou acompanhada do modelo indeformado, como mostram as Figuras 6.4 e 6.5.

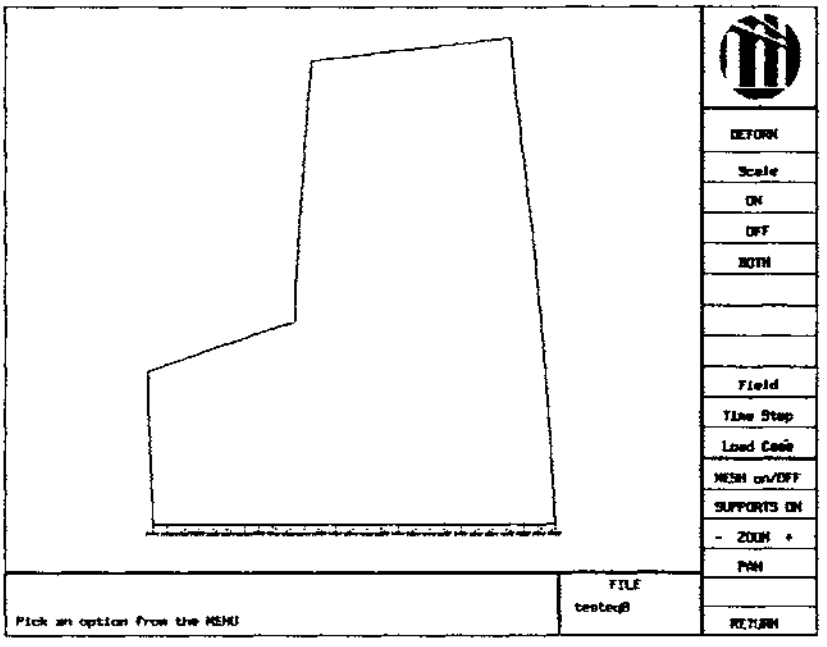

Figura 6.4: Modelo com a configuração deformada.

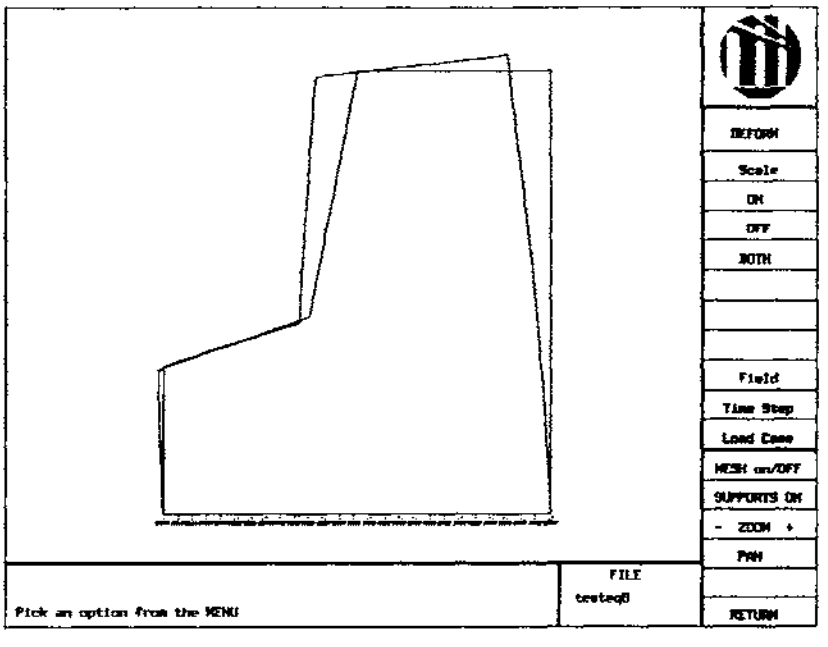

Figura 6.5: Modelo com as configurações deformada e indeformada.

Outro recurso implementado foi a possibilidade de se alterar o fator de escala da deformada como mostra a Figura 6.6.

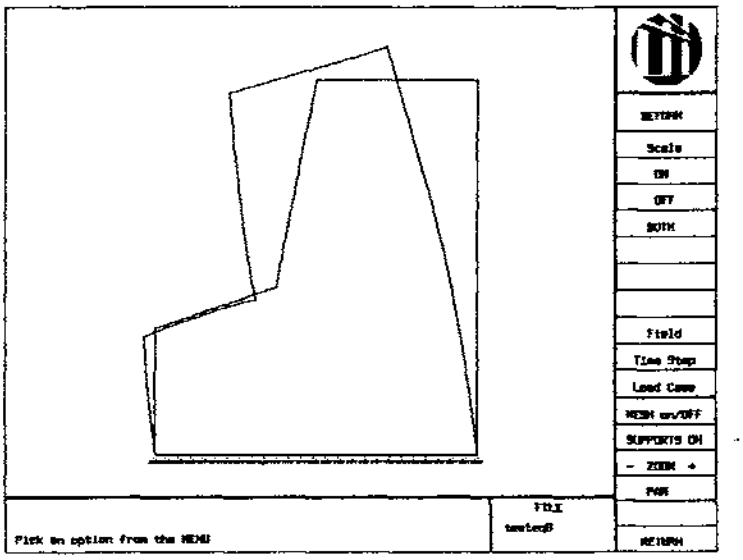

Figura 6.6: Modelo com as configurações indeformada e deformada ampliada.

6.3 *O fluxo de tensões*

O fluxo de tensões é um tipo de representação de resultados bastante útil, pois mostra claramente o caminhamento das tensões do ponto de aplicação do carregamento até os apoios. Este recurso utiliza as tensões principais nos pontos de integração de Gauss. O fluxo é representado através de barras, onde a direção é dada pelo ângulo da tensão principal do campo visualizado naquele ponto e o tamanho é proporcional à magnitude da componente de tensão principal pontual, como mostra a Figura 6.7.

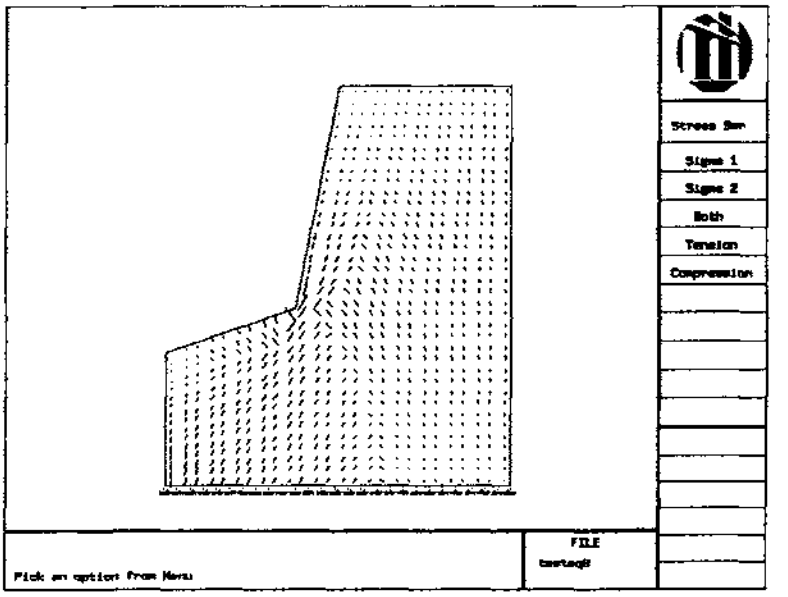

Figura 6.7: Representação do fluxo de tensão principal máxima.

O sistema permite a visualização das tensões principais simultaneamente, quando em cada ponto de integração de Gauss haverá duas barras que representam as tensões máxima e mínima, como mostra a Figura 6.8.

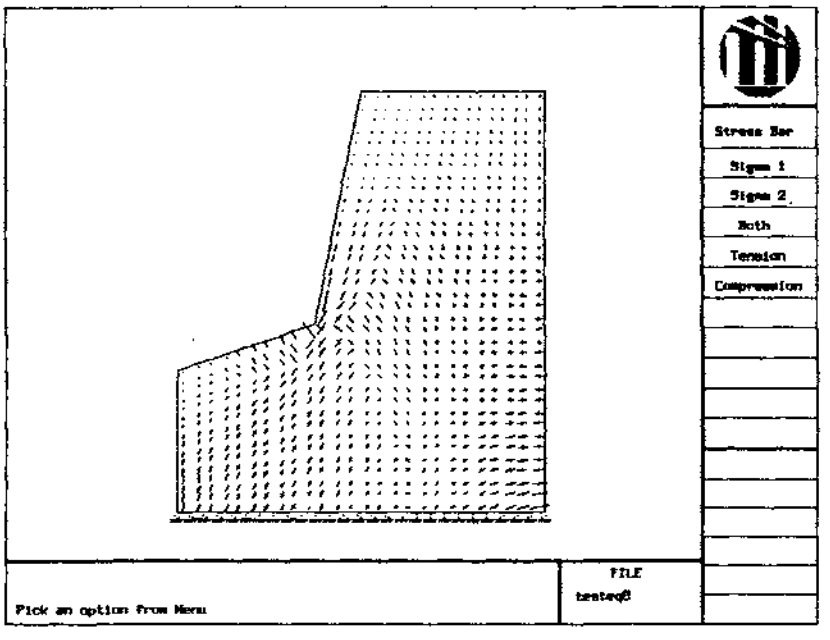

Figura 6.8: Visualização do fluxo das tensões principais máximas e mínimas

Também é possivel a visualização das tensões principais divididas em tensões de tração e compressão como mostram as Figuras 6.9 e 6.10.

