UNIVERSIDADE FEDERAL DE MINAS GERAIS
Escola de Engenharia
Departamento de Engenharia de Estruturas
Curso de Pós-Graduação em Engenharia de Estruturas

PÓS-PROCESSADOR PARA MODELOS BIDIMENSIONAIS NÃO-LINEARES DO MÉTODO DOS ELEMENTOS FINITOS

Samuel Silva Penna

Dissertação apresentada ao curso de Pós-Graduação em Engenharia de Estruturas da UNIVERSIDADE FEDERAL DE MINAS GERAIS, como parte dos requisitos para obtenção do título de MESTRE EM ENGENHARIA DE ESTRUTURAS.

Orientador: Prof. Roque Luiz da Silva Pitangueira

Belo Horizonte
Maio de 2007

UNIVERSIDADE FEDERAL DE MINAS GERAIS
ESCOLA DE ENGENHARIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ESTRUTURAS

"PÓS-PROCESSADOR PARA MODELOS BIDIMENSIONAIS NÃO-LINEARES DO MÉTODO DOS ELEMENTOS FINITOS"

Samuel Silva Penna

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Estruturas da Escola de Engenharia da Universidade Federal de Minas Gerais, como parte dos requisitos necessários à obtenção do título de "Mestre em Engenharia de Estruturas".

Comissão Examinadora:

_____
Prof. Dr. Roque Luiz da Silva Pitangueira
DEES - UFMG - (Orientador)

_____
Prof. Dr. Felicio Bruzzi Barros
DEES - UFMG

_____
Prof. Dr. Rodrigo Weber dos Santos
UFJF

Belo Horizonte, 23 de março de 2007

A imaginação é mais importante que a inteligência

*(*Albert Einstein*)*

Dedico a meus pais.

Sumário

Lista de Tabelas

Lista de Figuras

Resumo

Esta dissertação de mestrado apresenta um programa para pós-processamento de modelos bidimensionais do Método dos Elementos Finitos. O programa é capaz de representar modelos lineares e não-lineares através de recursos gráficos interativos.

O programa faz uso de uma estrutura de adjacência baseada em semi-arestas para manipular as informações geométricas e os dados dos modelos, bem como para permitir a generalização do pós-processador.

A representação de grandezas por isofaixas de valores usa técnicas de subdivisão de domínios, baseadas na triangulação de Delaunay e na própria malha do modelo. Técnicas de extrapolação e suavização de grandezas são disponibilizadas.

A implementação, segundo o paradigma de programação orientado a objetos, usa a linguagem de programação JAVA e diversas APIs (*Application Program Interface*) e pacotes gráficos disponíveis nesta linguagem.

Através da utilização de várias soluções tecnológicas para desenvolvimento de software, a implementação é feita com a progressiva fatoração e a ampliação dos recursos do sistema **INSANE** (*INteractive Structural ANalysis Environment*), um ambiente computacional segmentado e amigável a mudanças. A organização dos aplicativos do sistema, bem como os padrões de projeto de software utilizados, são apresentados, e o projeto orientado a objetos do pós-processador é, então, detalhado.

Diversos exemplos são usados para demonstrar as funcionalidades e principais características do programa.

Abstract

This master's thesis presents a post-processor program for two dimensional Finite Element Method models. The program can represent linear and nonlinear models through graphics interactive resources. The program makes use of an adjacency structure based on half-edges to manipulate the geometrical information and the data of the models, as well to allow the post-processor generalization.

The representation of results by isoranges of values uses domain subdivision techniques, based on Delaunay's triangulation and on the model's mesh. Techniques for results extrapolation and smoothness are available.

The implementation, according to the object-oriented programming paradigm, uses the JAVA programming language and several API's (Application Program Interface) and graphical packages available in this language.

Through the use of various technological solutions for software development, the implementation takes place with the progressive rearrangement and extension of the **INSANE** (*INteractive Structural ANalysis Environment*) system resources, a segmented and friendly to change computational environment . The system applications organization, as well all design patterns used, are present, and the post-processor's object-oriented design is, then, detailed.

Several examples are used to demonstrate the functionalities and main features of the program.

Agradecimentos

A *DEUS* por me dar forcas para conquistar meus objetivos e iluminar a minha vida.

Aos *meus pais*, meus maiores incentivadores, por me apoiarem em todos os momentos.

Ao meu orientador e professor *Roque Luiz da Silva Pitangueira*, que me orientou de forma exemplar, com entusiasmo, dedicação e amizade.

À *todos do projeto Insane* pela amizade, principalmente nas horas de muito estudo.

Aos *professores* e *funcionários* do Departamento de Engenharia de Estruturas da $UFMG$ pela disponibilidade e atenção em todos os momentos, desde a graduação.

Ao *CNPQ* pelo apoio financeiro.

Capítulo 1

INTRODUÇÃO

O grande desenvolvimento do Método dos Elementos Finitos (MEF) vem sendo sustentado por programas processadores capazes de trabalhar com modelos cada vez mais complexos e discretizações cada vez maiores. O uso de diversas tecnologias de desenvolvimento de software tais como, compiladores cada vez mais avançados e IDE's (*Integrated Development Environment*) com recursos automatizados de refatoração e compilação instantânea, em conjunto com padrões de projetos de software sofisticados, permitem que os modelos implementados sejam cada vez mais genéricos.

Tais processadores são, atualmente, auxiliados por geradores de malhas automáticos capazes de operar uma diversidade de elementos e detalhes geométricos, com grande precisão e eficiência. O tratamento geométrico usado pelos pré-processadores generalizam o processo de criação de malhas e a definição do modelo. A riqueza dos detalhes e as informações são passadas automáticamente para a etapa de processamento.

Também é comum o uso de pós-processadores capazes de tratar os dados gerados de forma a representar as grandezas envolvidas no problema com clareza e objetividade, auxiliando o engenheiro analista de estruturas na interpretação de resultados.

1.1 Objetivos

1.1.1 Objetivos Gerais

O projeto **INSANE** (*INteractive Structural ANalysis Environment*), em realização no Departamento de Engenharia de Estruturas da Universidade Federal de Minas Gerais, disponível em *www.dees.ufmg.br/insane*, visa desenvolver softwares na área de métodos numéricos e computacionais aplicados à engenharia. Apoiado em modernos recursos tecnológicos, o projeto trabalha no aprimoramento de modelos, utilizando um ambiente computacional segmentado, escalável em complexidade e amigável a mudanças.

O **INSANE** possui aplicativos implementados em JAVA, com ferramentas de pré-processamento, processamento e pós-processamento de modelos discretos do MEF.

O presente trabalho trata de um aplicativo da etapa de pós-processamento do INSANE, expandindo o sistema sem a necessidade de reescrever a implementação.

1.1.2 Objetivos Específicos

A dissertação tem como objetivo desenvolver um pós-processador para o **INSANE** capaz de representar as grandezas físicas obtidas na análise não-linear de modelos bidimensionais do Método dos Elementos Finitos (MEF).

O pós-processador disponibiliza recursos gráficos interativos diversos através de estrutura de dados para subdivisões planares, técnicas de triangulações de domínios e recursos para suavização, extrapolação e representação de grandezas apropriados.

A implementação segue os padrões de projeto *Model-View-Controller* (MVC), *Command* e *Observer*, o que proporciona uma independência entre os modelos implementados e o pós-processador, facilitando possíveis expansões, tanto dos modelos quanto das interfaces com o usuário.

1.2 Organização do Texto

O trabalho está organizado em 12 capítulos.

No Capítulo 2 são postas em paralelo duas estruturas de adjacências, a de arestas-aladas (ou "*winged-edge*") e a de semi-arestas (ou "*half-edge*"). A estrutura de arestas-aladas é abordada a fim de introduzir a idéia de organização em arestas e maiores detalhes são dedicados à estrutura de semi-arestas.

No Capítulo 3 é feita uma breve apresentação sobre geometria computacional, fundamental para o desenvolvimento do programa, e uma abordagem sobre divisão de domínios por diagramas de Voronoi dando base para a triangulação de Delaunay.

A geometria computacional continua no Capítulo 4, que apresenta as transformações geométricas, fundamentais para os diversos recursos de visualização, disponibilizados no pós-processador.

O Capítulo 5 expõe diversas técnicas de suavização de grandezas, desde os métodos simples, como média nodal, até os mais sofisticados. A combinação de métodos usada no pós-processador é, então, apresentada.

O Capítulo 6 trata dos métodos de representação de resultados. Primeiramente são apresentados métodos para o traçado de isocurvas e em seguida métodos para isofaixas. Por fim é detalhado o método usado no programa que baseia-se em uma técnica de subdivisão de domínios.

A fim de contextualizar o pós-processamento e a análise não-linear, o Capítulo 7 abrange questões que são vistas na representação de resultados e ao longo dos exemplos expostos neste trabalho.

O Capítulo 8 discute os padrões de projeto de softwares usados na implementação. São abordados os padrões *Modelo-View-Controller* (MVC), *Command* e *Observer*. O Capítulo finaliza com a integração destes padrões para uso no sistema como um todo.

O Capítulo 9 apresenta o projeto orientado a objetos do pós-processador e a

organização do núcleo numérico do **INSANE**, relacionando-os. Destaca-se a organização de classes, a hierarquia entre elas, as dependências e as associações das principais instâncias. Neste capítulo os padrões de projeto são caracterizados no contexto do pós-processador.

O Capítulo 10 expõe os recursos disponibilizados e o funcionamento do programa através de exemplos.

Os recursos já apresentados são melhor explorados no Capítulo 11, que apresenta diversos exemplos que ilustram as principais características do pós-processador.

O Capítulo 12 apresenta considerações sobre as principais contribuições deste trabalho para a etapa de pós-processamento de modelos do MEF, detalhando os recursos agregados ao sistema **INSANE**, e lista sugestões de possíveis aprimoramentos do mesmo.

Capítulo 2

ESTRUTURAS DE DADOS PARA SUBDIVISÃO PLANAR

2.1 Introdução

O desenvolvimento de modelos geométricos requer o armazenamento de dados, que possibilite a manipulação de construções e operações geométricas, de forma estruturada e acessível. Tendo isto em vista, são usadas estruturas de adjacências no tratamento de modelos computacionais.

As estruturas de dados para subdivisões planares, ou simplesmente estruturas de adjacências, são uma forma organizada de armazenamento de dados de modelos geométricos computacionais, possibilitando um rápido acesso a estes dados posteriormente.

Este capítulo aborda inicialmente a estrutura de "arestas-aladas" ou *winged-edge*, para em seguida apresentar a estrutura de "semi-arestas" ou *half-edge*.

2.2 Estrutura de Dados de "Arestas-Aladas" ou "Winged-Edge"

A estrutura de "arestas-aladas" tem as arestas de uma subdivisão planar como a principal informação do modelo geométrico, embora também faça uso dos vértices e

faces para armazenamento de dados.

Nesta estrutura, os vértices, as faces e as arestas são organizados em listas, sendo cada lista preenchida de acordo com o tipo de elemento a ser representado. A lista de vértices armazena as coordenadas (x, y, z) e uma das suas arestas (av) incidentes nos respectivos vértices. A lista de faces armazena somente uma de suas arestas (af) e os seus vértices (ver tabela 2.2).

As arestas são os elementos que armazenam o maior número de dados. Cada uma das arestas é representada por um par de vértices, que são informados na ordem de orientação da aresta, como pode-se ver na figura 2.1 (vértices $\mathbf{v}_1$ e $\mathbf{v}_2$). A orientação da aresta proporciona a localização das faces adjacentes, sendo a face à esquerda à aresta chamada de **face counterclockwise** ou **fccw**, pois o sentido da aresta nesta face é anti-horário. Por sua vez, a face à direta é chamada de **face clockwise**, representada por **fcw**, que possui sentido horário de acordo com a orientação da aresta. Seguindo a mesma representação, são guardadas na lista das arestas, as arestas anteriores e posteriores incidentes nos vértices da aresta analisada. Portanto, tem-se **previous counterclockwise** (**pccw**), e **next counterclockwise** (**nccw**), as arestas anterior e posterior no sentido da aresta, **previous clockwise** (**pcw**) e **next clockwise** (**ncw**), no sentido contrário ao da aresta. A tabela 2.3 ilustra os dados armazenados nas arestas.

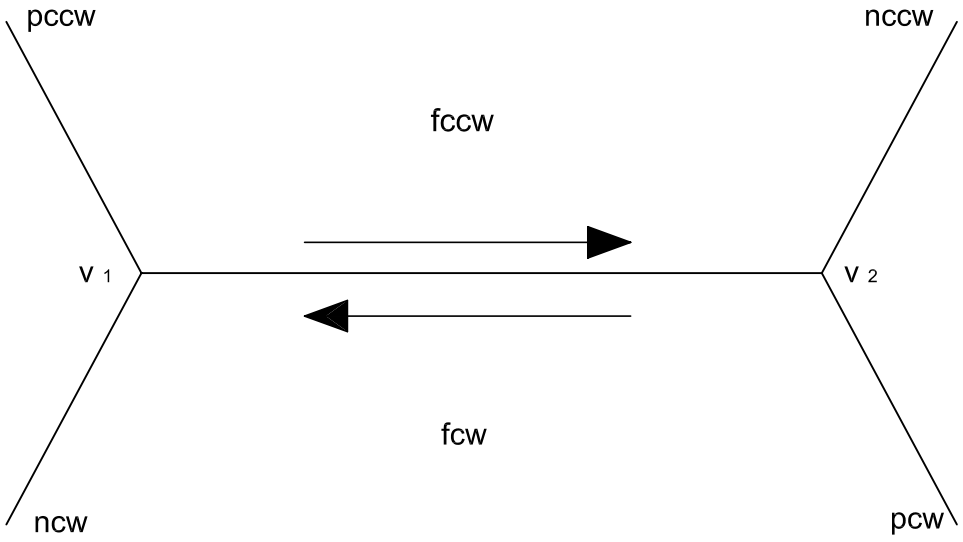


Figura 2.1: Estrutura *winged-edge*.

A estrutura de arestas aladas, como já dito, armazena dados essenciais dos vértices, das faces e das arestas, para compor a estrutura de uma subdivisão planar. Para uma melhor compreensão desta estrutura de dados, as tabelas 2.1, 2.2 e 2.3 mostram os dados para o caso da subdivisão planar do cubo unitário na figura 2.2 (Figueiredo e Carvalho, 1991).

Tabela 2.1: Vértices

Vértices	**Coordenadas**	av
v_1	(1,0,0)	a_1
v_2	(1,1,0)	a_2
v_3	(1,1,1)	a_3
v_4	(1,0,1)	a_4
v_5	(0,0,0)	a_5
v_6	(0,1,0)	a_6
v_7	(0,1,1)	a_7
v_8	(0,0,1)	a_8

Tabela 2.2: Faces

Faces	af
F_1	a_1
F_2	a_2
F_3	a_{10}
F_4	a_{12}
F_5	a_1
F_6	a_8

Tabela 2.3: Arestas

Arestas	v_1	v_2	**fccw**	**fcw**	**pccw**	**nccw**	**pcw**	**ncw**
a_1	v_1	v_2	F_1	F_5	a_4	a_2	a_6	a_5
a_2	v_2	v_3	F_1	F_2	a_1	a_3	a_7	a_6
a_3	v_3	v_4	F_1	F_6	a_2	a_4	a_8	a_7
a_4	v_4	v_1	F_1	F_4	a_3	a_1	a_5	a_8
a_5	v_1	v_5	F_5	F_4	a_1	a_9	a_{12}	a_4
a_7	v_3	v_7	F_6	F_2	a_3	a_{11}	a_{10}	a_2
a_8	v_4	v_8	F_4	F_6	a_4	a_{12}	a_{11}	a_3
a_9	v_5	v_6	F_5	F_3	a_5	a_6	a_{10}	a_{12}
a_{10}	v_6	v_7	F_2	F_3	a_6	a_7	a_{11}	a_9
a_{11}	v_7	v_8	F_6	F_3	a_7	a_8	a_{12}	a_{10}
a_{12}	v_8	v_5	F_4	F_3	a_8	a_5	a_9	a_{11}

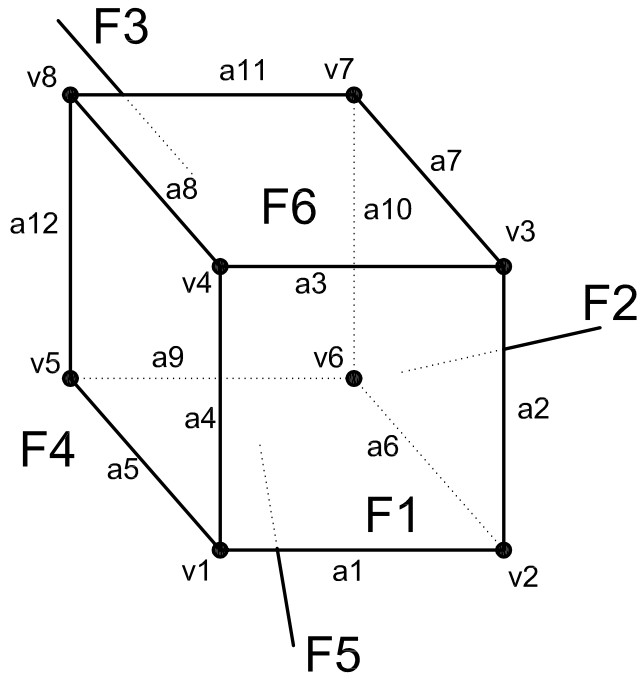


Figura 2.2: Cubo unitário.