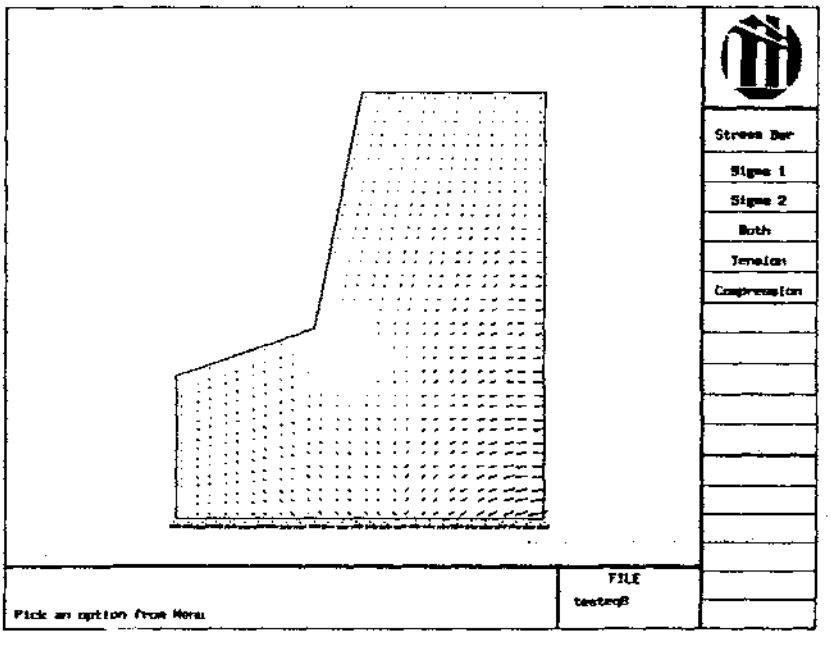

Figura 6.9: Visualização do fluxo das tensões principais de tração.

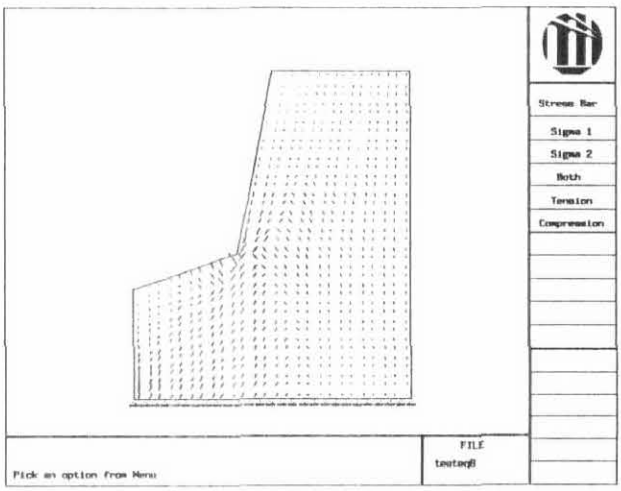

Figura 6.10: Visualização do fluxo das tensões principais de compressão.

6.4 *O contorno de tensões nodais*

Como já foi mencionado no capítulo anterior, o contorno de tensões nodais está disponível com ou sem a suavização dos resultados, como mostram as Figuras 6.11 e 6.12.

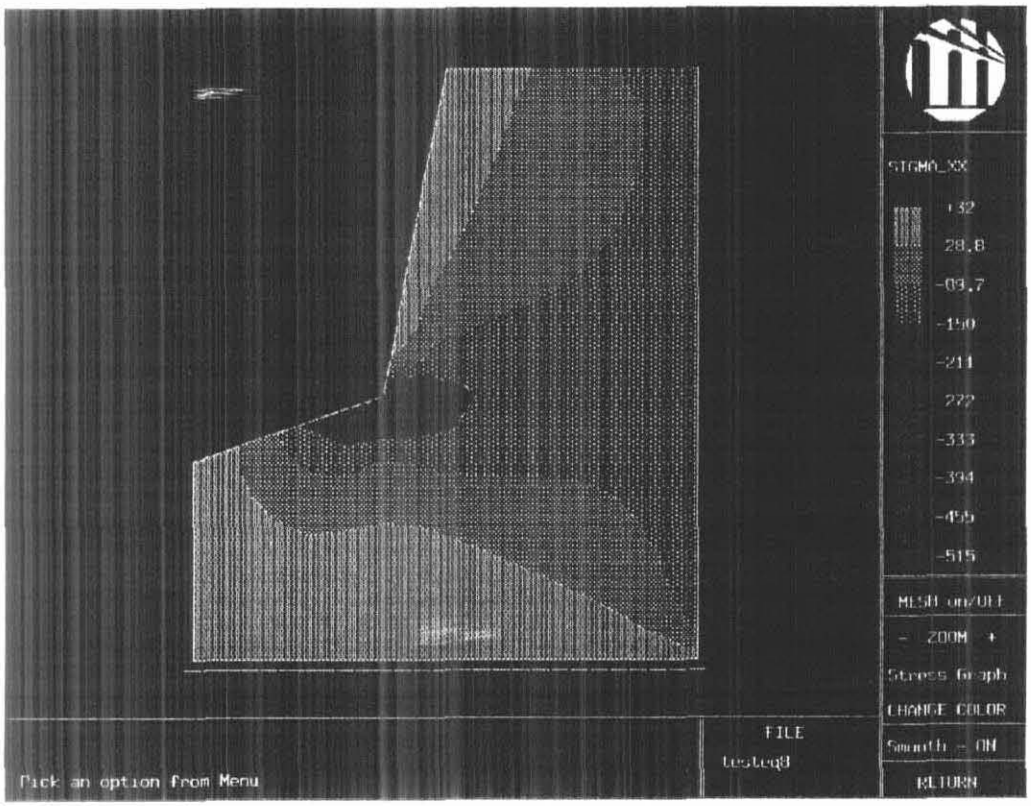

Figura 6.11: Visualização do contorno de tensões nodais suavizado.

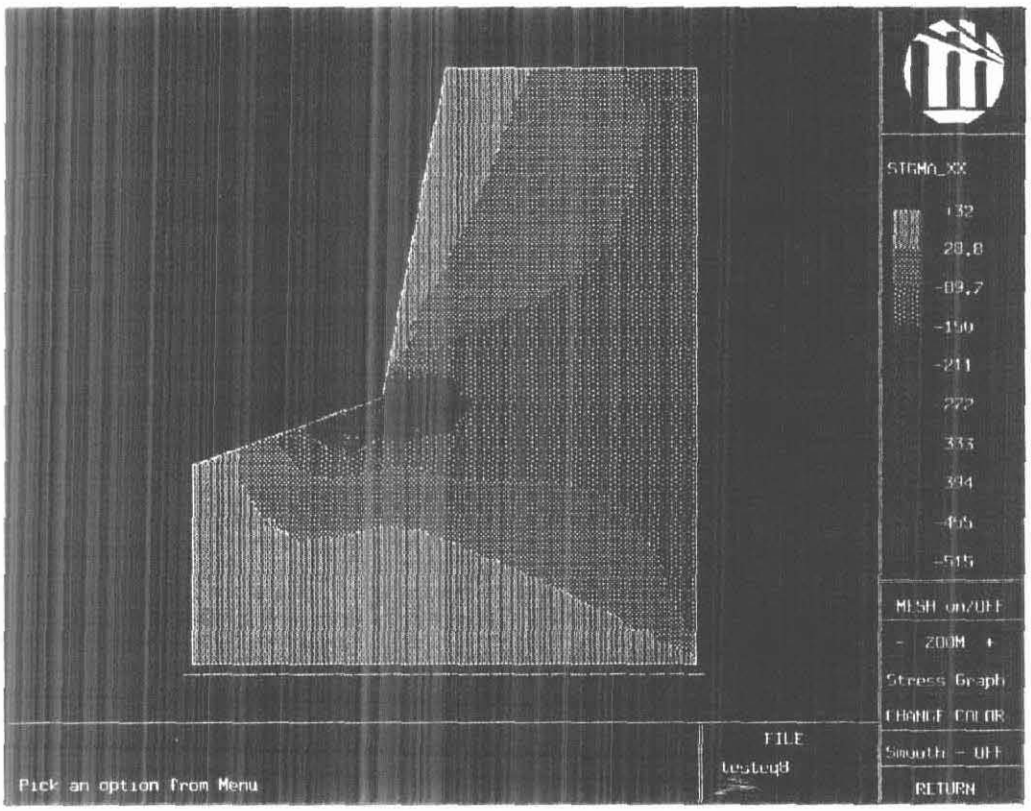

Figura 6.12: Visualização do contorno de tensões nodais não suavizado.

A suavização dos valores nodais de tensão pode ser feita de forma global, para todo o modelo, ou de forma local, por elemento (Hinton, 1974). Usualmente adota-se o procedimento local de extrapolação, o que acarreta uma descontinuidade do campo de tensões na interface dos elementos. Para a suavização das tensões nodais, normalmente calcula-se a média dos valores vindos dos elementos adjacente a um nó.

A extrapolação dos valores obtidos nos pontos de integração para os nós pode ser feita de duas formas: ou os valores são interpolados ou faz-se uma aproximação por um plano ajustado pelo método dos erros mínimos quadrados, metodologia adotada neste trabalho.

Outra questão que deve ser abordada é a da definição da quadratura de Gauss adequada para avaliação de tensões. Hinton (1974) mostra que, quando um polinômio de grau n-1 é usado para aproximar no sentido dos mínimos quadráticos um polinômio de grau n, a solução é exata em alguns pontos. Pode-se mostrar que estes pontos correspondem aos pontos da quadratura de Gauss de n-ésima ordem (que integram exatamente um polinômio de grau 2n-1), justificando, assim, o procedimento de se fixar a ordem de integração para obtenção de tensões para cada tipo de elemento. Deve-se frisar que a ordem de integração para a obtenção da matriz de rigidez é independente da ordem para avaliação de tensões.

O algoritmo utilizado para determinação do contorno de tensões (Gattass et.al., 1991) é executado para cada elemento e é iniciado com a criação de uma tabela com o número de colunas igual ao número de faixas de cores consideradas (cada faixa de cor está limitada por dois valores consecutivos na escala de valores utilizada) onde são armazenadas as coordenadas dos pontos referentes aos polígonos que representam cada faixa de cor no interior do elemento. Também é criado um vetor com a mesma dimensão da tabela onde se guarda o número de pontos de cada polígono que representa uma faixa de tensão. De posse da escala de valores previamente definida, faz-se um passeio pelas arestas do elemento corrente, depositando na tabela as coordenadas dos nós e dos pontos intermediários (pontos de interseção de uma curva de iso-valor com as arestas do elemento, determinados por interpolação linear das tensões nodais) e incrementando o número de pontos do polígono associado à faixa no vetor. Ao final, tem-se a tabela com os pontos que compoem cada polígono na coluna correspondente à sua cor e o vetor com o número de pontos do polígono. De posse destas informações, basta desenhar os polígonos pintando-os com as cores correspondentes.

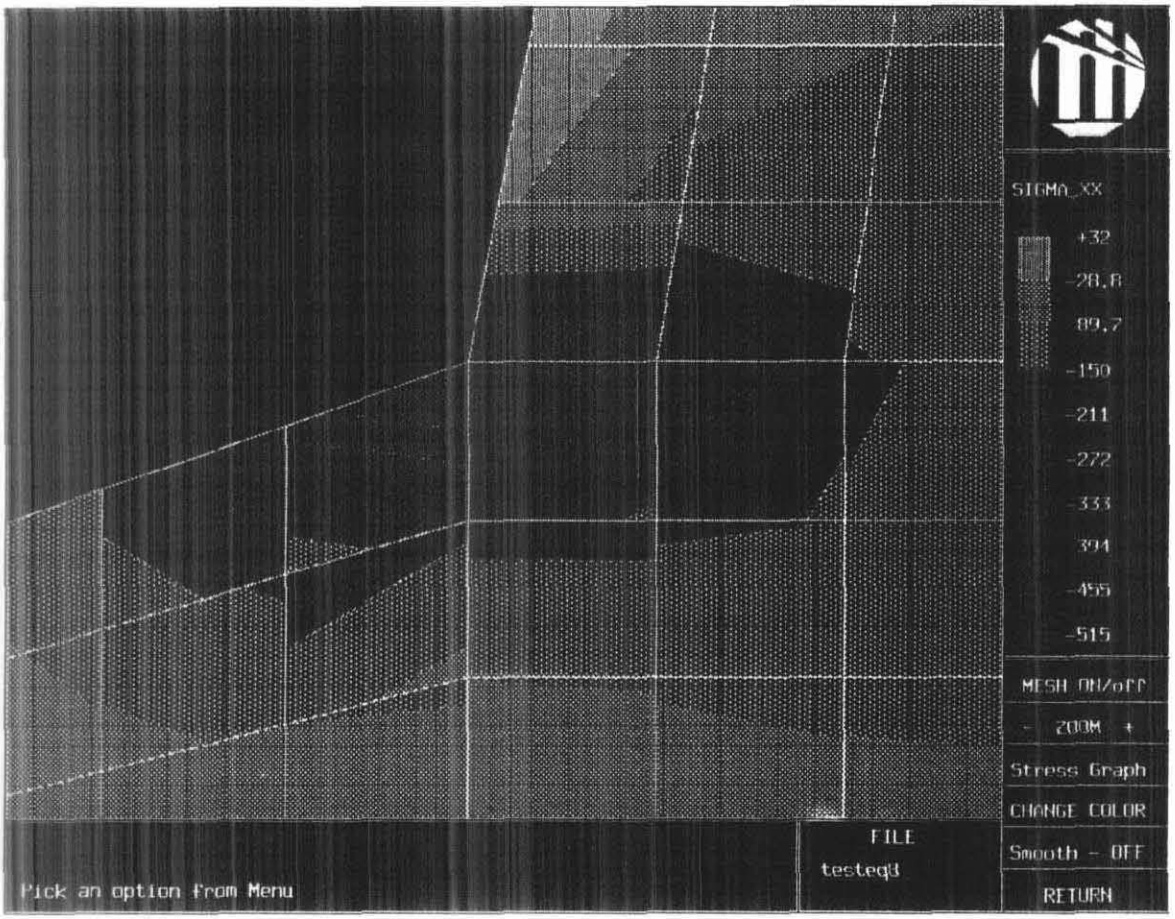

Figura 6.13: Detalhe do contorno não suavizado em região de concentração de tensões.

A visualização do contorno não suavizado é importante para que se possa avaliar as descontinuidades do campo de tensão entre os elementos. Uma descontinuidade brusca pode significar uma necessidade de maior refinamento da malha naquela região. Observa-se que nas regiões onde a discretização é suficiente, o contorno suavizado e o não suavizado tendem a se confundir, como mostram as Figuras 6.11 e 6.12.

A Figura 6.13 mostra em detalhe a região de concentração de tensões sem suavização. A descontinuidade ressaltada pela presença da malha na representação mostra que um refinamento localizado melhoraria o resultado. A Figura 6.14 mostra a mesma região processada com suavização. Observa-se que o resultado aparenta ser de boa qualidade, o que pode acarretar um certo risco.

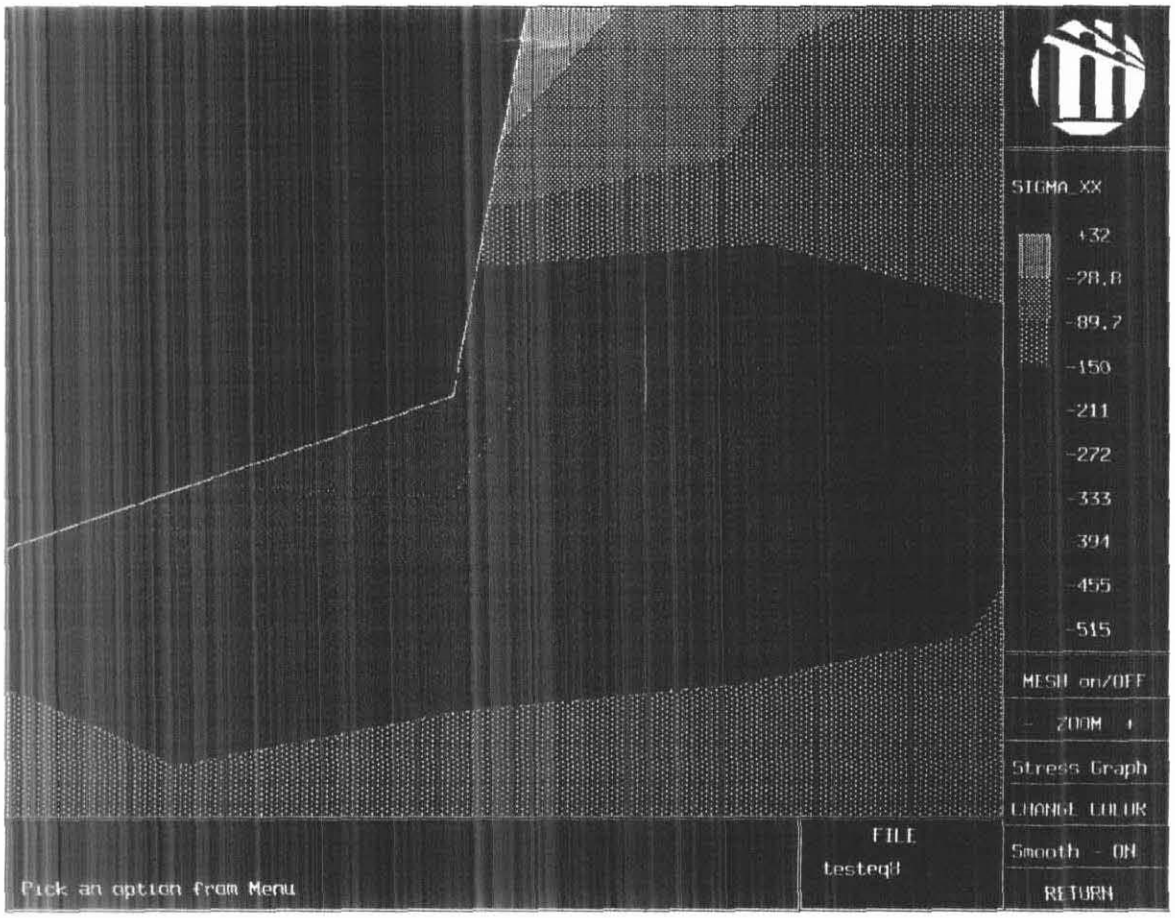

Figura 6.14: Detalhe do contorno suavizado em região de concentração de tensões.