2.3 Estrutura de Dados "*Half-Edge*" ou de "Semi-Arestas"

A estrutura de semi-aresta, analogamente à estrutura "*winged-edge*", permite armazenar informações relativas a uma subdivisão planar, de modo que estas proporcionem a relação de adjacência entre as faces do modelo. A estrutura de semi-arestas é uma variação da estrutura de "arestas-aladas", vista anteriormente, diferenciando-se pela sua organização topológica. Conforme ilustra a figura 2.3, a principal alteração é a transformação de uma aresta em duas semi-arestas.

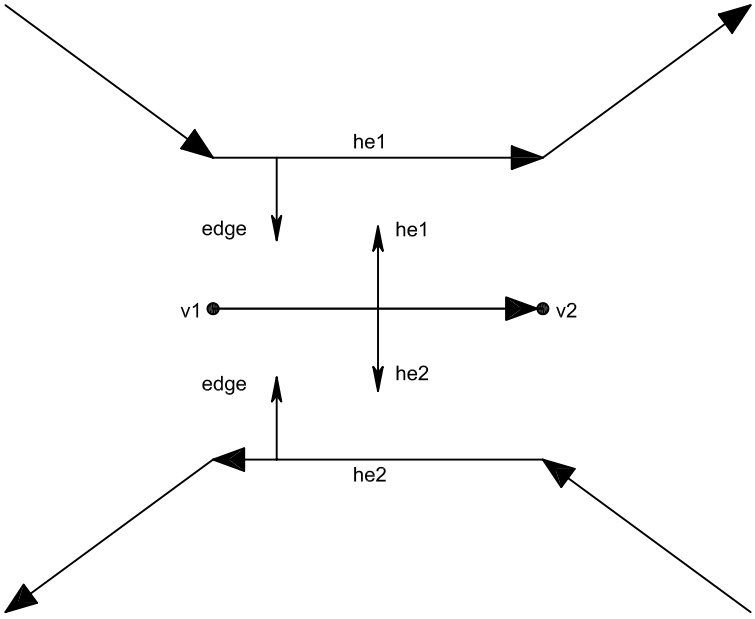


Figura 2.3: Estrutura "*Half-Edge*".

Esta estrutura de dados se desenvolve em uma hierarquia, composta por cinco níveis diferentes, sendo eles a ***Subdivisão Planar***, a ***Face***, o ***Loop***, a ***Semi-Aresta*** e o ***Vértice***, conforme ilustra a figura 2.4.

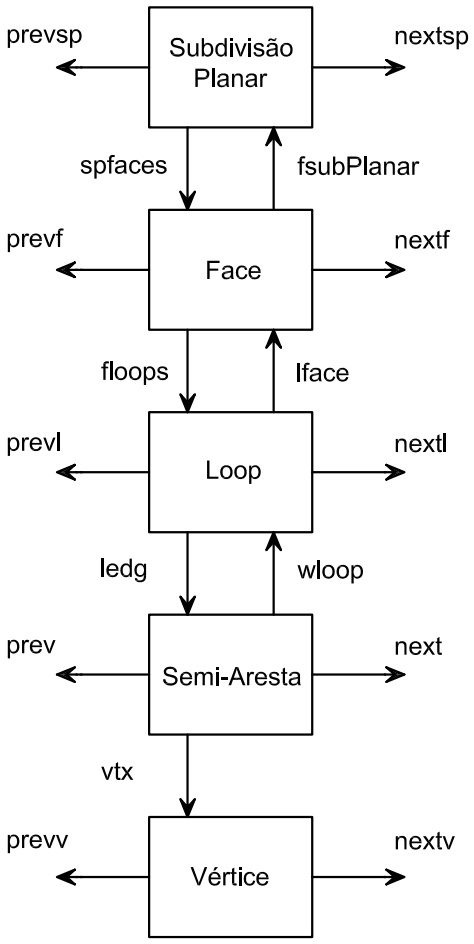


Figura 2.4: Hierarquia de Semi-Arestas.

No topo da hierarquia tem-se o nível ***Subdivisão Planar***. Este possui uma referência que permite o acesso ao nível ***Face***. A ***Subdivisão Planar*** também possui uma lista duplamente encadeada de modelos geométricos.

Em seguida, o nível ***Face*** representa as faces da subdivisão planar. Em um modelo plano, bem como em um modelo sólido, composto por várias faces, é necessário uma lista duplamente encadeada relativa às várias faces do modelo. Cada face é formada por "*loops*" que compõe seu contorno, sendo necessário uma referência para o acesso ao nível ***Loop***. Uma face pode ser composta por vários "*loops*" e estes podem ser externos, representando o contorno da face, e ou internos, representando furos na face. Por fim, tem-se que cada face possui uma referência para a sua respectiva

Subdivisão Planar.

O nível seguinte é o ***Loop***. Como uma face pode ser composta por vários "*loops*", é necessário uma referência para ***Face*** e uma lista duplamente encadeada de "*loops*". O ***Loop***, por sua vez, é composto por semi-arestas e logo, possui uma referência para ***Semi-Aresta***.

O nível ***Semi-Aresta*** possui uma referência para o ***Loop*** e possui também uma lista duplamente encadeada, respectiva às semi-arestas do modelo. Também possui uma referência para ***Vértice***, que é o vértice de início da respectiva semi-aresta. As semi-arestas também são armazenadas em uma lista duplamente encadeada.

O último nível é o ***Vértice***, com uma lista duplamente encadeada para os vértices do modelo.

A estrutura de adjacência como representada até agora não faz conexão entre as faces com a finalidade de compor o modelo geométrico. Para tanto, basta adicionar um nível intermediário, conforme ilustrado na figura 2.5.

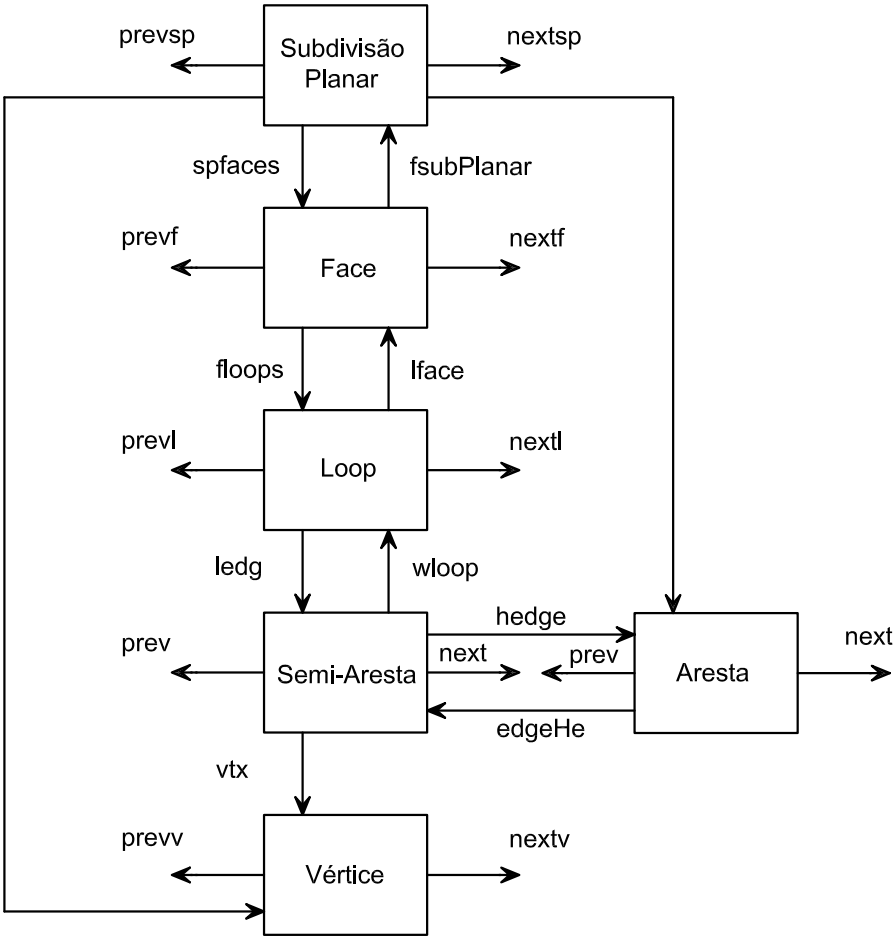


Figura 2.5: Estrutura de dados de Semi-Arestas .

Este nível é a ***Aresta***, que possui uma referência para ***Semi-Aresta***, uma vez que a aresta é composta por duas semi-arestas, uma à esquerda e uma à direita, incidentes nos mesmos vértices. As arestas também ficam armazenadas em uma lista duplamente encadeada. Com esta alteração, a ***Subdivisão Planar*** passa a ter uma referência para ***Aresta*** e ***Vértice***, e a ***Semi-Aresta*** uma referência para a sua respectiva aresta.

O armazenamento dos dados de um modelo geométrico na estrutura acima discutida permite o acesso aos elementos do modelo, por consulta simples, seguindo a ordem da hierarquia. Para a subdivisão planar detalhada na figura 2.6 a estrutura de semi-arestas correspondente está mostrada nas tabelas 2.4, 2.5, 2.6 e 2.7.

Pode-se observar que o mesmo cubo unitário da figura 2.2 é agora detalhado

em sua forma planar (2.6-b), sendo especificado suas respectivas faces, vértices e arestas. Na figura 2.6-c tem-se a composição dos "*loops*" das faces e na figura 2.6-d a definição das semi-arestas.

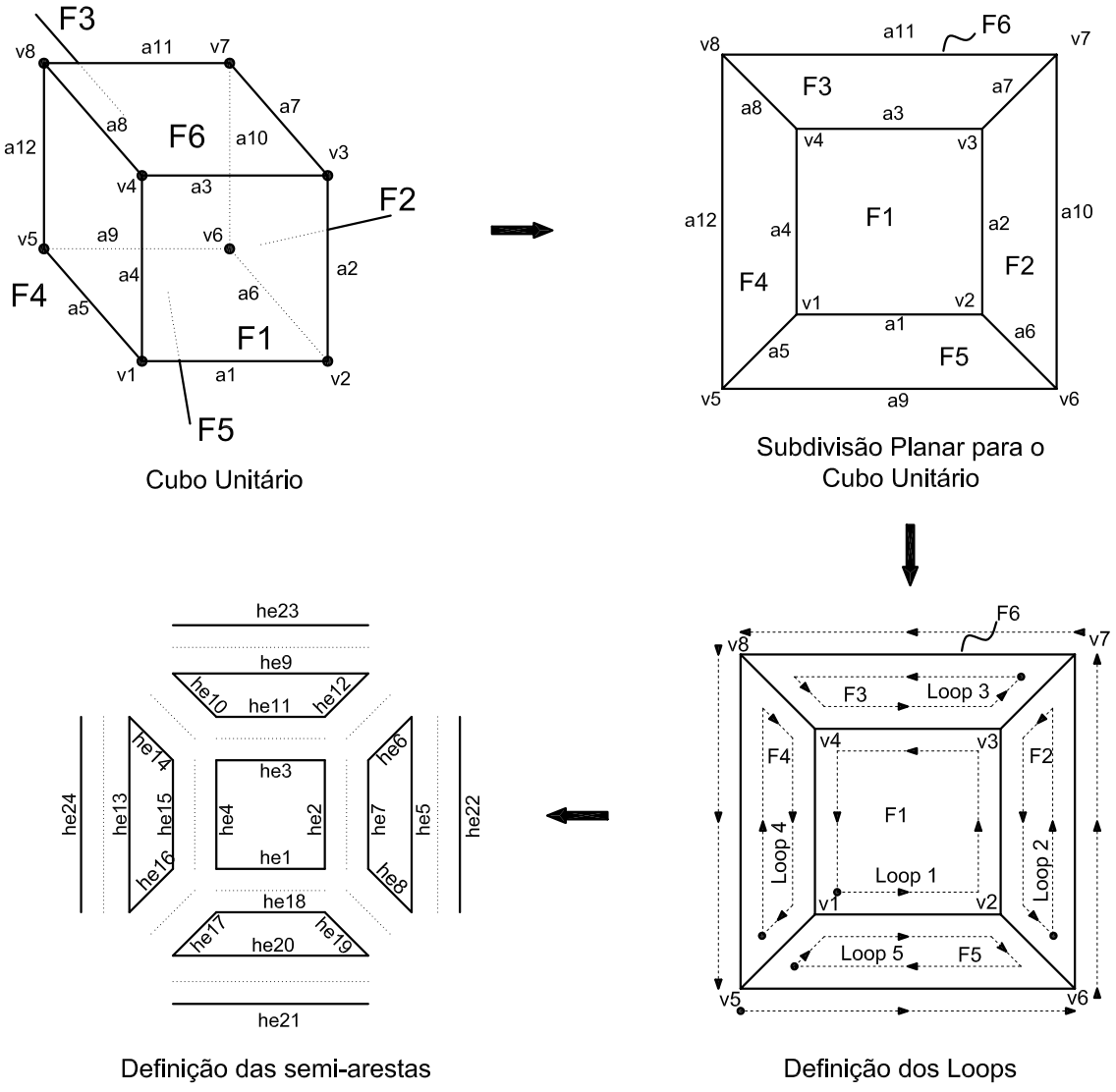


Figura 2.6: Detalhamento de uma Subdivisão Planar para um cubo unitário

Tabela 2.4: Subdivisão Planar

Faces	**Arestas**	**Vértices**
F_1	a_1	v_1
F_2	a_2	v_2
F_3	a_3	v_3
F_4	a_4	v_4
F_5	a_5	v_5
F_6	a_6	v_6
	a_7	v_7
	a_8	v_8
	a_9	
	a_{10}	
	a_{11}	
	a_{12}	

Tabela 2.5: Faces e Loops

Face	**Loop**	**Semi-Aresta**
1	$Loop_1$	he_1, he_2, he_3, he_4
2	$Loop_2$	he_5, he_6, he_7, he_8
3	$Loop_3$	he_9, he_{10}, he_{11}, he_{12}
4	$Loop_4$	he_{13}, he_{14}, he_{15}, he_{16}
5	$Loop_5$	he_{17}, he_{18}, he_{19}, he_{20}
6	$Loop_6$	he_{21}, he_{22}, he_{23}, he_{24}

Tabela 2.6: Semi-Arestas

HE	**Loop**	$v_{inicial}$	**HE**	**Loop**	$v_{inicial}$	**HE**	**Loop**	$v_{inicial}$
1	1	v_1	9	3	v_7	17	5	v_5
2	1	v_2	10	3	v_8	18	5	v_1
3	1	v_3	11	3	v_4	19	5	v_2
4	1	v_4	12	3	v_3	20	5	v_6
5	2	v_6	13	4	v_5	21	6	v_5
6	2	v_7	14	4	v_8	22	6	v_6
7	2	v_3	15	4	v_4	23	6	v_7
8	2	v_2	16	4	v_1	24	6	v_8

Tabela 2.7: Arestas

Aresta	v_i	v_j	HE_s	**Aresta**	v_i	v_j	HE_s
a_1	v_1	v_2	he_1, he_{18}	a_7	v_3	v_7	he_6, he_{12}
a_2	v_2	v_3	he_2, he_7	a_8	v_4	v_8	he_{10}, he_{14}
a_3	v_3	v_4	he_3, he_{11}	a_9	v_5	v_6	he_{20}, he_{21}
a_4	v_4	v_1	he_4, he_{15}	a_{10}	v_6	v_7	he_5, he_{22}
a_5	v_1	v_5	he_{16}, he_{17}	a_{11}	v_7	v_8	he_9, he_{23}
a_6	v_2	v_6	he_8, he_9	a_{12}	v_8	v_5	he_{13}, he_{24}

2.4 Estrutura de Dados Adotada no Pós-Processador

Na implementação do modelo do pós-processador foi adotada a estrutura de dados de ***Semi-arestas***. A estrutura adotada facilita a pesquisa de entidades e, de forma simples, soluciona questões fundamentais para a manipulação de dados geométricos como, por exemplo, dado um do vértices quais faces são comuns a ele, ou dada uma face quais são seus vértices.

A organização do código foi adaptada da proposta sugerida por Mäntylä (1987), cujos detalhes da implementação são apresentados no Capítulo 9.

Capítulo 3

GEOMETRIA COMPUTACIONAL E TRIANGULAÇÕES

3.1 Introdução

No desenvolvimento de aplicativos gráficos é imprescindível a manipulação de modelos geométricos. No caso do pós-processamento a interpretação de malhas, a subdivisão de domínios e a criação de desenhos e gráficos são recursos característicos e fundamentais para a análise de resultados.

A geometria computacional se dispõe a estudar algoritmos capazes de tratar problemas geométricos através do computador. As operações fundamentais mais conhecidas, como por exemplo a distância entre dois pontos, são usadas em conjunto a fim de realizar operações mais elaboradas e complexas.

Um outro tema, também tratado pela geometria computacional, muito importante para o desenvolvimento de aplicativos gráficos é o estudo de estruturas de dados, ou de adjacências, eficientes e capazes de promover a manipulação de informações geométricas de forma clara e metódica. No capítulo 2 já foram mostradas as estruturas de dados que serviram de base para o desenvolvimento dos problemas a seguir apresentados.

3.2 Problemas de Geometria Computacional e Primitivas Geométricas

Inúmeros problemas podem ser solucionados usando a geometria computacional, destacando-se: obtenção do fecho convexo de um conjunto de pontos, triangulação de domínios, intersecção de segmentos, localização de pontos dentre outros.

Operações básicas, conhecidas como primitivas geométricas e problemas clássicos são apresentados a seguir, sem muitos detalhes de implementação, apenas com o intuito de abordar o tema. Tal tema é exaustivamente discutido, por exemplo, em (Figueiredo e Carvalho, 1991). Os detalhes serão mostrados no capítulo 9 juntamente com a organização da implementação e a inclusão do tema no contexto da aplicação.

3.2.1 Primitivas Geométricas

Figuras geométricas, no computador, são representadas através de pontos, para compor retas, curvas e planos. Desta forma, operações básicas da geometria analítica são usadas para descrever algoritmos geométricos.

3.2.1.1 Operações com vetores

As operações com vetores são dadas pelas primitivas a seguir:

$$somavetorial(\mathbf{x},\mathbf{y}) = \mathbf{x} + \mathbf{y} \tag{3.1}$$

$$multescalar(\lambda,\mathbf{y}) = \lambda\mathbf{y} \tag{3.2}$$

$$prodescalar(\mathbf{x},\mathbf{y}) = \mathbf{x}.\mathbf{y} = x_1y_1 + x_2y_2 + ... + x_ny_n \tag{3.3}$$

$$norma(\mathbf{x}) = \|\mathbf{x}\| = \sqrt{x_1^2 + x_2^2 + ... + x_n^2} \tag{3.4}$$

Onde: $\mathbf{x} = (x_1,x_2,...,x_n)$ e $\mathbf{y} = (y_1,y_2,...,y_n)$ são vetores do $\mathbb{R}^n$ e λ é um número real.

3.2.1.2 Distâncias e ângulos

Na geometria analítica muitas vezes deseja-se obter a distância entre dois pontos, ou o ângulo entre vetores, assim sendo, usando as primitivas já definidas anteriormente tem-se:

$$distancia(\mathbf{x,y}) = norma(\mathbf{x - y}) \ , \tag{3.5}$$

a distância entre $\mathbf{x}$ e $\mathbf{y}$, que são pontos do $\mathbf{R}^n$ e

$$angulo(\mathbf{x,y}) = \arccos(\frac{\mathbf{x.y}}{\|\mathbf{x}\|\|\mathbf{y}\|}) \ . \tag{3.6}$$

o ângulo entre os vetores $\mathbf{x}$ e $\mathbf{y}$ pertencentes a $\mathbb{R}^n$.

3.2.1.3 Ângulos orientados no plano

Na obtenção de fechos convexos é necessário ordenar polarmente os vetores do problema sendo essa ordenação feita através da medida dos ângulos entre os vetores bem como a orientação destes mesmos vetores.

A primitiva, já definida, para a obtenção do ângulo entre vetores não gera a orientação relativa de $\mathbf{x}$ e $\mathbf{y}$, não sendo capaz portanto de solucionar o problema de ordenação polar descrito por: dados vetores $\mathbf{x}_1$,$\mathbf{x}_2$,...,$\mathbf{x}_n$ do $\mathbb{R}^2$, ordená-los angularmente no sentido horário (Figueiredo e Carvalho, 1991).

Para a solução do problema seja o vetor $\mathbf{x} = (x_1,x_2) \neq \mathbf{0}$ de $\mathbb{R}^2$, e seja o **ângulo orientado**, definido por $\mathbf{x}$, como sendo igual ao comprimento do arco correspondente no círculo unitário, orientado no sentido anti-horário. Tomando a partir do eixo horizontal, o vetor unitário na direção do semi-eixo horizontal positivo por $\mathbf{u} = (1,0)$, tem-se:

ângulo($\mathbf{x}$) = ângulo($\mathbf{u,x}$) se $x_2 \geq 0$

ângulo($\mathbf{x}$) = -ângulo($\mathbf{u,x}$) se $x_2 < 0$

sendo: $\text{ângulo}(\mathbf{u},\mathbf{x}) = \arccos(\frac{\mathbf{u.x}}{\|\mathbf{u}\|\|\mathbf{x}\|})$,

o que define a orientação do vetor em relação ao semi-eixo.

3.2.1.4 Pseudo-ângulos

Nas primitivas que envolvem ângulos verifica-se o uso de funções *arccos* que apesar de ser uma função conhecida, apresentam duas desvantagens em seu uso. A primeira é que esta função não constitui o elenco das operações elementares, pois não é uma função algébrica. A segunda desvantagem é que, se há a necessidade de comparar ângulos, o conceito de ângulo entre vetores deve ser substituído por outra medida que seja função monótona do ângulo entre eles. Portanto usa-se a função,

$$f(\theta) = 1 - \cos(\theta), \qquad (0 \leq \ \theta \leq \pi) \tag{3.7}$$

sendo esta primitiva o **pseudo-ângulo**, dada por:

$$pseudoAngulo(\mathbf{x},\mathbf{y}) = 1 - \frac{\mathbf{x.y}}{\|\mathbf{x}\|\|\mathbf{y}\|}, \tag{3.8}$$

cujo cálculo envolve somente operações aritméticas.

De forma análoga pode-se definir o pseudo-ângulo em sua forma orientada. O ângulo orientado correspondente a $\mathbf{x}$ é igual ao comprimento do arco orientado correspondente, tomado sobre o círculo unitário centrado na origem. Substituindo o círculo por qualquer outra curva contínua que satisfaça a propriedade de que cada semi-reta partindo da origem a corta em um único ponto (isto é, por uma curva que seja o gráfico de uma função em coordenadas polares), a medida do arco tomado sobre esta curva será uma função monótona do arco tomado sobre o círculo unitário, como visto na figura 3.1. Desta forma o pseudo-ângulo poderá ser utilizado para comparar ângulos orientados (Figueiredo e Carvalho, 1991).

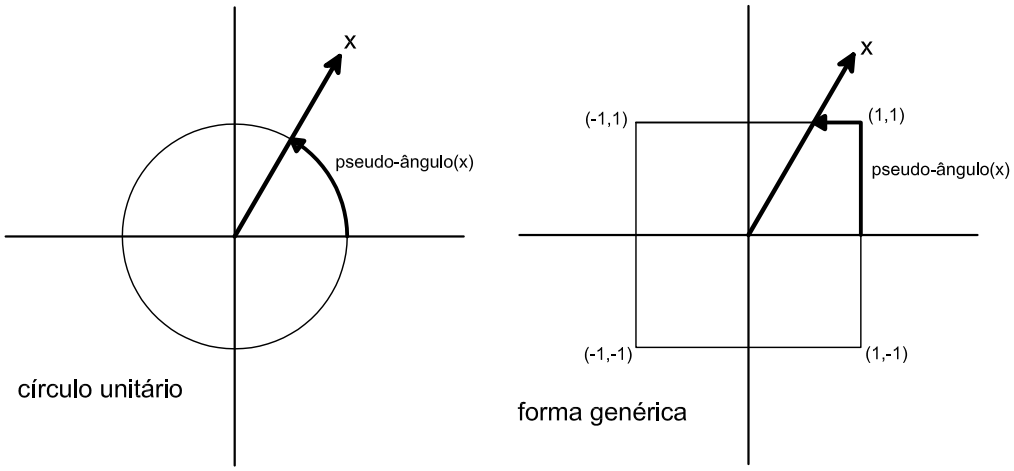


Figura 3.1: Pseudo-ângulos.

3.2.1.5 Produto vetorial

A operação do produto vetorial, como definida pela equação 3.9 é também pode ser usada para estudar a orientação de dois vetores.

$$(x_1, x_2, x_3) \times (y_1, y_2, y_3) = (x_2y_3 - x_3y_2, x_3y_1 - x_1y_3, x_1y_2 - x_2y_1) \tag{3.9}$$

Sejam $\mathbf{x}$ e $\mathbf{y}$ vetores não colineares do $\mathbb{R}^3$, o produto vetorial $\mathbf{x}\times\mathbf{y}$ é um vetor ortogonal a $\mathbf{x}$ e $\mathbf{y}$ e orientado igualmente ao triedro formado pelos eixos coordenados $\boldsymbol{x}$, $\boldsymbol{y}$ e $\boldsymbol{z}$, como mostra a figura 3.2.

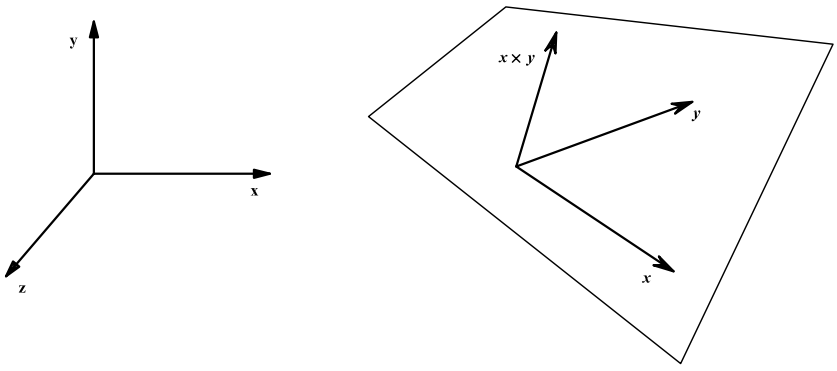


Figura 3.2: Orientação de $\mathbf{x}\times\mathbf{y}$.

O sinal deste produto vetorial indica se o ângulo é positivo ou negativo, sendo que $\mathbf{x}\times\mathbf{y}$ é positivo se $\mathbf{y}$ estiver à esquerda de $\mathbf{x}$, e negativo se $\mathbf{y}$ estiver à direita de

x.

3.2.1.6 Áreas orientadas de polígonos planos

Como visto anteriormente, o sinal do produto vetorial dá a orientação de dois vetores. Considerando o valor absoluto do vetor $\mathbf{x}\times\mathbf{y}$, tem-se a área relativa do paralelogramo definido pelos vetores em questão.

Sejam $\mathbf{x}$ e $\mathbf{y}$ vetores não nulos no espaço. Então,

$$\|\mathbf{x}\times\mathbf{y}\| = \|\mathbf{x}\|\|\mathbf{y}\|\sin\theta$$

é igual a área do paralelogramo determinado por $\mathbf{x}$ e $\mathbf{y}$.

Verifica-se que $\|\mathbf{x}\|\|\mathbf{y}\|\sin\theta$, é a base do paralelogramo vezes a altura (figura 3.3).

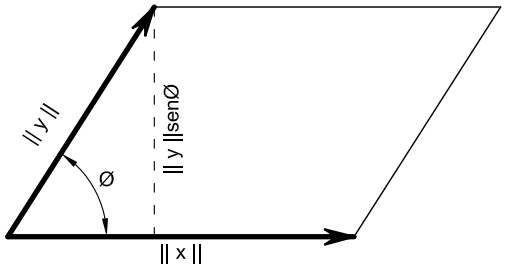


Figura 3.3: Área do paralelogramo.

Da mesma forma, a orientação de uma primitiva é dada pelo mesmo produto vetorial, sendo o sentido anti-horário positivo.

O cálculo da área de um polígono qualquer pode ser generalizado a partir da forma proposta acima.

Seja $p_1, p_2, ..., p_n$ $(n \geq 3)$ um polígono plano simples do $\mathbb{R}^3$ ou $\mathbb{R}^3$. Considere a expressão

$$\mathbf{S} = \frac{1}{2}(op_1 \times op_2 + op_2 \times op_3 + ... + op_n \times op_1) \ , \tag{3.10}$$

onde $\boldsymbol{o}$ é um ponto arbitrário. Então (Figueiredo e Carvalho, 1991):

1. No caso do $\mathbb{R}^3$, $\mathbf{S}$ é um vetor normal ao plano de $p_1, p_2, ..., p_n$, orientado positivamente em relação ao polígono (isto é, de acordo com a regra da mão direita

aplicada ao seu sentido de rotação) e de norma igual à sua area.

2. No caso do $\mathbb{R}^2$, **S** é um escalar igual a área orientada de $p_1, p_2, ..., p_n$. Isto é, $|\mathbf{S}|$ é igual à area do polígono e **S** é positivo se e somente se $p_1, p_2, ..., p_n$, nesta ordem, estão no sentido anti-horário.

Dado um polígono qualquer, como mostrado na figura 3.4, e p_1p_i uma diagonal qualquer, tem-se dois polígonos. Sendo ***o*** um ponto arbitrário,

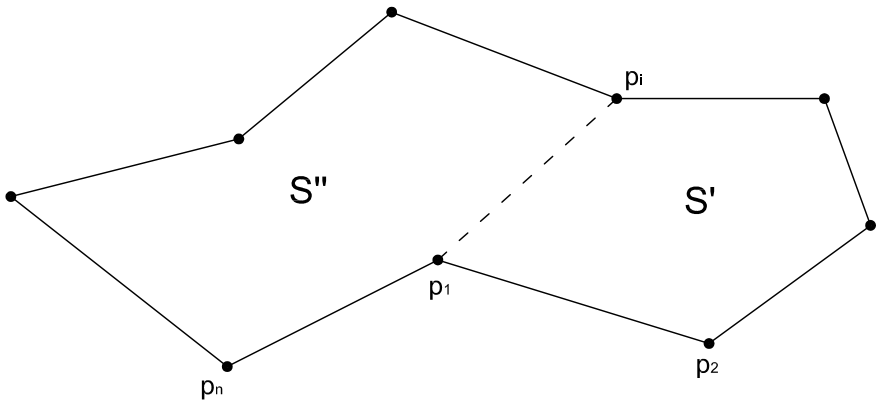


Figura 3.4: Área de um polígono.

$$\mathbf{S'} = \frac{1}{2}(op_1 \times op_2 + op_2 \times op_3 + ... + op_i \times op_1) \quad e \tag{3.11}$$

$$\mathbf{S''} = \frac{1}{2}(op_1 \times op_i + op_i \times op_{i+1} + ... + op_n \times op_1), \tag{3.12}$$

S' + **S"** é um vetor com mesma direção e sentido, e define a área do dado polígono.