Conclui-se que o processo de suavização deve ser utilizado criteriosamente, já que pode esconder falhas na discretização do modelo.

6.5 *O diagrama de tensões*

Outra ferramenta implementada foi o gráfico 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. O diagrama de tensões acompanha o estado da visualização, ou seja, se o contorno estiver suavizado, o diagrama também estará, senão, apresentará a descontinuidade na interface dos elementos.

Para a montagem do diagrama de tensões, imaginou-se que a linha fornecida pelo usuário forme um plano vertical que intercepta cada um dos planos de extrapolação de tensões (plano que leva as tensões dos pontos de integração de Gauss para os nós do elemento), como mostra a Figura 6.15. O diagrama de tensões é, então, a interseção do plano fornecido com cada um dos planos de extrapolação. Obviamente, no caso em que as tensões nodais não estão suavizadas, este diagrama é descontínuo. Para suavizá-lo, basta que se faça a média nodal nas extremidades da aresta interceptada e depois proceder a interpolação dos valores suavizados.

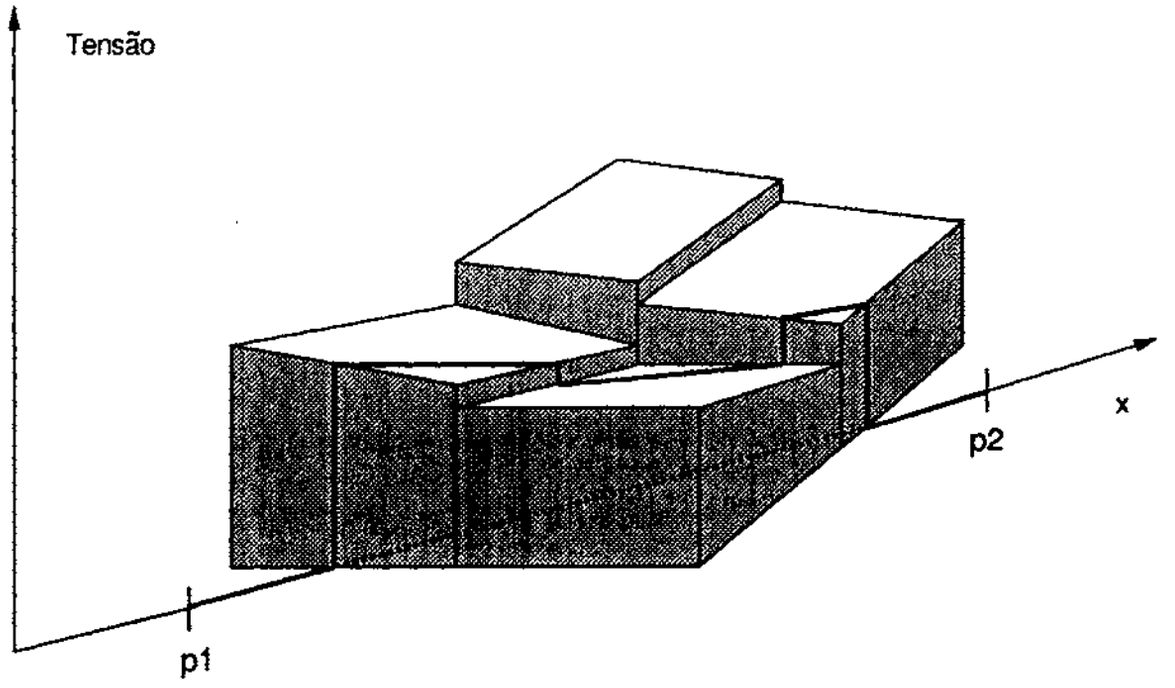

Figura 6.15: Interseção do plano de corte com os planos de extrapolação de tensões.

Apesar do resultado final do diagrama ser o mesmo, a sua determinação não foi feita através da interseção dos planos. Partindo do princípio de que o campo de tensões é admitido como linear em um elemento (é representado por um plano) e, de posse das

tensões nodais, basta que se proceda uma interpolação linear entre os valores nodais para que se determine o valor do diagrama no ponto de interseção entre a linha fornecida e a aresta de cada elemento cortado pela linha, como esclarece a Figura 6.16. A posição exata da interseção da aresta com a reta deve ser calculada com base nas funções de forma dos elementos, já que as arestas podem ser curvas. Além deste procedimento ser um pouco complexo, pois envolve um tratamento não linear, o algoritmo utilizado para representar o contorno de tensões só suporta contornos poligonais. Portanto, por uma questão de simplificação e coerência, adotou-se o procedimento de se considerar as arestas sempre formadas por segmentos retos, não deixando de levar em consideração os nós de meio de aresta para os elementos quadráticos.

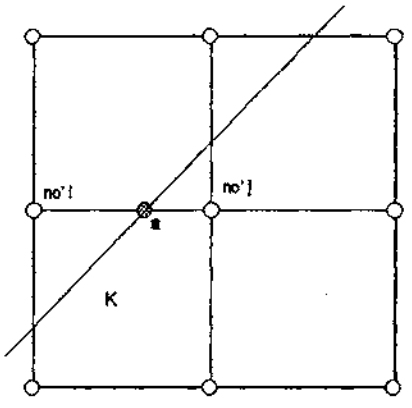

Figura 6.16: Interseção da linha fornecida com as arestas do modelo.

O algoritmo que constroi o diagrama de tensões parte do ponto inicial fornecido pelo usuário e, caminhando pela reta, calcula cada interseção com as arestas atingidas e procede a interpolação. Se o contorno estiver suavizado, a interpolação é calculada com base nos valores nodais suavizados (tarefa facilitada pela estrutura de dados), senão, somente os valores do elemento corrente são utilizados. O resultado final é exemplificado nas Figuras 6.17 e 6.18.

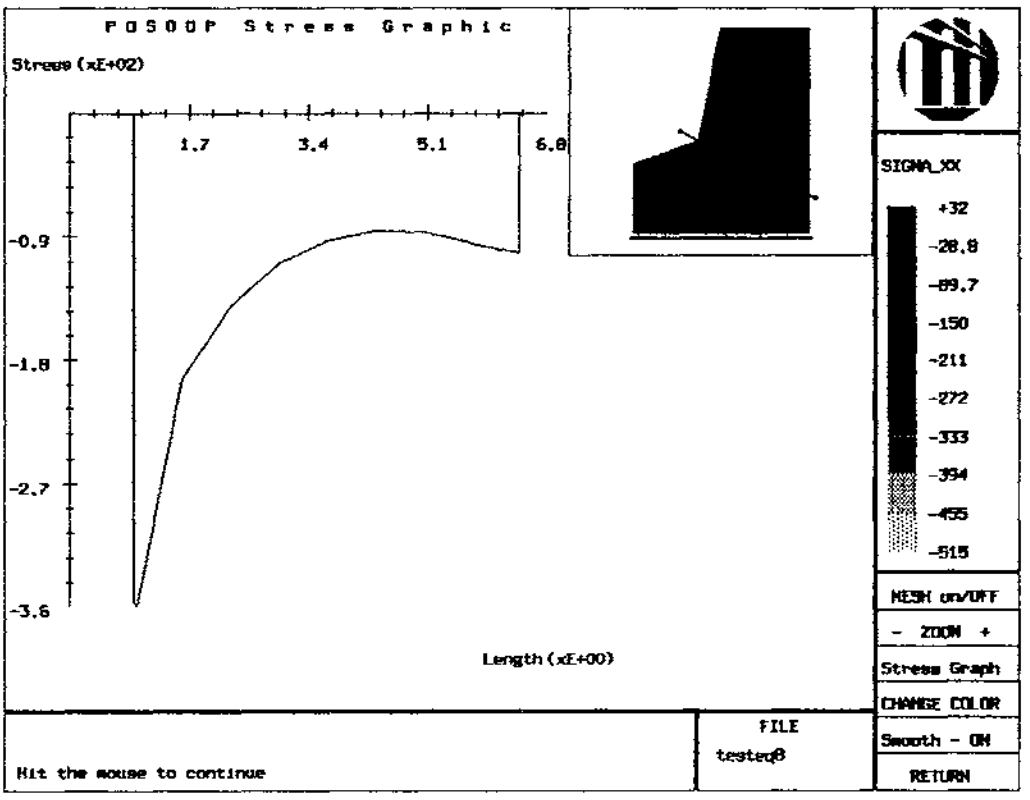

Figura 6.17: Diagrama de tensões referente ao campo suavisado visualizado.