Por exemplo, seja $p_1, p_2, ..., p_n$, com p_i=(x_i,y_i) e o= (0,0), a área resultante é

$$\begin{aligned} \mathbf{S} &= (p_1 \times p_2) + (p_2 \times p_3) + ... + (p_{n-1} \times p_n) + (p_n \times p_1) \\ &= x_1y_2 - x_2y_3 + x_2y_3 - x_3y_2 + ... + x_{n-1}y_n - x_ny_n + x_ny_1 + x_1y_n \end{aligned} \tag{3.13}$$

ou matricialmente:

$$\mathbf{S} = \frac{1}{2}\begin{vmatrix} x_1 & x_2 & ... & x_n & x_1 \\ y_1 & y_2 & ... & y_n & y_1 \end{vmatrix} \tag{3.14}$$

3.2.1.7 Localização de um ponto em relação a uma reta

Esta primitiva permite obter a posição de um ponto em relação a uma reta. Dados três pontos p_1, p_2 e p_3, tal que p_1p_2 é um segmento de reta (figura 3.5), deve-se calcular a distância de p_3 em relação a p_1p_2 e, se o valor for negativo, o ponto está a esquerda da reta, se for positivo, o ponto está a direita. Se o ponto é incidente na reta o resultado é zero.

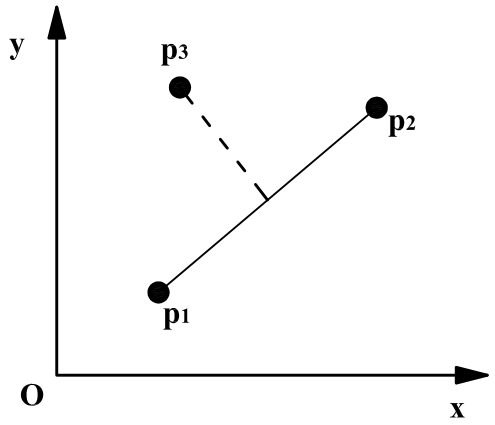


Figura 3.5: Localização de ponto em relação a uma reta.

Seja a equação de uma reta r

$$by = ax + c \tag{3.15}$$

Dado um ponto p de coordenadas x_0 e y_0, pode-se calcular a distância ($d_{p,r}$), de p em relação a r, por

$$d_{p,r} = \frac{ax_0 + by_0 + c}{\sqrt{a^2 + b^2}} \tag{3.16}$$

Logo, se

$$d_{p,r} = 0 \quad , \tag{3.17}$$

o ponto é incidente à reta, se

$$d_{p,r} \geq 0 \quad , \tag{3.18}$$

o ponto está à direita da reta, se

$$d_{p,r} \leq 0 \quad , \tag{3.19}$$

o ponto está à esquerda da reta.

3.2.1.8 Localização de um ponto em relação a uma circunferência

Esta primitiva permite obter a posição de um ponto em relação a uma circunferência no plano. Sabe-se que, a partir de três pontos de uma circunferência no plano, pode-se representar tal circunferência. Desta forma, dados quatro pontos p_1, p_2, p_3 e p_4 em que p_1, p_2 e p_3 representam uma única circunferência de centro O e raio R. Um ponto p_4 é externo à esta circunferência se a distância entre O e p_4 for maior que R. E p_4 é interno se tal distância for menor.

O cálculo desta primitiva pode ser feito a partir dos pontos $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$ e $p_3 = (x_3, y_3)$, definindo uma circunferência e $p_4 = (x_4, y_4)$, o ponto que se deseja verificar. Neste caso se o determinante

$$\begin{vmatrix} x_1 & y_1 & x_1^2 + y_1^2 & 1 \\ x_2 & y_2 & x_2^2 + y_2^2 & 1 \\ x_3 & y_3 & x_3^2 + y_3^2 & 1 \\ x_4 & y_4 & x_4^2 + y_4^2 & 1 \end{vmatrix} = 0 \quad , \tag{3.20}$$

o ponto é incidente à circunferência, se

$$\begin{vmatrix} x_1 & y_1 & x_1^2 + y_1^2 & 1 \\ x_2 & y_2 & x_2^2 + y_2^2 & 1 \\ x_3 & y_3 & x_3^2 + y_3^2 & 1 \\ x_4 & y_4 & x_4^2 + y_4^2 & 1 \end{vmatrix} \geq 0 \quad , \tag{3.21}$$

o ponto é interno à circunferência e se

$$\begin{vmatrix} x_1 & y_1 & x_1^2 + y_1^2 & 1 \\ x_2 & y_2 & x_2^2 + y_2^2 & 1 \\ x_3 & y_3 & x_3^2 + y_3^2 & 1 \\ x_4 & y_4 & x_4^2 + y_4^2 & 1 \end{vmatrix} \leq 0 \quad , \tag{3.22}$$

o ponto é externo à circunferência.

3.2.2 Problemas de Geometria Computacional

3.2.2.1 Ponto em Polígono

Um problema bastante conhecido é o ponto em polígono. Neste problema, dado um ponto, deseja-se saber se ele é interno ou externo a um determinado polígono, como visto na figura 3.6

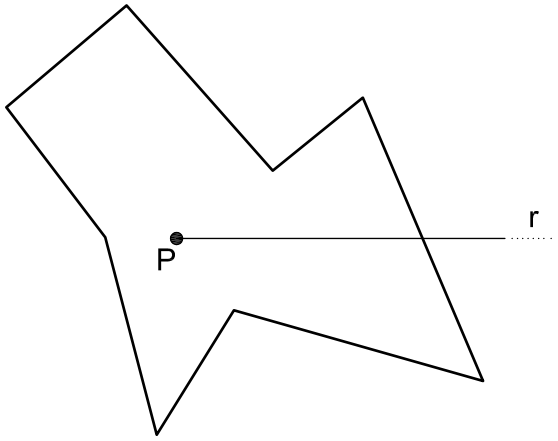


Figura 3.6: Ponto em polígono.

O problema pode ser resolvido pela contagem do número de vezes que a semi-reta $\boldsymbol{r}$, saindo de $\boldsymbol{P}$, intercepta uma aresta do polígono. Desta forma, caso o número de interseções seja par o ponto é exterior ao polígono e, se o número de interseções for ímpar, o ponto é interior, uma vez que a semi-reta no infinito encontra-se exterior ao polígono.

O problema aparentemente simples pode apresentar complicadores que devem ser contornados. Há situações em que a semi-reta intercepta o polígono em um de seus vértices, como mostrado nas figuras 3.7-a e 3.7-b.

A forma mais simples, porém menos eficiente, de resolver o problema consiste em redefinir a semi-reta de forma que a mesma não cruze o polígono em nenhum de seus vértices. Esta solução perde eficiência para geometrias complexas pois as arestas do polígono bem como seus vertices devem ser testados até que uma semi-reta possa ser definida.

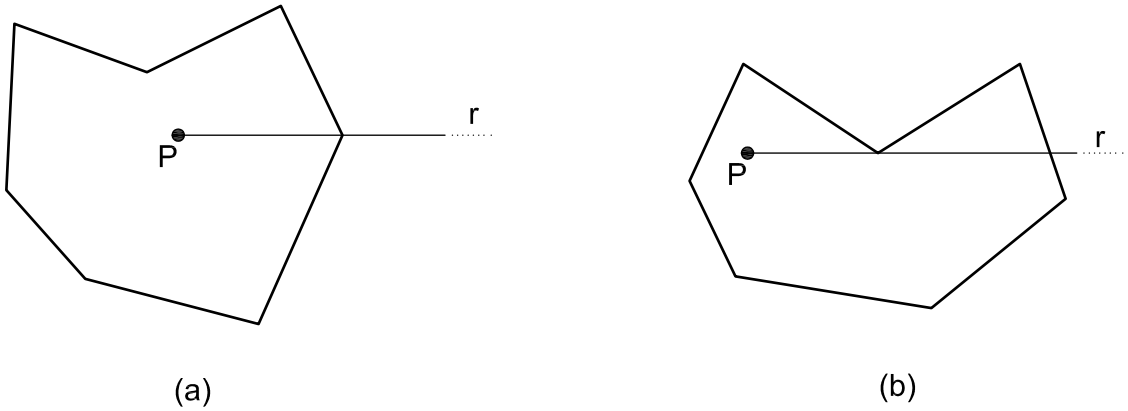


Figura 3.7: Intercessões em vértices de um polígono.

Uma solução mais eficiente, consiste em estabelecer regras para a contagem do número de vezes que a semi-reta toca o polígono. A contagem é feita se o ponto de intersecção da semi-reta $\boldsymbol{r}$ com um lado do polígono não for de ordenada mínima do respectivo lado. Na figura 3.8 é ilustrado este critério.

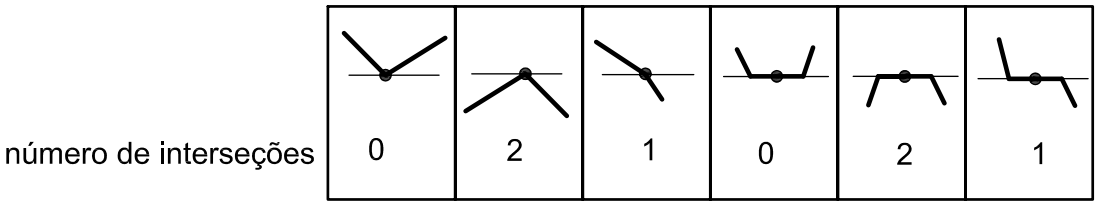


Figura 3.8: Casos de interseção.

3.2.2.2 Fecho Convexo

O fecho convexo ou conv($\mathbf{C}$) de um conjunto $\mathbf{C}$ de pontos é a menor região convexa de $\mathbb{R}^n$ que contém o conjunto $\mathbf{C}$ (figura 3.9). Obter o fecho convexo de um conjunto de pontos é um problema que aparece ao organizar o respectivo conjunto, agrupando os pontos em uma região simples. No caso o conjunto $\mathbf{C}$, do qual se quer obter o fecho convexo conv($\mathbf{C}$), é um conjunto finito de pontos, e o fecho convexo é chamado de politopo.

Existem várias formas de se obter o fecho convexo de um conjunto de pontos, seja para o caso bidimensional ou tridimensional. Os principais algoritmos usados para

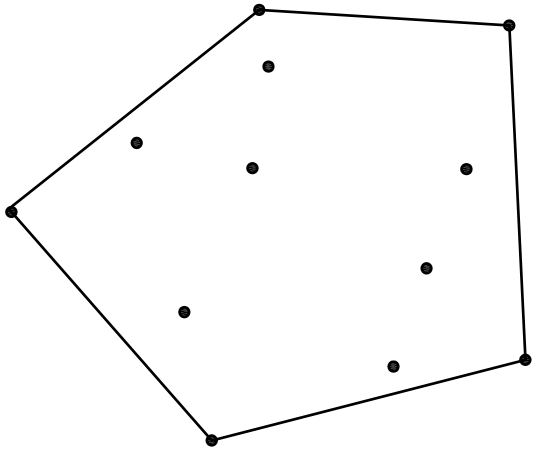


Figura 3.9: Fecho Convexo.

solucionar este problema são: o algoritmo de Chand e Kapur, o algoritmo de Jarvis, o algoritmo de Graham, o algoritmo Quickhull, o algoritmo Mergehull e um algoritmo usando a união de fechos convexos (maiores detalhes destes algoritmos podem ser encontrados em Sedgewick (1988) e Figueiredo e Carvalho (1991)). A seguir serão apresentados o caso bidimensional do algoritmo de Jarvis e do algoritmo de Graham.

O algoritmo de Jarvis, também conhecido como "embrulho-pra-presente", é o caso bidimensional do algoritmo de Chand e Kapur. O algoritmo parte de um ponto (de preferência, o de maior abcissa e menor ordenada), que já seja um ponto do fecho convexo, e uma semi-reta de origem neste ponto é posta a girar no sentido anti-horário até encontrar um outro ponto do conjunto (o ponto de menor ângulo com a horizontal). A semi-reta passa para o ponto encontrado que é um vértice do fecho. O processo termina no momento em que todo o contorno foi determinado. Na figura 3.10 pode-se visualizar o algoritmo.

O algoritmo de Graham parte do princípio de que, estando os pontos ordenados polarmente em torno de um ponto interior qualquer, por exemplo o baricentro, pode-se obter um polígono estrelado (figura 3.11) e, a partir deste polígono, determinar o fecho convexo.

O problema pode ser solucionado analisando cada vértice do polígono estrelado,

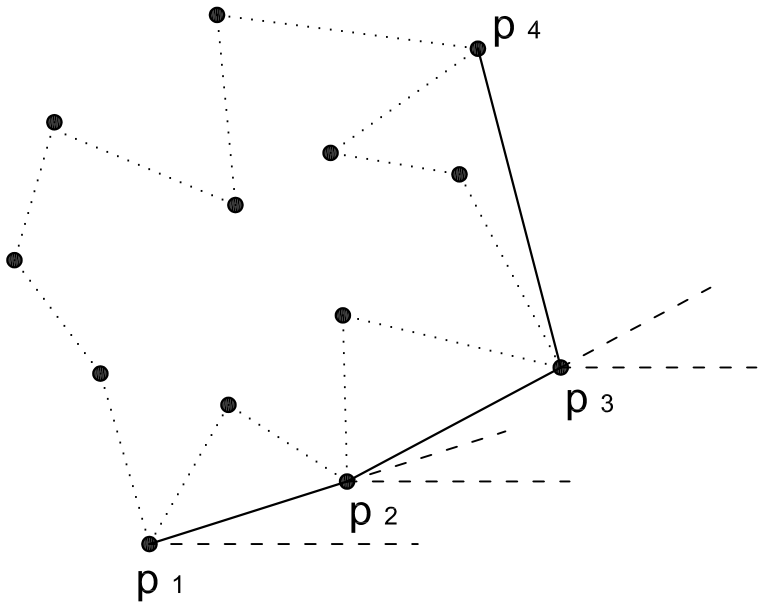


Figura 3.10: Algoritmo de Jarvis.

Se o ângulo interno for convexo, este ponto não é vértice do fecho convexo. Após percorrer toda a lista dos vértices do polígono, ao eliminar-se um dos pontos, o ponto anterior tem seu ângulo mudado, devendo ser avaliado novamente. O procedimento é realizado até que não exista mais vértices com ângulos convexos. Desta forma, os vértices restantes constituem, portanto, um fecho convexo. O algoritmo é melhor entendido na figura 3.12.

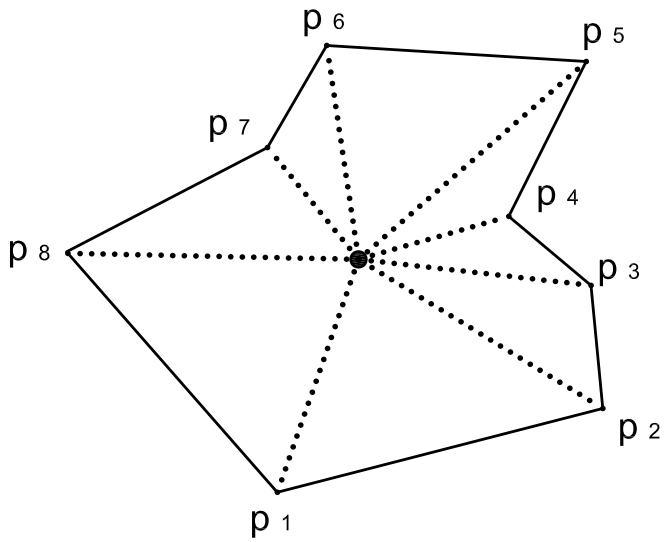


Figura 3.11: Polígono Estrelado.

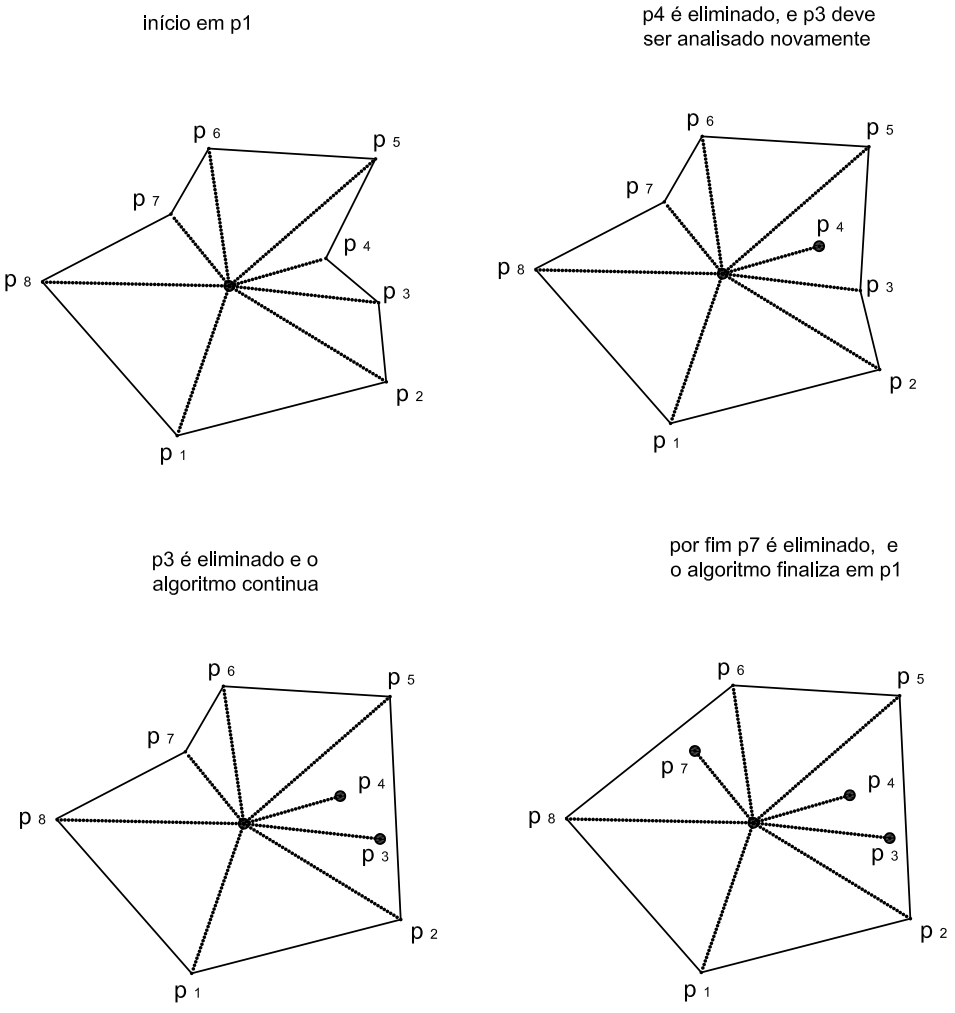


Figura 3.12: Algoritmo de Graham.

3.3 Triangulações

Seja uma função f com imagens conhecidas para o domínio de pontos $x_1, x_2, ..., x_n \in \mathbb{R}$. Pretende-se interpolar a função $f(x_i)$, por uma função F(x), a fim de estender f a um intervalo de pontos.

A forma mais simples de interpolação é a linear por partes, sendo os valores de F em cada um dos pontos obtidos a partir de $f(x_i)$. Portanto basta unir linearmente os intervalos $[x_i, x_{i+1}]$, calculando $F(x) = \lambda f(x_i) + (1 - \lambda) f(x_{i+1})$, como mostra a figura 3.13.

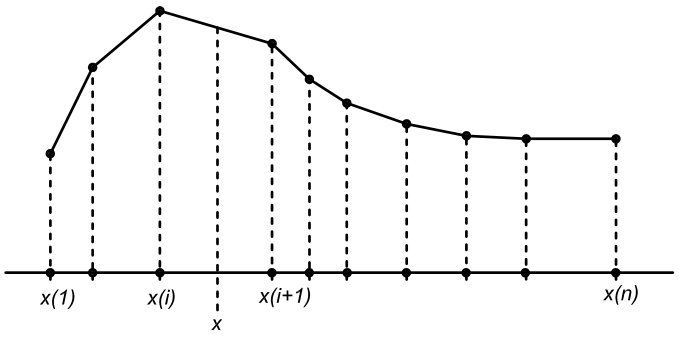


Figura 3.13: Interpolação linear .

A idéia discutida anteriormente para uma dimensão pode ser expandida para duas dimensões. Para um domínio bidimensional D, a base para o processo de interpolação é a representação do mesmo como uma união de triângulos, compondo um **conjunto simplicial** de duas dimensões, isto é: um conjunto cuja interseção de dois triângulos seja um lado ou um vértice comum.

Após a definição dos triângulos, tem-se os valores de x expressos em coordenadas baricêntricas $(\lambda_1, \lambda_2, \lambda_3)$,

$$x = \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3, \tag{3.23}$$

e, desta forma, tem-se:

$$F(x) = \lambda_1 f(x_1) + \lambda_2 f(x_2) + \lambda_3 f(x_3). \tag{3.24}$$

podendo-se assim definir o seguinte problema a ser resolvido:

TRIANGULAÇÃO: *Dado um conjunto* ***C*** *de pontos do plano, obter uma triangulação do fecho convexo, ou conv(****C****), (isto é, um conjunto simplicial cuja união seja conv(****C****)), cujo conjunto de vértices seja* ***C*** (Figueiredo e Carvalho, 1991).

3.3.1 Diagrama de Voronoi

A interpolação de uma função é desnecessária quando esta assume valores discretos. Assim, dado um conjunto de pontos $\mathbf{C} = \{x_1, x_2, ..., x_n\}$, para os valores de $\boldsymbol{x}$ que não estão contidos em **C**, assume-se $\mathrm{F}(\boldsymbol{x}) = f(x_i)$, onde x_i é próximo a $\boldsymbol{x}$. A figura 3.14 ilustra o caso unidimensional.

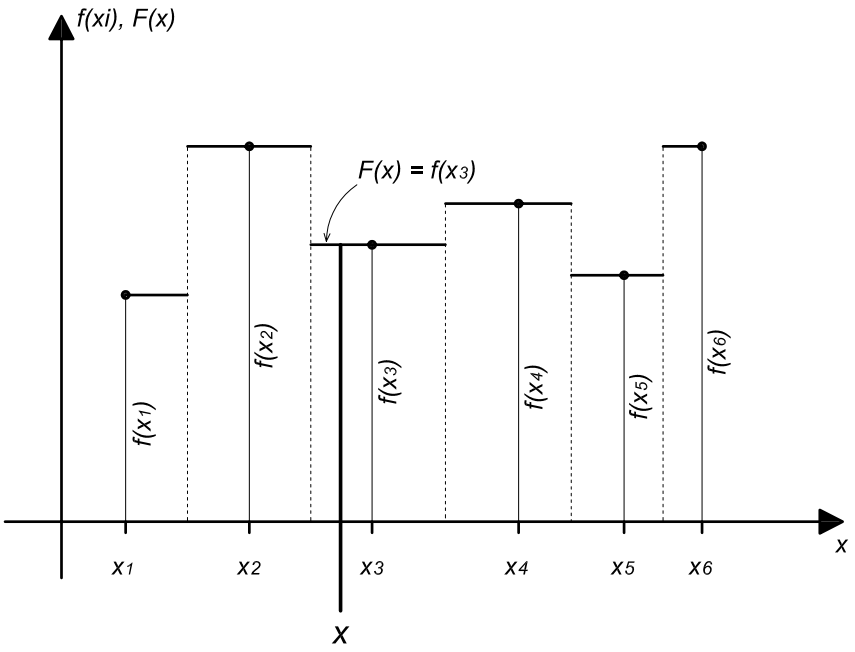


Figura 3.14: Voronoi unidimensional

Para o caso unidimensional é fácil visualizar que F é constante na parte formada pelos pontos médios dos intervalos gerados pela ordenação dos elementos de **C**.

Para o caso bi-dimensional, os pontos $\boldsymbol{x}$ para os quais $F(\boldsymbol{x}) = f(x_i)$ estão contidos no polígono V_i, chamado de **Polígono de Voronoi** relativo a x_i, e a composição destes polígonos forma o **Diagrama de Voronoi**, ou **Vor(C)** (Figura 3.15).

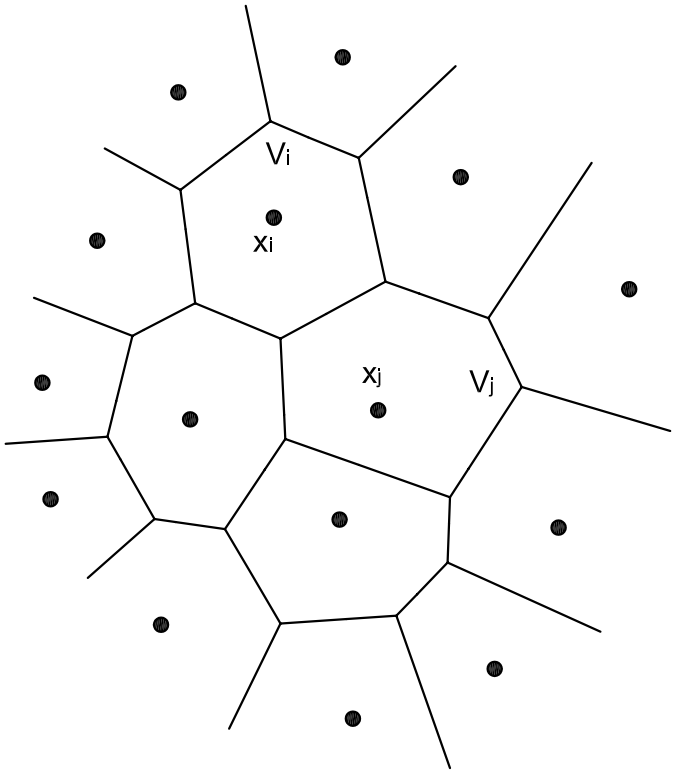


Figura 3.15: Diagrama de Voronoi.

3.3.2 Propriedades do Diagrama de Voronoi

O diagrama de Voronoi possui diversas propriedades, fundamentais para a composição dos polígonos e por conseguinte do próprio diagrama:

(i) O diagrama é formado por polígonos de Voronoi, como já dito anteriormente, e cada aresta é comum a dois polígonos, sendo eqüidistante de dois pontos (x_i e x_j) dos respectivos polígonos (V_i e V_j). Em outras palavras, as arestas são segmentos de mediatrizes definidas por pontos de **C** (figura 3.16(a)).

(ii) Uma aresta só é comum a dois polígonos (V_i e V_j), se existe um círculo que contenha x_i e x_j, e se os demais pontos de **C** forem exteriores ao mesmo (figura 3.16(b)).

(iii) Uma aresta é comum a dois polígonos e um vértice interno é comum a pelo menos três polígonos, sendo este vértice centro de uma circunferência definida

pelos pontos dos polígonos de Voronoi. Desta forma, pode-se compor subconjuntos formados por três pontos cada e cada um destes subconjuntos definem os chamados "círculos vazios" (figura 3.16(c)).

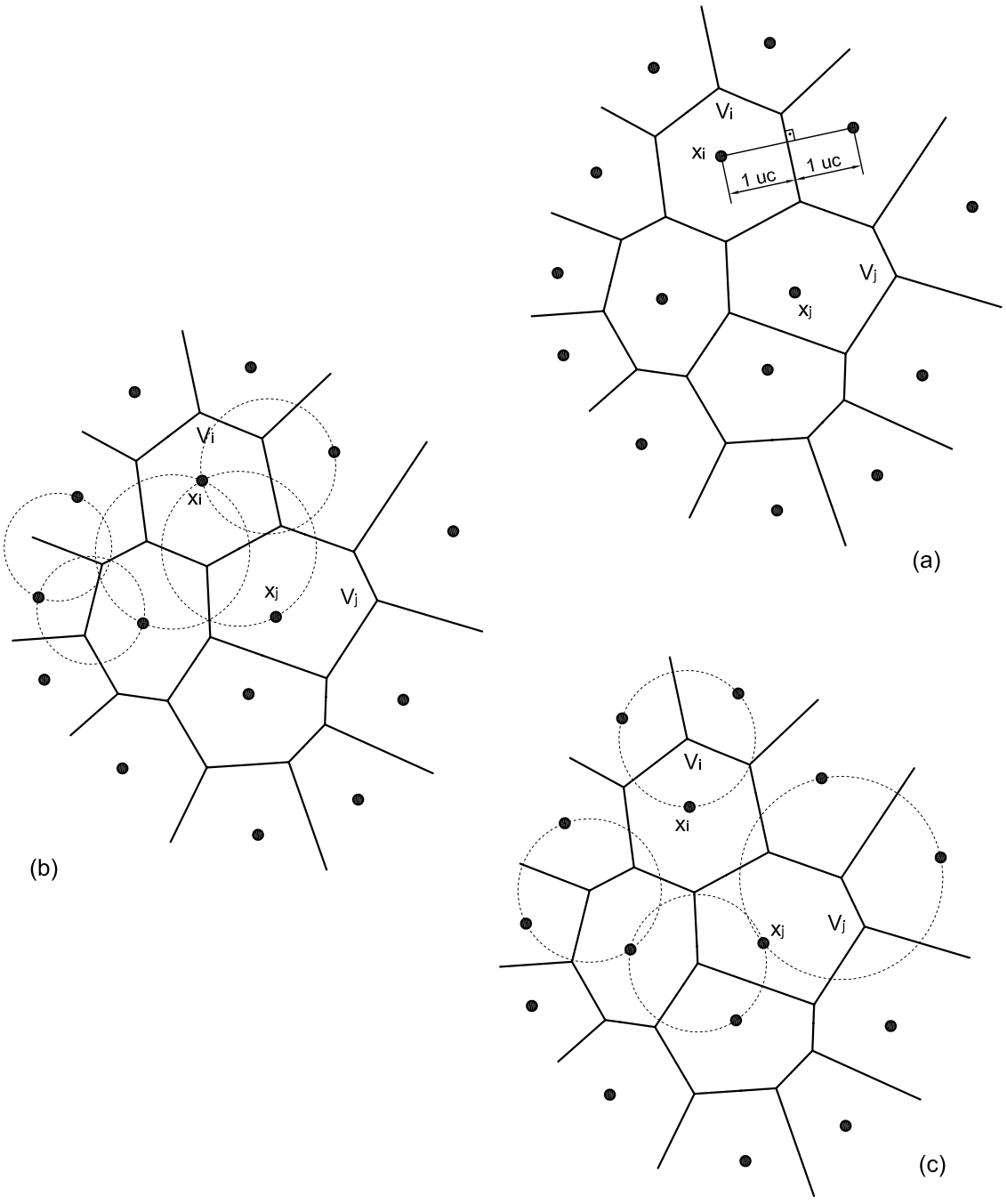


Figura 3.16: Propriedades do Diagrama de Voronoi.

3.3.3 Triangulação de Delaunay

O diagrama de Voronoi e a triangulação de Delaunay estão fortemente ligados, pela construção do grafo planar e pelo dual deste grafo planar (figura 3.17). Isto é, dado um Vor(**C**), dois pontos de polígonos vizinhos formam uma aresta do grafo dual, e esta aresta é parte do diagrama de Delaunay, ou Del(**C**). Assim, dois pontos só representam uma aresta de Del(**C**), se existe uma circunferência contendo somente os dois pontos, como ilustrado na figura 3.17.

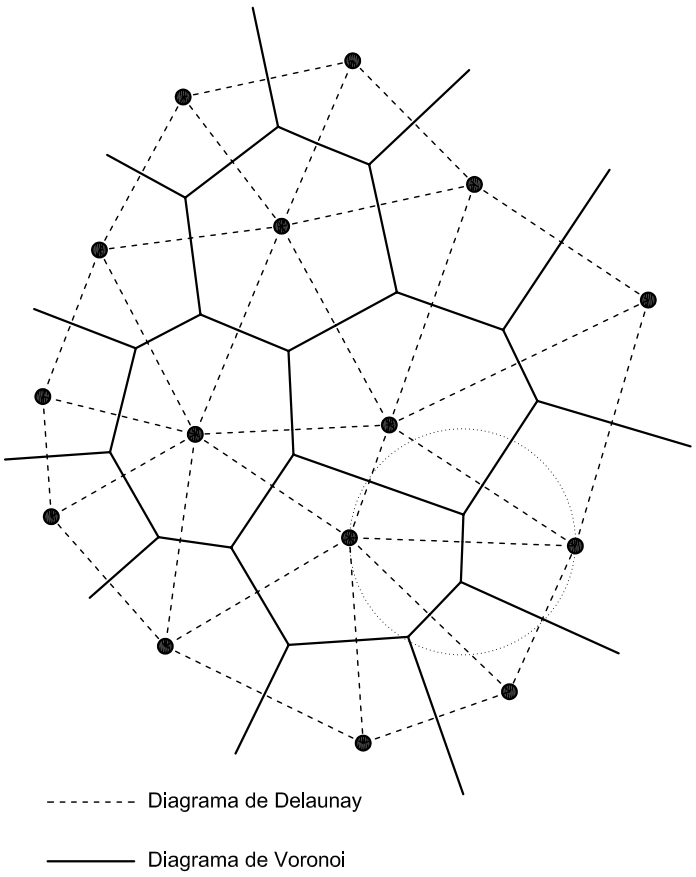


Figura 3.17: Diagramas de Voronoi e Delaunay

O diagrama de Delaunay, obtido a partir do grafo dual de um diagrama de Voronoi, apresenta diversas características que podem ser usadas para se obter a triangulação:

(i) Em um triângulo interno a uma triangulação de Delaunay, com os vértices *ijk*,

existe um círculo que contém os vértices deste triângulo e exclui todos os demais pontos da triangulação (figura 3.18).