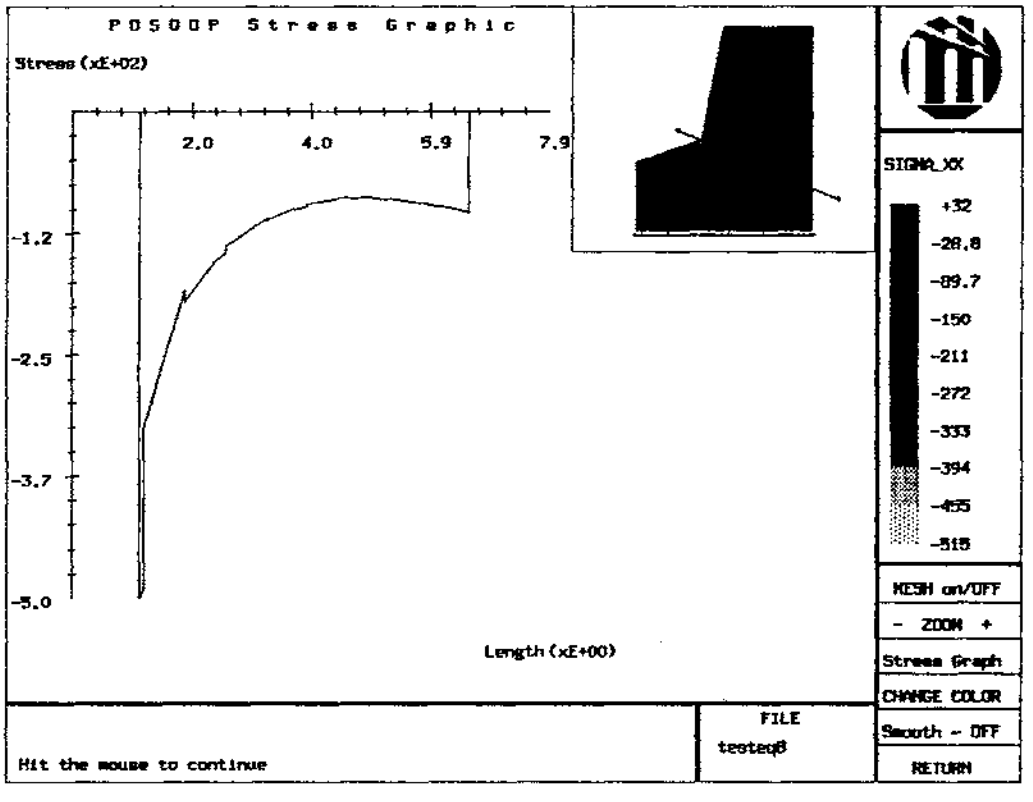

Figura 6.18: Diagrama de tensões referente ao campo não suavisado visualizado.

6.6 *O contorno de tensões em pontos de integração de Gauss*

Conforme visto anteriormente, o algoritmo que processa os dados para o desenho do contorno de tensões necessita receber como dados de entrada um vetor contendo as coordenadas dos pontos de um polígono e outro vetor com os valores do campo a ser interpolado na mesma ordem da incidência do polígono fornecido. No caso das tensões nodais, o polígono é o próprio elemento finito. Porém no caso do contorno de tensões nos pontos de Gauss, os polígonos têm que ser construidos, pois não existem diretamente na estrutura de dados. Para isto, tem-se que criar uma malha auxiliar que passe pelos pontos de integração, desprezando a malha original. Dois fatos devem ser salientados: primeiro, os nós de fronteira devem ser mantidos pois os pontos de integração sempre são internos ao modelo e, segundo, a presença de duas malhas simultaneamente pode comprometer o uso da memória.

Os nós de fronteira não representaram nenhum problema adicional já que o sistema possui uma estrutura de dados auxiliar que armazena os polígonos de contorno (descrita no capítulo 5).

Já a questão da memória é um pouco mais delicada. Para a geração da malha auxiliar interna passando pelos pontos de integração, foram avaliadas duas possibilidades: uma, genérica, que é a triangulação passando por uma nuvem de pontos de Gauss de todos os elementos, desprezando os nós internos e, outra, que é a construção dos polígonos um a um utilizando as informações de adjacência da estrutura de dados. A opção da triangulação possui como vantagem a generalidade, porém o desempenho do algoritmo pode comprometer o processamento interativo. Portanto, os dados criados pela triangulação teriam que permanecer na memória. A construção individual dos polígonos através das informações topológicas contidas na estrutura de dados criada é veloz, pois é linear com o número de elementos, podendo, por isso, cada polígono ser descartado após desenhado. Portanto, este foi o algoritmo adotado.

O único detalhe contra este algoritmo é a perda da generalidade no que diz respeito à ordem de integração para avaliação de tensões. Isto porque é adotada uma quadratura fixa para cada tipo de elemento, de acordo com o trabalho de Hinton (1974), conforme visto anteriormente. Por exemplo, um elemento Q8, independente da ordem de integração adotada para a geração da matriz de rigidez, deve possuir 4 pontos de Gauss para o cálculo das tensões. Como alguns programas não têm a possibilidade de computar a matriz de rigidez e avaliar tensões em quadraturas distintas, isto pode acarretar uma perda de generalidade.

O algoritmo que gera a malha auxiliar passando pelos pontos de integração de Gauss explora a estrutura de dados descrita no capítulo 5. Ele pode ser dividido em três etapas distintas: geração de polígonos internos a cada elemento, ver Figura 6.19, geração de polígonos em torno dos nós internos do modelo passando pelos pontos de Gauss mais próximos (um ponto por elemento), ver Figura 6.20 e, finalmente, a criação de polígonos em todas as arestas internas, como mostra a Figura 6.21.

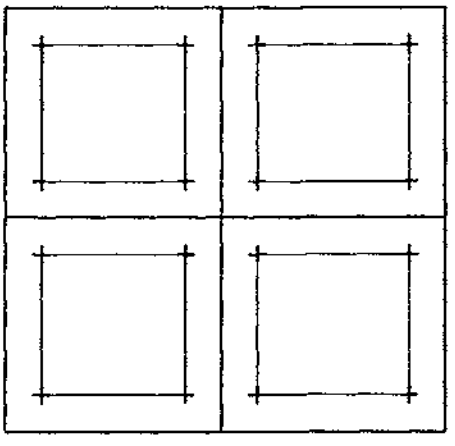

Figura 6.19: Polígonos internos aos elementos.

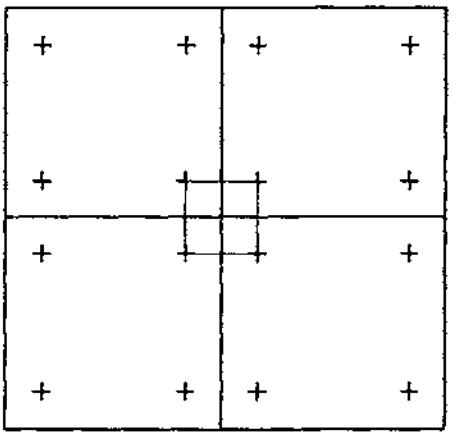

Figura 6.20: Polígonos em torno dos nós internos.

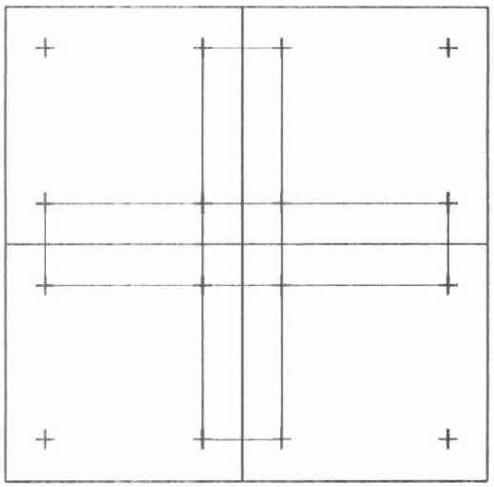

Figura 6.21: Polígonos referentes a cada aresta interna.

Assim como existe a possibilidade de visualização do modelo com ou sem a malha de elementos finitos, optou-se por permitir a representação da malha auxiliar interna que passa pelos pontos de integração de Gauss, como mostra a Figura 6.22.