(ii) Não há triângulos com ângulos muito pequenos e nem muito grandes, uma vez a maximização do menor ângulo de cada triângulo é garantida, conforme ilustra a figura 3.19.

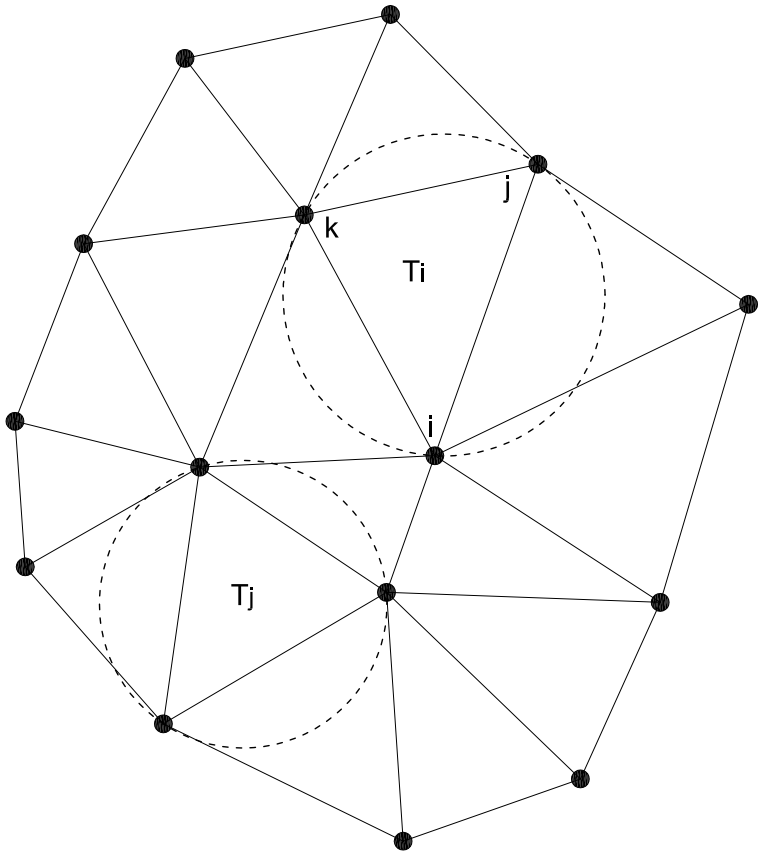


Figura 3.18: Propriedades do diagrama de Delaunay

Uma forma de se obter a triangulação garantindo-se às características acima, para um dado fecho convexo, é, partindo de uma das arestas, obter o próximo ponto do triângulo de Delaunay, com aquele que forma o maior ângulo como a respectiva aresta (figura 3.19).

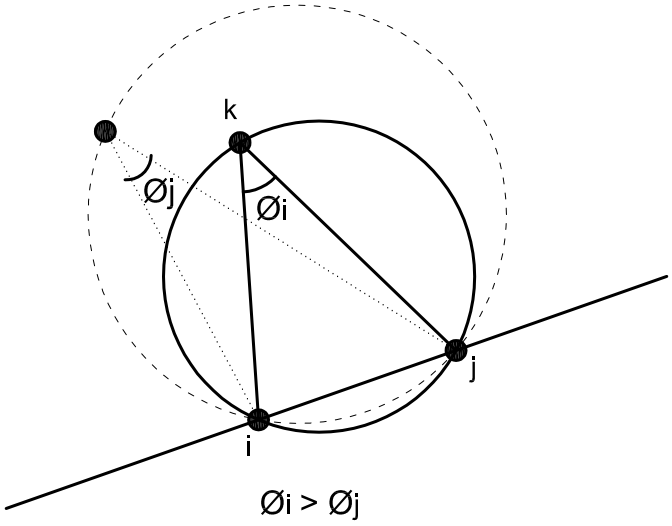


Figura 3.19: Composição da Triangulação de Delaunay

3.4 Triangulação para a Subdivisão de Domínios no Pós-processador

Na representação de grandezas de modelos numéricos muitas vezes é necessário a subdivisão de domínios em polígonos, como será visto no capítulo 6. A triangulação de Delaunay, como visto, somente necessita das informações de vértices para ser gerada, o que permite uma geração de malha triangular bem genérica. No capítulo 2 foi descrita a estrutura de "semi-arestas", na qual o modelo do pós-processador se baseia. Desta forma, as informações dos vértices podem ser obtidas com facilidade, propiciando o uso da triangulação de Delaunay para a subdivisão de domínios.

No pós-processador a triangulação foi usada para a criação de uma malha interior aos elementos finitos visando a representação das grandezas internas de cada

elemento (tensão e deformação por exemplo). O algoritmo implementado percorre elemento por elemento gerando a triangulação em cada um separadamente. Para tanto são usados os pontos de integração do elemento e os nós como vértices da malha de Delaunay.

O algoritmo da triangulação de Delaunay foi adaptado da implementação proposta por Christian Icking (2001). Tal implementação sugere uma triangulação dinâmica de forma que os vértices são introduzidos um por vez e a malha é redefinida a cada adição de um novo vértice. O processo pode ser visto nas figuras 3.20-a e 3.20-b.

Verifica-se que, no caso ilustrado na figura 3.20-a, há uma expansão da malha de triângulos. Isto ocorre quando um vértice é adicionado externo ao fecho convexo do conjunto de vértices dos triângulos existentes. Já no caso ilustrado na figura 3.20-b, o vértice adicionado é interno ao fecho convexo dos vértices que compõe a malha triangular, portanto há uma redefinição da triangulação.

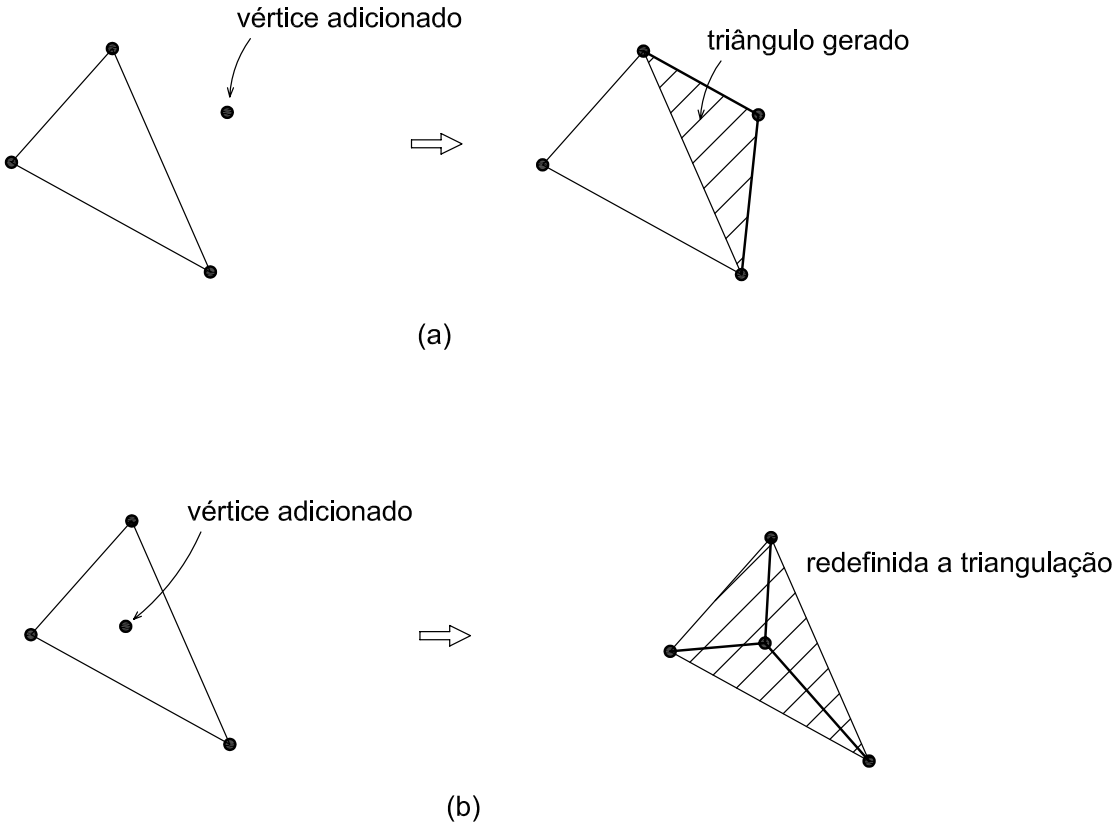


Figura 3.20: Algoritmo de Delaunay: (a) Expansão da malha (b) Redefinição da malha

A geração dinâmica da triangulação de Delaunay pode ser usada em geradores de malhas para o pré-processamento, uma vez que a formação da malha pode ser visualizada em tempo de execução. Entretanto no pós-processamento a redefinição dinâmica perde um pouco o sentido uma vez que a triangulação não precisa ser vista. Partindo desta idéia, o algoritmo foi ampliado permitindo também que a triangulação fosse feita por todos os vértices de uma única vez (e não, elemento por elemento), se adequando mais à subdivisão de domínios proposta pelo pós-processador, como será visto adiante (capítulo 6).

Capítulo 4

Transformações Geométricas

4.1 Introdução

As transformações geométricas estão presentes de diversas formas em sistemas gráficos computacionais, desde a criação de imagens, até a simples visualização.

No pré-processamento de modelos estruturais, a manipulação geométrica é de extrema importância, pois a geração de malhas, conformação da geometria e a visualização do modelo são definidas de forma mais apurada com o auxílio das operações de transformações.

Em um programa de pós-processamento, o uso das transformações geométricas também assume extrema importância. A visualização de resultados, como forma deformada de um modelo, é melhor representada fazendo o uso de tais operações.

Pode-se definir transformações geométricas como sendo funções matemáticas que alteram pontos no espaço, mais especificamente, uma correspondência entre dois conjuntos: domínio e imagem da função. Os pontos tomados no domínio da função são representados graficamente por algum lugar geométrico, que levam a uma imagem com modificações bem definidas.

As principais transformações geométricas são as operações de **translação**, **rotação**, **escala** e as **projeções**.

Ao longo deste trabalho serão apresentados diversos exemplos nos quais o uso

das transformações foram fundamentais para se obter um melhor resultado na visualização e representação de grandezas.

4.2 Operações de Transformação e Projeção

Transformações geométricas são operações que podem ser utilizadas visando a alteração de algumas características como posição, orientação, forma ou tamanho do objeto a ser desenhado (Azevedo e Conci, 1999). Entretanto, estas operações não alteram a topologia de um objeto. As transformações alteram as coordenadas dos pontos que descrevem o objeto, mas não alteram o sistema de coordenadas onde ele está definido.

4.2.1 Translação

A **translação** é a mais simples das transformações podendo ser aplicada a uma imagem (Rowe, 2001). Esta operação consiste em mover um objeto de um lugar para outro sem alterar a sua forma, tamanho ou orientação, como pode ser visto na figura 4.1.

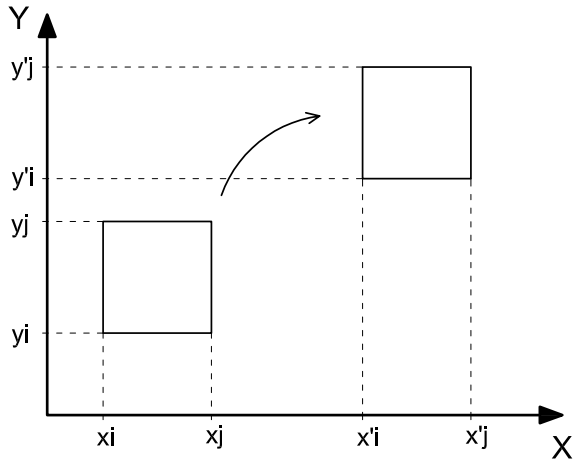


Figura 4.1: Transformação de translação.

As equações de translação, como podem ser vistas a seguir, adicionam fatores de translação às coordenadas dos pontos que formam o objeto a ser transladado. Para um ponto tem-se:

$$x' = x + T_x \tag{4.1}$$

$$y' = y + T_y \tag{4.2}$$

$$z' = z + T_z \tag{4.3}$$

onde:

x', y', z' são as coordenadas finais do ponto após a translação;

x, y, z são as coordenadas originais do ponto e

T_x, T_y, T_z são os fatores de translação.

Representando na forma matricial,

$$\left[\begin{array}{ccc} x' & y' & z' \end{array}\right] = \left[\begin{array}{ccc} x & y & z \end{array}\right] + \left[\begin{array}{ccc} T_x & T_y & T_z \end{array}\right] \tag{4.4}$$

Em termos vetoriais é a soma do vetor de coordenadas original do ponto com o vetor de translação.

4.2.2 Rotação

A rotação de um objeto significa o mesmo que girá-lo de um determinado ângulo como mostrado na figura 4.2. A rotação ocorre em relação a um ponto fixo sem alterações na forma e tamanho desse objeto. A trigonometria é usada para efetuar as operações de rotação. Através das funções trigonométricas seno e cosseno, pode-se rotacionar qualquer figura geométrica. Para isso, é necessário a utilização de equações adequadas.

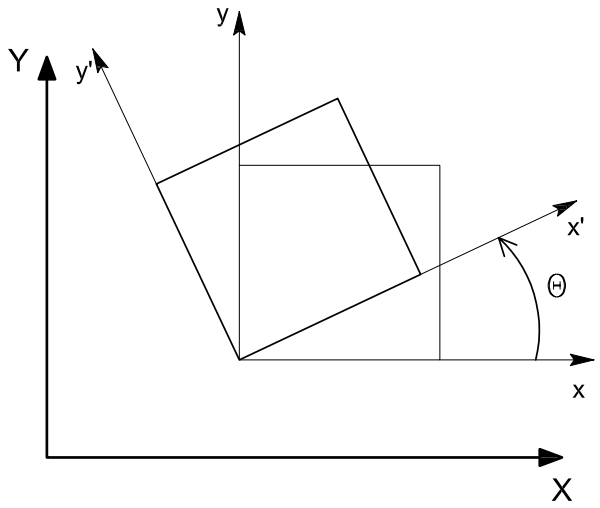


Figura 4.2: Transformação de rotação.

No plano as, equações de rotação, vistas abaixo, podem ser obtidas pela rotação de eixos conforme ilustrado na figura 4.3.

$$x' = x.cos(\theta) - y.sen(\theta) \tag{4.5}$$

$$y' = y.cos(\theta) + x.sen(\theta) \tag{4.6}$$

Organizando as equações na forma matricial, obtém-se a matriz de rotação, como mostra a equação 4.7.

$$\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} cos(\theta) & sen(\theta) \\ -sen(\theta) & cos(\theta) \end{bmatrix}, \tag{4.7}$$

sendo a matriz de rotação R definida por

$$R = \begin{bmatrix} cos(\theta) & sen(\theta) \\ -sen(\theta) & cos(\theta) \end{bmatrix} \tag{4.8}$$

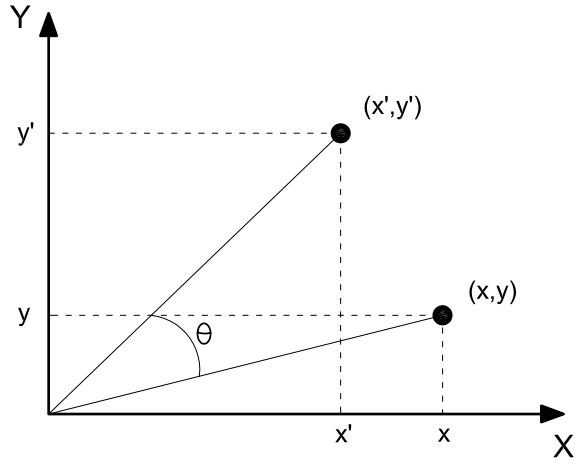


Figura 4.3: Rotação de um ponto em torno da origem.