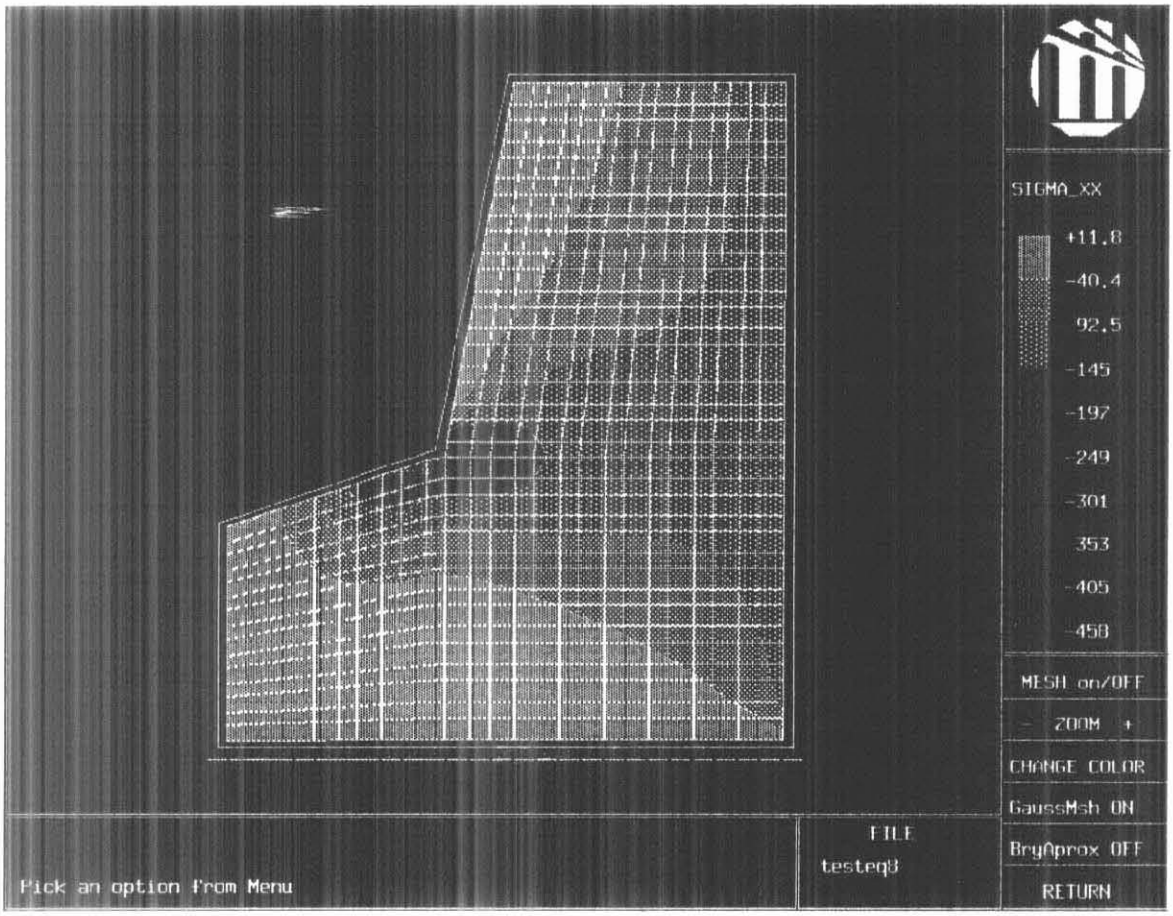

Figura 6.22: Visualização do contorno de tensões em pontos de Gauss com a representação da malha auxiliar gerada.

Como não existem pontos de integração na fronteira, nota-se a ausência de resultados no contorno, explicando a existência de uma faixa sem representação de tensões. Desenvolveu-se um procedimento para complementar a imagem do campo de tensões do modelo que é tratado como aproximação na fronteira. Este algoritmo gera uma outra malha auxiliar que preenche a faixa não representada, ligando os nós da fronteira aos pontos de Gauss dos elementos extremos. Assim como a malha auxiliar interna, os polígonos gerados são descartados logo após a sua representação. Este algoritmo possui duas etapas fundamentais: a primeira é a geração de polígonos em torno de cada nó da fronteira (somente quando é vértice de canto e pertence a mais de um elemento), como mostra a Figura 6.23, e a segunda é a criação de polígonos utilizando a aresta da fronteira e os pontos de Gauss dos elementos da borda, como esclarece a Figura 6.24. As tensões adotadas nos nós do contorno da malha para proceder a aproximação na fronteira são as tensões extrapoladas dos pontos de integração e suavizadas, conforme descrito anteriormente.

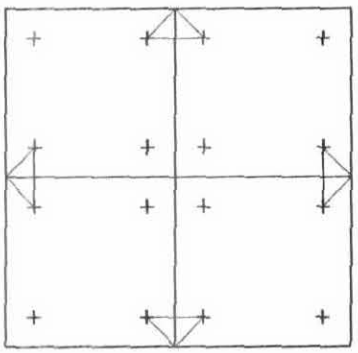

Figura 6.23: Polígonos em torno de cada nó da fronteira.

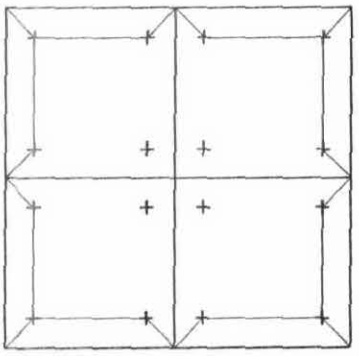

Figura 6.24: Polígonos referentes a cada aresta da fronteira.

A Figura 6.25 mostra um campo de tensões em pontos de Gauss sem aproximação na fronteira e a Figura 6.26 representa um detalhe da região de concentração de tensões, onde também são mostrados os polígonos auxiliares que têm como vértices os pontos de Gauss.

As Figuras 6.27 e 6.28 representam a mesma situação das Figuras 6.25 e 6.26 com a aproximação na fronteira.

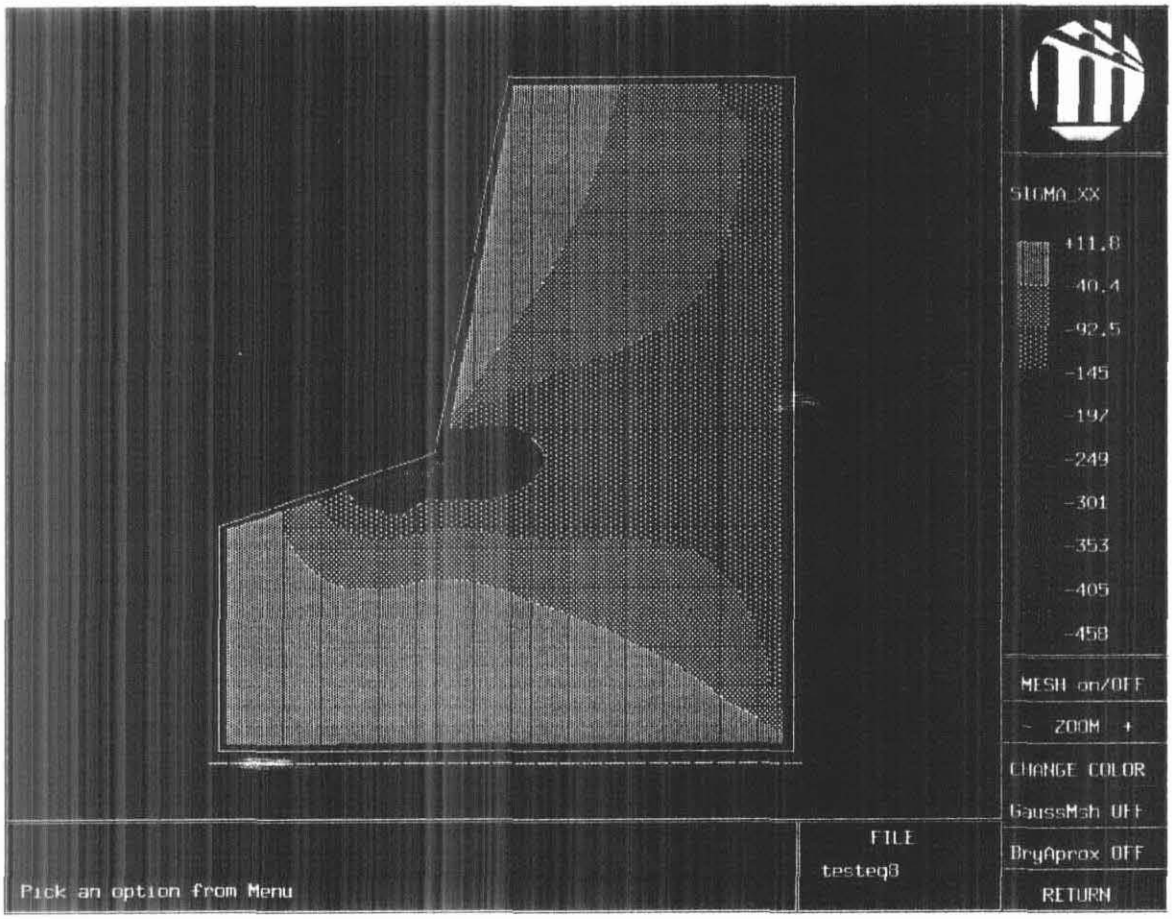

Figura 6.25: Representação do contorno de tensões em pontos de Gauss.