Desta forma, pode-se definir matrizes de rotação em três planos distintos: XY, XZ, YZ. A rotação no plano XY se dá em torno do eixo Z. A rotação do plano XZ se dá em torno do eixo Y e a rotação do plano YZ se dá em torno do eixo X. As equações de rotação de um ponto para os três eixos são dadas por:

$$\begin{bmatrix} x' & y' & z' \end{bmatrix} = \begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} cos(\alpha) & sen(\alpha) & 0 \\ -sen(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{4.9}$$

para a rotação no plano XY de um ângulo α.

$$\begin{bmatrix} x' & y' & z' \end{bmatrix} = \begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\delta) & sen(\delta) \\ 0 & -sen(\delta) & cos(\delta) \end{bmatrix}, \tag{4.10}$$

para a rotação no plano YZ de um ângulo δ.

$$\begin{bmatrix} x' & y' & z' \end{bmatrix} = \begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} cos(\beta) & 0 & -sen(\beta) \\ 0 & 1 & 0 \\ sen(\beta) & 0 & cos(\beta) \end{bmatrix}, \tag{4.11}$$

para a rotação no plano XZ de um ângulo β.

4.2.3 Escala

Assim como na translação, a operação de escala é muito simples, consistindo em uma aplicação de um fator de escala às coordenadas dos pontos de objeto. Este fator pode ter o efeito de aumentar ou reduzir o tamanho do objeto, como mostrado na figura 4.4.

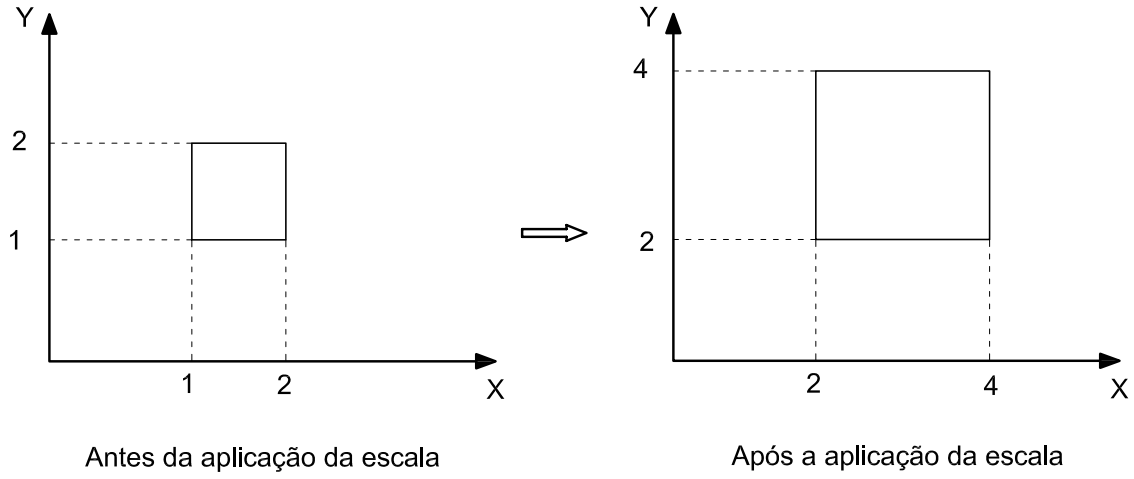


Figura 4.4: Transformação de escala.

A equação 4.12 representa o caso tridimensional para a transformação de escala, podendo-se adotar fatores de escala diferentes para os três eixos.

$$\begin{bmatrix} x' & y' & z' \end{bmatrix} = \begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & S_z \end{bmatrix} = \begin{bmatrix} xS_x & yS_y & zS_z \end{bmatrix} \tag{4.12}$$

4.2.4 Cisalhamento

O cisalhamento é uma transformação que distorce o formato de um objeto. É aplicado um deslocamento aos valores das coordenadas x, y ou z dos pontos do objeto, proporcional ao valor das outras coordenadas de cada ponto transformado (figura 4.5).

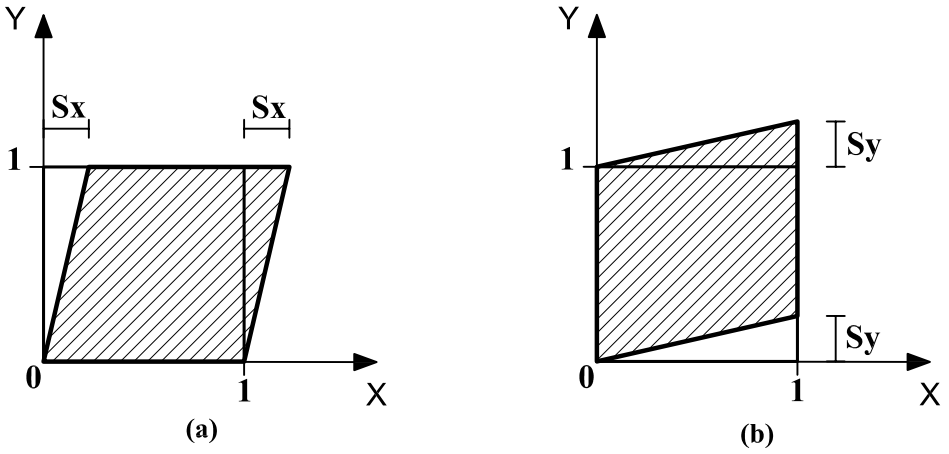


Figura 4.5: Transformação de cisalhamento.

Na figura 4.5(a) a transformação pode ser obtida com a seguinte matriz

$$C = \begin{bmatrix} 1 & 0 & 0 \\ Sx & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.13}$$

Para a figura 4.5(b) a matriz de transformação é

$$C = \begin{bmatrix} 1 & Sy & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.14}$$

sendo

C a matriz de cisalhamento;

Sx o fator de cisalhamento na direção x;

Sy o fator de cisalhamento na direção y.

Qualquer número real pode ser usado sendo possível fazer a distorção em qualquer direção. Por exemplo, a seguinte matriz

$$C = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \tag{4.15}$$

irá distorcer um objeto ao longo da direção y de forma proporcional às coordenadas x e z de cada um de seus pontos.

4.2.5 Projeções Geométricas

As projeções geométricas são operações que visam transformar o espaço 3D para o plano 2D. Em decorrência dessas operações é possível visualizar um objeto tridimensional em uma tela bidimensional.

As projeções podem ser Paralelas ou Perspectivas. As projeções Paralelas são subdivididas em Oblíquas (Cavaleira ou Cabinet) ou Ortográficas (Axonométricas e múltiplas vistas ortográficas).

As projeções definidas por raios de projeções paralelos são tradicionalmente usadas em engenharia e desenhos técnicos. Em alguns casos, elas preservam as verdadeiras dimensões do objeto, mas não produzem uma imagem realista, por não tornarem menores as medidas mais distantes do observador (Azevedo e Conci, 1999).

4.2.5.1 Projeções Paralelas Ortográficas

As projeções ortográficas apresentam o plano de projeção paralelo aos planos cartesinos (XY, XZ, YZ). Em objetos cujas faces são ortogonais entre si, a aplicação de uma projeção ortográfica esconde uma das faces. Abaixo são apresentadas as equações para as projeções de pontos nos planos XY, XZ e YZ.

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & T_z & 1 \end{bmatrix}, \tag{4.16}$$

para projeção no plano XY, onde T_z é a coordenada z do plano de projeção.

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & T_y & 0 & 1 \end{bmatrix}, \tag{4.17}$$

para projeção no plano XZ, onde T_y é a coordenada y do plano de projeção.

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & 0 & 0 & 1 \end{bmatrix}, \tag{4.18}$$

para projeção no plano YZ, onde T_x é a coordenada x do plano de projeção.

4.2.5.2 Projeções Paralelas Axonométricas

As projeções vista lateral, frontal e planta permitem uma observação parcial do objeto projetado. Nestes casos, ao se observar apenas uma delas, não é possível se conceber corretamente a forma do objeto. De forma a solucionar este problema, é usual a adoção das projeções vista lateral, frontal e planta, mais uma projeção axonométrica, a qual permite que se tenha uma visão mais integrada do objeto. As projeções paralelas ortográficas axonométricas têm a direção de projeção e a normal ao plano de projeção não coincidentes com a direção de um dos eixos principais (Battaiola, 1998).

Obter uma projeção axonométrica usando métodos computacionais implica em usar uma série de rotações e projeções concatenadas gerando assim a requerida projeção. Para uma projeção no plano XY, precedida de duas rotações, uma de β em torno do eixo Y e outra rotação δ em torno do eixo X, tem-se a seguinte equação:

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} R_\delta \, R_\beta \, T \, , \tag{4.19}$$

com

$$R_\beta = \begin{bmatrix} cos(\beta) & 0 & -sen(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ sen(\beta) & 0 & cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R_\delta = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\delta) & sen(\delta) & 0 \\ 0 & -sen(\delta) & cos(\delta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ e}$$

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

resultando em

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} cos(\beta) & sen(\beta)sen(\delta) & 0 & 0 \\ 0 & cos(\delta) & 0 & 0 \\ sen(\beta) & -cos(\beta)sen(\delta) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.20)$$

A equação 4.20 qualquer projeção, independente do seu tipo, pode ser obtida através de concatenações de transformações geométricas.

4.3 Transformações Geométricas Utilizadas no Pós-processador

No pós-processador transformações geométricas como rotações, translações e projeções são amplamente usadas. Foram implementadas as comumente usadas: transformações de translação, rotação, escalas, cisalhamento (ou distorção), reflexão e projeções perspectivas, paralelas oblíquas (Cavaleira e Cabinet) e paralelas ortográficas axonométricas.

Os detalhes das implementações, bem como a organização das classes serão vistas no capítulo 9.

Capítulo 5

SUAVIZAÇÃO DE GRANDEZAS DO MEF

5.1 Introdução

A representação dos resultados deve descrever a variação ou distribuição de uma dada grandeza de forma mais aproximada do fenômeno real. O Método dos Elementos Finitos é um método aproximado que fornece bons resultados em apenas alguns poucos pontos, ditos pontos de Gauss. Além deste problema, tem-se a questão da descontinuidade de resultados entre elementos de uma malha, como mostrado na figura 5.1.

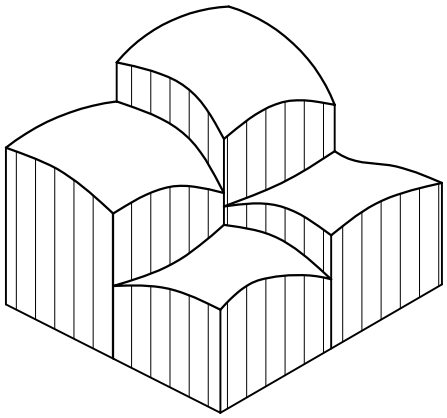


Figura 5.1: Descontinuidade de resultados entre elementos.

O tratamento para estes problemas, portanto, deve considerar que:

1. A descrição de valores deve ser baseada nos valores nodais;

2. Cada elemento produz um valor nodal diferente;

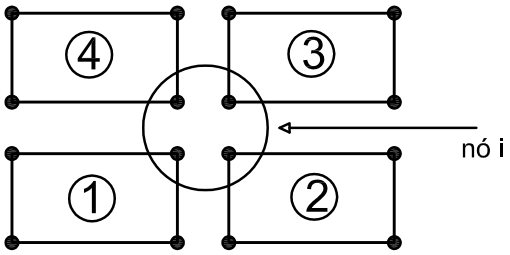
3. Os pontos nodais não são pontos ótimos para o cálculo dos resultados.

A obtenção de um campo contínuo de resultados, pode se basear no cálculo das médias nodais, cálculo dos mínimos quadrados ou no uso de funções de interpolação.

5.2 Média Nodal

A média nodal é uma técnica simples e parte do cálculo da média aritmética dos diversos valores que possivelmente podem ocorrer em dado nó do modelo.

Para o cálculo, basta obter os respectivos resultados provindos dos elementos incidentes no nó. A partir destes valores calcula-se a média, e o resultado é o valor representativo da grandeza naquele nó. O procedimento está ilustrado na figura 5.2 (Salem, 1988).



$$Vi = \frac{Vi(1) + Vi(2) + Vi(3) + Vi(4)}{4}$$

Figura 5.2: Média Nodal.

5.3 Mínimos Quadrados

O objetivo é obter um campo contínuo dos resultados através de valores nodais. A suavização por mínimos quadrados pode ser feita em um elemento (**suavização Local**) ou em toda a malha (**suavização global**).

5.3.1 Suavização Global

Partindo de resultados nos pontos de integração dos elementos de determinada malha, deve-se calcular os valores das tensões suavizadas, σ_s, nos nós de toda malha, como ilustrado na figura 5.3 (Oñate, 1995).

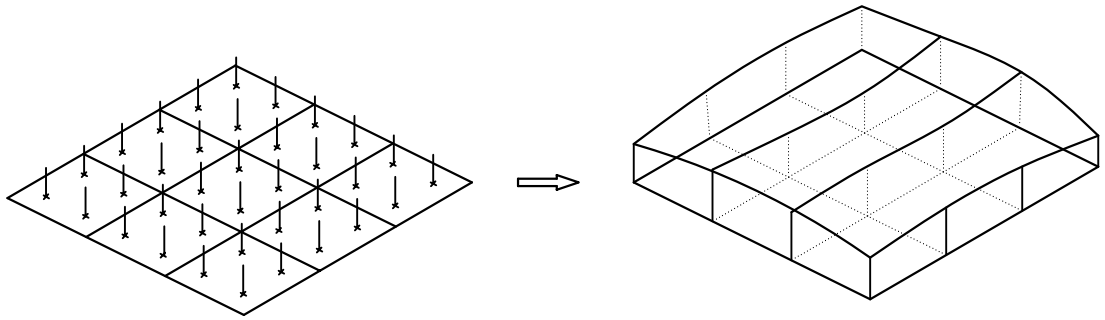


Figura 5.3: Suavização Global.

Para um elemento finito, pode-se escrever

$$\sigma_s \;=\; \sum_{i=1}^{n} N_i \sigma_i^{(e)} \;=\; \mathbf{N}\boldsymbol{\sigma}^{(\boldsymbol{e})} \tag{5.1}$$

sendo:

n, o numero de nós do elemento;

$\sigma_i^{(e)}$, o valor da tensão em um ponto nodal do elemento;

N_i, o valor da função de forma no nó do elemento.

De posse dos valores das tensões suavizadas e, a partir da minimização do erro, tem-se um problema de mínimos quadrados, cujo funcional a ser minimizado é:

$$F = \int_{V_e} e^2 dV \tag{5.2}$$

$$\text{sendo} \qquad e = \boldsymbol{\sigma_s} - \boldsymbol{\sigma} \qquad e \qquad V_e, \text{ o volume do elemento finito.} \tag{5.3}$$

Desta forma,

$$\frac{\partial F}{\partial \sigma_i^{(e)}} = 0 \tag{5.4}$$

remetendo a,

$$\boldsymbol{S}\ \hat{\boldsymbol{\sigma}}_{\boldsymbol{s}} = \boldsymbol{f} \tag{5.5}$$

onde:

$\boldsymbol{S}$ é a **matriz de suavização global**, dada por:

$$S_{ij}^{(e)} = \int_{Ve} \boldsymbol{N}_i^T \boldsymbol{N}_j dV \tag{5.6}$$

e $\boldsymbol{f}$ é o vetor de forças, dado por:

$$f_i^{(e)} = \int_{Ve} \boldsymbol{N}_i^T \boldsymbol{D}\boldsymbol{B}\boldsymbol{d}^{(e)} dV \ . \tag{5.7}$$

sendo :

$\boldsymbol{D}$, a matriz constitutiva do modelo;

$\boldsymbol{B}$, a matriz das deformações generalizadas;

$\boldsymbol{d}^{(e)}$, o vetor de deslocamentos nodais do elemento.

Por fim tem-se as tensões nodais suavizadas para todos os nós do modelo, dadas por:

$$\boldsymbol{\sigma_s} = \boldsymbol{S}^{-1}\boldsymbol{f} \ \ . \tag{5.8}$$

5.3.2 Suavização Local

A suavização local trata do problema de suavização no domínio de somente um elemento. Para tanto, são usados os valores das grandezas nos pontos de integração e uma **matriz de extrapolação**, que é obtida através de uma extrapolação bilinear no sistema de coordenadas do elemento. Com esta matriz, obtém-se os resultados nodais através da equação:

$$\boldsymbol{\sigma_s^{(e)}} \ = \ \boldsymbol{S} \ \boldsymbol{\sigma^e} \tag{5.9}$$

onde $\boldsymbol{\sigma_s^{(e)}}$ é o vetor dos valores suavizados, $\boldsymbol{S}$ é a matriz de suavização e $\boldsymbol{\sigma^{(e)}}$ é o vetor dos valores nos pontos de integração.