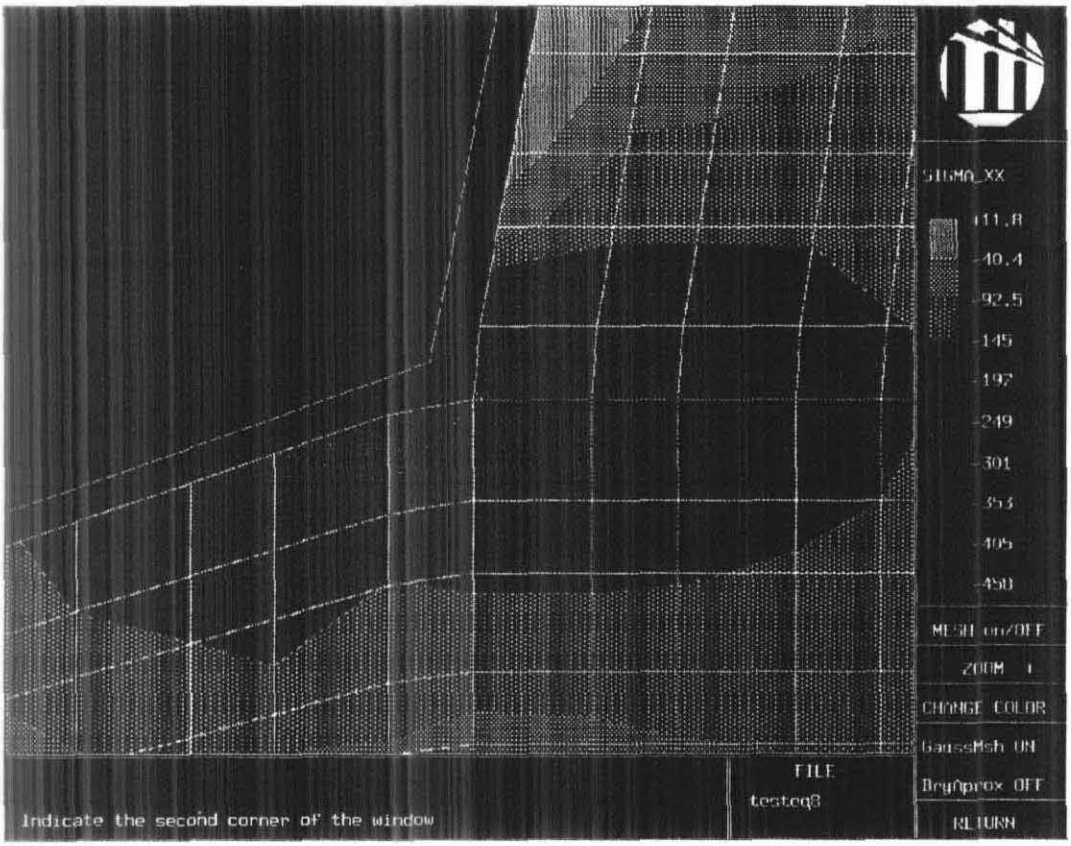

Figura 6.26: Representação do detalhe do contorno de tensões em pontos de Gauss.

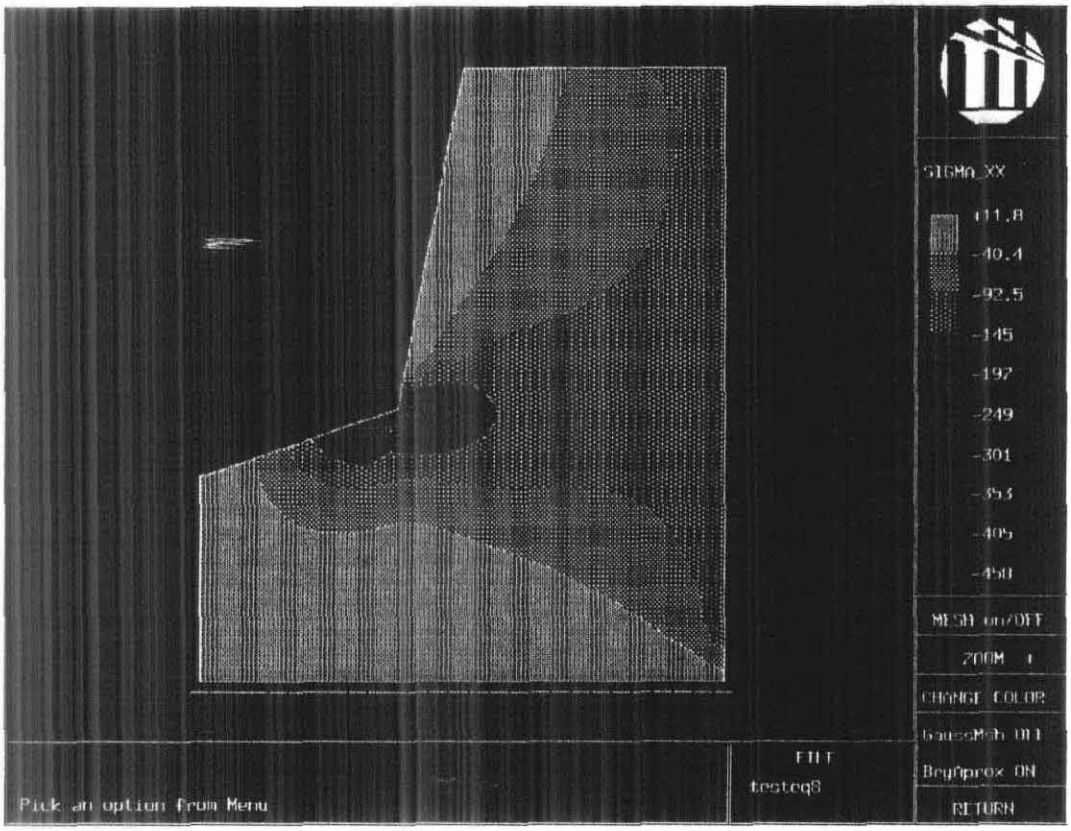

Figura 6.27: Representação do contorno de tensões em pontos de Gauss com aproximação na fronteira.

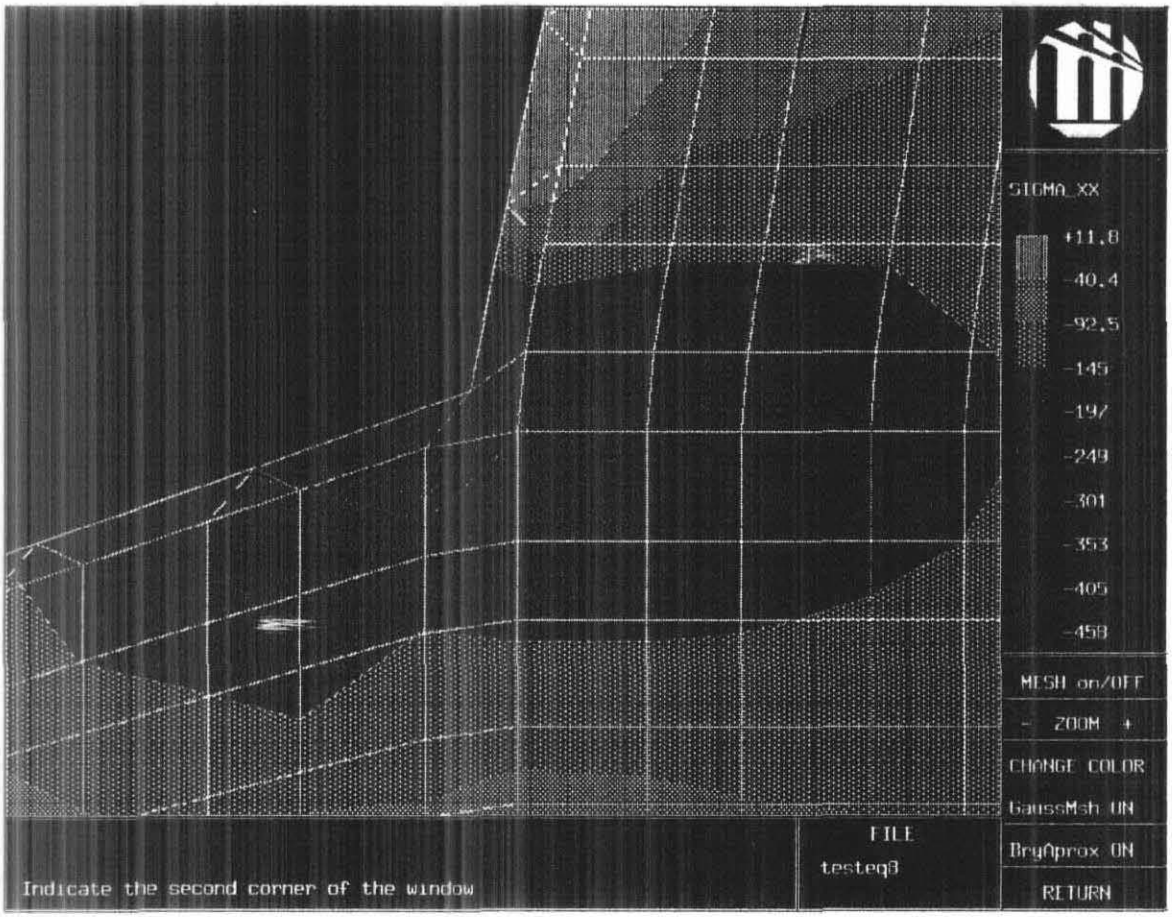

Figura 6.28: Representação do detalhe do contorno de tensões em pontos de Gauss com aproximação da fronteira.

6.7 *Balanço das técnicas de contorno de tensões utilizadas*

Observa-se, analisando os detalhes da região de concentração de tensões representados pelos três tipos de contorno disponíveis, que:

- O contorno de tensões nodais suavizado (Figura 6.14) mostra um resultado aparentemente bom.
- O contorno de tensões nodais não suavizado (Figura 6.13) apresenta descontinuidades bruscas nas interfaces dos elementos.
- O contorno de tensões em pontos de Gauss (Figura 6.26) mostra um foco de tensão em um dos pontos de Gauss, justamente no elemento que mostra o maior gradiente de tensão no contorno nodal não suavizado (Figura 6.13).

A partir destas observações, pode-se concluir que:

- Devido às mudanças bruscas de tensão entre os elementos nesta região, a malha deve ser refinada. Isto comprova a importância de se apresentar contornos nodais não suavizados.

- Há um ganho de qualidade no contorno de tensões em pontos de Gauss. Isto se deve à não existência das etapas de extrapolação e suavização das tensões.
- A aproximação na faixa da fronteira no contorno em pontos de Gauss auxilia na crítica ao nível de discretização da malha pois, onde esta é adequada, a imagem é complementada de forma coerente com a representação de tensões nodais e, nas regiões pouco discretizadas, salienta o problema da malha. A região pouco discretizada neste caso apresenta um foco de tensão em um determinado ponto de Gauss, pois os vizinhos não estão próximos o suficiente para captar o pico de tensão. Quando se procede a aproximação na fronteira, o resultado é a salientação do foco interno, já que o valor utilizado no nó do contorno fica menor, pois recebe a contribuição de valores inferiores dos elementos vizinhos.

7

Conclusões e Sugestões

A utilização da filosofia de orientação a objetos, além de facilitar muito qualquer manutenção ou expansão do sistema, produz um grande impacto nas fases de projeto e desenvolvimento quando comparado a sistemas convencionais.

Quando as classes, objetos, mensagens, estruturas de dados privadas, etc. são elaboradas, surge a obrigação de se pensar genericamente sobre o problema, o que, além de ser muito proveitoso conceitualmente, traz como consequência natural um sistema modularizado. Como somente novos procedimentos devem ser codificados, o ambiente torna-se extremamente favorável à programação incremental. Este fato é de grande interesse neste trabalho, pois o prosseguimento da linha de pesquisa transcorre de uma forma muito mais suave, já que as próximas pessoas envolvidas na continuação do trabalho estarão focadas basicamente em seus novos desenvolvimentos.

Os mecanismos básicos da filosofia de orientação a objetos que foram implementados atingiram os objetivos da proposta deste trabalho. O fato de não se utilizar uma linguagem formal exigiu uma disciplina muito maior a nível de implementação, uma vez que o ambiente híbrido permitia uma flexibilidade muito grande, o que não foi suficiente para penalizar a produção no que diz respeito ao desenvolvimento. Verificou-se que a portabilidade ficou assegurada e a perda de certos formalismos que seriam impostos por uma linguagem formal orientada a objetos foi compensada pela ausência de perda de performance e pelo potencial de expansão que os mecanismos conferiram.

Quanto à portabilidade, obteve-se versões do programa, sem alteração de código, para vários ambientes, como micro-computadores compatíveis com a linha IBM-PC e estações de trabalho baseadas no sistema operacional UNIX (SUN SPARCstation, DEC 5000 e IBM RISC 6000).

Quanto ao potencial de expansão, dois aspectos devem ser salientados: o primeiro, é o crescimento no sentido de se aumentar o número de subclasses na base das organizações das classes existentes e o segundo envolve mudanças mais profundas com a criação de novas superclasses. O primeiro aspecto foi aferido por este trabalho com a numerosa família de elementos e variedade de tipos de análise implementados. Já

o segundo aspecto deve ser avaliado em trabalhos futuros. Já se encontram em andamento dois trabalhos que buscam a expansão do sistema em duas direções. Um, em nível de Mestrado, que implementa análise térmica e outro, em nível de Doutorado, que implementa análise não linear.

Outro aspecto que deve ser salientado é o bom resultado da seleção de classes. A inclusão da classe Modelo de Análise teve fundamental importância na organização global do problema, pois foi o fator que permitiu a consideração de diversos tipos de análise numérica, desde estruturas reticuladas até modelos sólidos.

Quanto ao pós-processador, considera-se que os objetivos foram atingidos. As estruturas de dados tiveram papel fundamental, pois a de adjacência de elementos aos nós viabilizou a implementação interativa do contorno de tensões nodais suavizados, a criação da malha auxiliar passando pelos pontos de Gauss e a criação do diagrama de tensões. Além disto, o armazenamento explícito dos nós da fronteira possibilitou a visualização do modelo com ou sem a representação da malha e a aproximação na fronteira do contorno de tensões em pontos de Gauss.

As técnicas de visualização se mostraram eficazes tanto para representação dos resultados quanto para crítica à discretização do modelo.

O fluxo de tensões mostrou-se uma ferramenta importante, já que pode-se perceber claramente os desvios no caminhamento das tensões dentro do modelo, mostrando a formação de bielas e tirantes.

Quanto ao contorno de tensões em pontos de Gauss, apesar de apresentar um resultado contínuo, ele é capaz de captar problemas na discretização. Em uma região bem discretizada, seu aspecto se assemelha bastante ao do contorno nodal, como esperado. A aproximação na fronteira também se mostrou útil, pois além de compor bem a imagem nas regiões bem discretizadas, acentua os problemas da malha. Portanto, o contorno de tensões nos pontos de Gauss mostrou-se uma boa ferramenta de visualização.

Outra facilidade disponível é o gráfico de tensões que, além de ser traçado em uma linha arbitrária sobre o modelo, é importante tanto no aspecto da sua forma, quanto quantitativamente. O diagrama não suavizado também se mostra eficiente na captação de descontinuidades bruscas de tensão, evidenciando problemas na discretização.

Este trabalho, aliado ao proposto por Vianna (1992), formam um sistema integrado com pré-processamento, análise e pós-processamento, natural para ensino de modelagem por elementos finitos aplicada à elasticidade plana.

Pode-se sugerir para trabalhos futuros:

- Estudo aprofundado sobre a viabilidade de implementar o programa de análise numérica utilizando uma linguagem formal orientada a objetos.
- Expansão da família de elementos no pós-processamento com a inclusão de, por exemplo, elementos infinitos, de interface, Serendipity (com número variado de nós) e elementos de treliça.
- Expansão do pós-processador para visualização de resultados de placas, com inclusão de elementos de barra.
- Exploração do potencial de visualização de grandezas em pontos de integração de Gauss que o pós-processador oferece para adaptá-lo à análise não linear.
- Com respeito ao sistema integrado, sugere-se a implementação de procedimentos de avaliação de erro de modo a possibilitar a redefinição automática da malha, de forma auto-adaptativa, para minimização do erro.
- Extensão do sistema integrado para simular processos de Mecânica da Fratura onde há propagação arbitrária de trincas com redefinição automática da malha.

Referências Bibliográficas

(Alves Filho, Devloo, 1991)

Alves Filho, J.S.R.; Devloo, P.R.B. - "Object-Oriented Programming in Scientific Computations: The Beginning of a New Era", Engineering Computations, 8, pp. 81-87.

(Bailey, 1986)

Bailey, B. C.; Hajjar, J. F.; Abel J. F. - "Towards Effective Interactive Three-Dimensional Colour Post-processing", Eng. Comput., Vol. 3, June.

(Celes Filho, 1990)

Celes Filho, W. - "Um Pós-Processador de Elementos Finitos Sólidos baseado na representação da fronteira dos elementos", Dissertação de Mestrado, PUC-Rio.

(Cook, 1989)

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

(Cox, 1987)

Cox, B.J. - Object Oriented Programming - An Evolutionary Approach, Addison-Wesley, Reading, MA.

(Fenves, 1990)

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

(Forde et. al., 1990)

Forde, B.W.R.; Foschi, R.O.; Stiemer, S.F. - "Object-Oriented Finite Element Analysis", Computer and Structures, 34, pp. 355-374.

(Gattass et. al., 1991)

Gattass, M.; Celes Filho, W.; Fonseca, G.L. - "Computação Gráfica Aplicada ao Método dos Elementos Finitos", Notas do micurso do XIV SBMAC, Nova Fibrurgo, 02 a 05 de setembro de 1991.

(Goldberg, 1983)

Goldberg, A.; Robson, D. - Smalltalk-80: The Language and the Implementation, Addison-Wesley, Reading, MA.

(Hinton, 1974)

Hinton, E. ; Campbell J. S.,"Local and Global Smoothing of Discontinuous Finite Element Functions using a Least Square Square Method", Int.J.Num.Met.Engng., Vol 8, pp.461-480.

(Panthaki,1987)

Panthaki, M., J. - "Colour Postprocessing for Three-Dimensional Finite Element Mesh Quality Evaluation and Envolving Graphical Workstations", dissertção de Mestrado, Cornell University.

(Vianna, 1992)

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

(Wawrzinek,1987)

Wawrzinek, P.A.; - "Interactive Finite Element of Fracture Processes: an Integrated Aproach', Dpt. of Structural Engineering Report, Cornell University, New York.

(Zienkiewicz, 1989)

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

"DISCIPLINA DE PROGRAMAÇÃO ORIENTADA A OBJETOS PARA ANÁLISE E VISUALIZAÇÃO BIDIMENSIONAL DE MODELOS DE ELEMENTOS FINITOS". Dissertação de Mestrado apresentada por LUIZ GIL SOLON GUIMARÃES em 31 de agosto de 1992 ao Departamento de Engenharia Civil da PUC-Rio e aprovada pela Comissão Julgadora, formada pelos seguintes professores:

Prof. Luiz Fernando C. R. Martha (Orientador)
Departamento de Engenharia Civil / PUC-Rio

Prof. Luiz Eloy Vaz
Departamento de Engenharia Civil / PUC-Rio

Prof. Marcelo Gattass
Departamento de Informática / PUC-Rio

Prof. Bruno Feijó
Departamento de Informática / PUC-Rio

Visto e permitida a impressão
Rio de Janeiro, 22/08/94

Coordenador dos Programas de Pós-Graduação
do Centro Técnico Científico