Por exemplo, em uma quadratura com 2x2 pontos de integração, para o elemento bilinear da figura 5.4, tem-se:

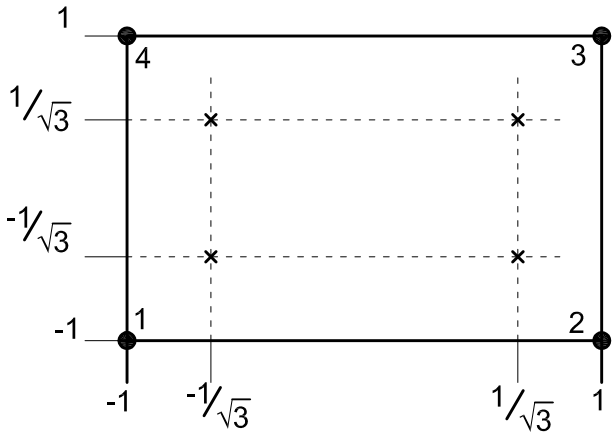


Figura 5.4: Elemento Bilinear

$$f(\xi,\eta) = a_0 + a_1\xi + a_2\eta + a_3\xi\eta \tag{5.10}$$

$$f(\xi,\eta) = \left[\begin{array}{cccc} 1 & \xi & \eta & \xi\eta \end{array}\right] \left\{\begin{array}{c} a_0 \\ a_1 \\ a_2 \\ a_3 \end{array}\right\} \tag{5.11}$$

ou

$$f(\xi,\eta) = \boldsymbol{\varphi}\ \boldsymbol{\alpha}\ , \tag{5.12}$$

com: $\boldsymbol{\varphi} = \left[\begin{array}{cccc} 1 & \xi & \eta & \xi\eta \end{array}\right]$ e $\boldsymbol{\alpha} = \left\{\begin{array}{c} a_0 \\ a_1 \\ a_2 \\ a_3 \end{array}\right\}$.

Aplicando 5.12 aos quatro pontos de integração (figura 5.4), tem-se:

$$f_1 = \boldsymbol{\varphi_1}\ \boldsymbol{\alpha} = \left[\begin{array}{cccc} 1 & \frac{-1}{\sqrt{3}} & \frac{-1}{\sqrt{3}} & \frac{1}{3} \end{array}\right] \boldsymbol{\alpha} \tag{5.13}$$

$$f_2 = \boldsymbol{\varphi_2}\ \boldsymbol{\alpha} = \left[\begin{array}{cccc} 1 & \frac{1}{\sqrt{3}} & \frac{-1}{\sqrt{3}} & \frac{-1}{3} \end{array}\right] \boldsymbol{\alpha} \tag{5.14}$$

$$f_3 = \boldsymbol{\varphi_3}\ \boldsymbol{\alpha} = \left[\begin{array}{cccc} 1 & \frac{-1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{-1}{3} \end{array}\right] \boldsymbol{\alpha} \tag{5.15}$$

$$f_4 = \boldsymbol{\varphi_4}\ \boldsymbol{\alpha} = \left[\begin{array}{cccc} 1 & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{3} \end{array}\right] \boldsymbol{\alpha} \tag{5.16}$$

ou

$$\boldsymbol{f_{PG}} = \boldsymbol{A_{PG}}\ \boldsymbol{\alpha} \tag{5.17}$$

com:

$$\boldsymbol{f_{PG}} = \left\{ \begin{array}{c} f_1 \\ f_2 \\ f_3 \\ f_4 \end{array} \right\} \text{ e } \boldsymbol{A_{PG}} = \left[\begin{array}{cccc} 1 & \frac{-1}{\sqrt{3}} & \frac{-1}{\sqrt{3}} & \frac{1}{3} \\ 1 & \frac{1}{\sqrt{3}} & \frac{-1}{\sqrt{3}} & \frac{-1}{3} \\ 1 & \frac{-1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{-1}{3} \\ 1 & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{3} \end{array} \right]$$

Invertendo 5.17, tem-se

$$\boldsymbol{\alpha} = \boldsymbol{A_{PG}}^{-1}\ \boldsymbol{f_{PG}} \tag{5.18}$$

Para os pontos nodais, têm-se

$$\boldsymbol{f_{PN}} = \left[\begin{array}{cccc} 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \end{array} \right] \boldsymbol{\alpha} \tag{5.19}$$

ou

$$\boldsymbol{f_{PN}} = \boldsymbol{A_{PN}}\ \boldsymbol{\alpha} \tag{5.20}$$

com:

$$\boldsymbol{A_{PN}} = \left[\begin{array}{cccc} 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \end{array} \right]$$

Substituindo 5.18 em 5.19

$$\boldsymbol{f_{PN}} = \boldsymbol{A_{PN}}\ \boldsymbol{A_{PG}}^{-1}\ \boldsymbol{f_{PG}} \tag{5.21}$$

com:

$$\boldsymbol{A_{PN}}\ \boldsymbol{A_{PG}}^{-1} = \begin{bmatrix} 1+\frac{\sqrt{3}}{2} & \frac{-1}{2} & 1-\frac{\sqrt{3}}{2} & \frac{-1}{2} \\ \frac{-1}{2} & 1+\frac{\sqrt{3}}{2} & \frac{-1}{2} & 1-\frac{\sqrt{3}}{2} \\ 1-\frac{\sqrt{3}}{2} & \frac{-1}{2} & 1+\frac{\sqrt{3}}{2} & \frac{-1}{2} \\ \frac{-1}{2} & 1-\frac{\sqrt{3}}{2} & \frac{-1}{2} & 1+\frac{\sqrt{3}}{2} \end{bmatrix}$$

Verifica-se que a suavização local apresenta uma forma mais simples de se obter resultados nodais suavizados. Entretanto, o fato de ser em cada elemento separadamente não soluciona o problema da continuidade das grandezas ao longo da malha. Mas, com os valores suavizados em cada elemento, os resultados nodais podem ser obtidos com continuidade por uma média aritmética dos valores coincidentes em um mesmo nó.

5.4 Funções de Interpolação

A suavização de resultados usando funções de interpolação consiste no uso de funções matemáticas, de preferência polinomiais bicúbicas, para a representação das grandezas a serem aproximadas. As funções polinomiais são escolhidas por apresentarem continuidade e suavidade.

Os valores a serem interpolados são os valores de grandezas avaliadas nos pontos de Gauss dos elementos de uma malha. A função interpoladora é definida por seu valor nos pontos e por sua derivada e, como são usados todos os valores dos pontos de integração da malha, a continuidade entre elementos é garantida. A figura 5.5 ilustra o uso de funções de interpolação. Note-se que em domínios bidimensionais o polinômio tem o mesmo comportamento nas direções X e Y.

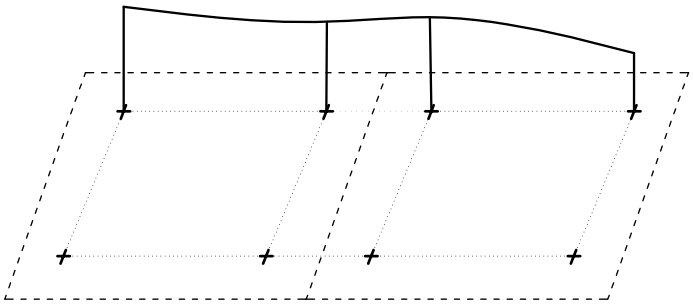


Figura 5.5: Funções de interpolação em domínios de malhas do MEF

Para que a suavização seja em toda a malha deve-se ainda extrapolar os valores da função de interpolação para os nós do contorno da malha do modelo. A eficácia deste método depende de uma malha regular com elementos uniformemente espaçados, o que normalmente não ocorre em modelos do MEF. Outro fator que torna o método de interpolação pouco prático é que nem sempre são conhecidos os valores das funções e de suas derivadas com a análise por elementos finitos.

5.5 Método de Suavização Implementado

No pós-processador, foi adotado um método de suavização de grandezas baseado em uma suavização local. O método parte de uma extrapolação para os nós dos valores obtidos nos pontos de integração e, em seguida, usa uma média nodal, quando necessário. Também foi implementado o cálculo dos valores sem a extrapolação, avaliando os mesmos diretamente nos nós. Neste caso, a suavização também é uma média nodal simples. Dispinibilizou-se, ainda, a visualização descontínua, usando valores dos elementos sem extrapolar para os nós.

O método adotado usa somente como parâmetro o número de pontos de integração de cada elemento, deixando o processo bem genérico, uma vez que independe das funções de forma ou do número de nós do elemento.

5.5.1 Descrição do Método de Suavização

O primeiro passo do método de suavização adotado é determinar suas equações de extrapolação. Portanto seja

$$\boldsymbol{\sigma} = \boldsymbol{\varphi_n}\ \boldsymbol{\alpha} \tag{5.22}$$

onde:

$\boldsymbol{\sigma}$ é o valor a ser extrapolado em um ponto do elemento;

$\boldsymbol{\varphi_n}$ são os termos em coordenadas naturais do polinômio usado para a extrapolação;

$\boldsymbol{\alpha}$ são os coeficientes polinomiais.

Escrevendo a equação anterior na forma matricial, com $\boldsymbol{\sigma}$ avaliado nos pontos de integração tem-se

$$\boldsymbol{\sigma_{IP}} = \boldsymbol{G}\ \boldsymbol{\alpha} \tag{5.23}$$

Isolando o termo dos coeficientes $\boldsymbol{\alpha}$, a equação fica:

$$\boldsymbol{\alpha} = \boldsymbol{G}^{-1}\ \boldsymbol{\sigma_{IP}} \tag{5.24}$$

Substituindo na equação original, para um ponto genérico, obtém-se

$$\boldsymbol{\sigma} = \boldsymbol{\varphi}_{1\times N}\boldsymbol{G}^{-1}_{N\times N}\boldsymbol{\sigma}_{IPN\ \times 1}, \tag{5.25}$$

sendo N o número de pontos de integração do elemento. Por fim, obtém-se o valor extrapolado para qualquer ponto do elemento, baseado nas coordenadas dos pontos de integração e seus respectivos valores, por:

$$\boldsymbol{\sigma_{PN}} = \boldsymbol{\varphi}_{PN}\ \boldsymbol{G}^{-1}\ \boldsymbol{\sigma}_{IP} \tag{5.26}$$

onde:

$\boldsymbol{\sigma_{PN}}$ é o valor extrapolado em um ponto do elemento, no caso um ponto nodal por exemplo;

$\boldsymbol{\varphi_{PN}}$ é o vetor com os termos polinomiais avaliada em um ponto qualquer do elemento.

$\boldsymbol{G}^{-1}$ é a matriz de extrapolação;

$\boldsymbol{\sigma_{IP}}$ é o vetor dos valores a serem extrapolados, avaliados nos pontos de integração.

A definição do polinômio usado na extrapolação é feita a partir do número de pontos de integração, segundo o triângulo de Pascal. Para uma integração com 3 × 3 pontos, os termos polinomiais são definidos como mostrado na figura 5.6.

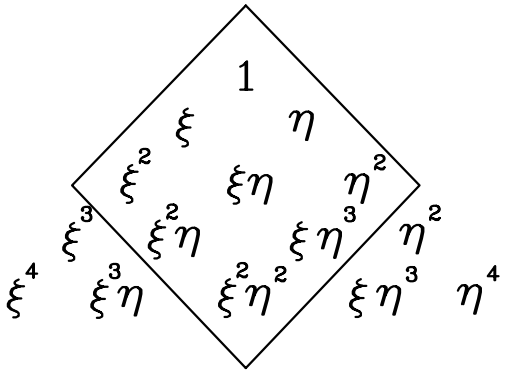


Figura 5.6: Definição dos termos do polinômio de extrapolação no triângulo de Pascal

O polinômio fica, assim, definido por:

$$f(\xi,\eta) = a_0 + a_1\xi + a_2\xi^2 + a_3\eta + a_4\xi\eta + a_5\xi^2\eta + a_6\eta^2 + a_7\xi\eta^2 + a_8\xi^2\eta^2 \qquad (5.27)$$

De posse do polinômio é obtida a matriz auxiliar $\boldsymbol{\varphi}_{\xi\eta}$, dada por

$$\boldsymbol{\varphi}_{\xi\eta} = \boldsymbol{\varphi}_{\xi}\ \boldsymbol{\varphi}_{\eta} \qquad (5.28)$$

com $\boldsymbol{\varphi}_{\xi}$ e $\boldsymbol{\varphi}_{\eta}$, vetores com os termos em ξ e em η do polinômio. Logo,

$$\boldsymbol{\varphi}_{\xi\eta} = \boldsymbol{\varphi}_{\xi}^{T}\ \boldsymbol{\varphi}_{\eta} = \begin{bmatrix} 1 \\ \xi \\ \xi^2 \end{bmatrix} \begin{bmatrix} 1 & \eta & \eta^2 \end{bmatrix} \qquad (5.29)$$

$$\boldsymbol{\varphi}_{\xi\eta} = \begin{bmatrix} 1 & \eta & \eta^2 \\ \xi & \xi\eta & \xi\eta^2 \\ \xi^2 & \xi^2\eta & \xi^2\eta^2 \end{bmatrix} \qquad (5.30)$$

Para obter o vetor $\boldsymbol{\varphi}$ de equação 5.22 basta organizar os termos da matriz $\boldsymbol{\varphi}_{\xi\eta}$ em

um vetor linha como mostrado a seguir.

$$\boldsymbol{\varphi} = \left[\begin{array}{cccccccccc} 1 & \xi & \xi^2 & \eta & \xi\eta & \xi^2\eta & \eta^2 & \xi\eta^2 & \xi^2\eta^2 \end{array} \right] \tag{5.31}$$

A montagem da matriz de extrapolação $\boldsymbol{G}$ parte dos termos de $\boldsymbol{\varphi}$ organizados da seguinte forma

$$\boldsymbol{G} = \left[\begin{array}{cccc} \varphi_i & \varphi_{i+1} & \cdots & \varphi_n \\ \varphi_i^2 & \varphi_{i+1}^2 & \cdots & \varphi_n^2 \\ \vdots & \vdots & \vdots & \vdots \\ \varphi_i^n & \varphi_{i+1}^n & \cdots & \varphi_n^n \end{array} \right], \tag{5.32}$$

com i variando de 0 a N.

A definição dos polinômios foi apresentada para o caso bidimensional, mas os casos unidimensional e tridimensional também foram implementados. A única modificação foi a definição da matriz auxiliar. Apresenta-se a seguir esta matriz para todos os casos implementados.

$$\boldsymbol{\varphi}_\xi = \left[\begin{array}{cccc} \xi^i & \xi^{i+1} & \ldots & \xi^n \end{array} \right], \tag{5.33}$$

para o caso unidimensional,

$$\boldsymbol{\varphi}_{\xi\eta} = \boldsymbol{\varphi}_\xi^T \, \boldsymbol{\varphi}_\eta = \left[\begin{array}{c} \xi^i \\ \xi^{i+1} \\ \vdots \\ \xi^n \end{array} \right] \left[\begin{array}{cccc} \eta^i & \eta^{i+1} & \ldots & \eta^n \end{array} \right], \tag{5.34}$$

para o caso bidimensional e

$$\boldsymbol{\varphi}_{\xi\eta\zeta} = \boldsymbol{\varphi}_{\xi\eta}^T \, \boldsymbol{\varphi}_\zeta = \boldsymbol{\varphi}_{\xi\eta}^T \left[\begin{array}{cccc} \zeta^i & \zeta^{i+1} & \ldots & \zeta^n \end{array} \right], \tag{5.35}$$

para o caso tridimensional.

Capítulo 6

REPRESENTAÇÃO GRÁFICA DE GRANDEZAS DO MEF

6.1 Introdução

A representação das grandezas obtidas pelo processamento de modelos do MEF é feita por isocurvas ou isofaixas.

As isocurvas são curvas traçadas no domínio do modelo representando valores constantes da grandeza avaliada. As isofaixas, de forma análoga, são faixas de valores com uma determinada variação, dependendo do gradiente da grandeza avaliada.

6.2 Traçado de Isocurvas por Subdivisão Triangular do Domínio

A técnica consiste na subdivisão do domínio em triângulos menores sendo que a variação dos resultados dentro de cada triângulo é linear, como mostrado na figura 6.1.

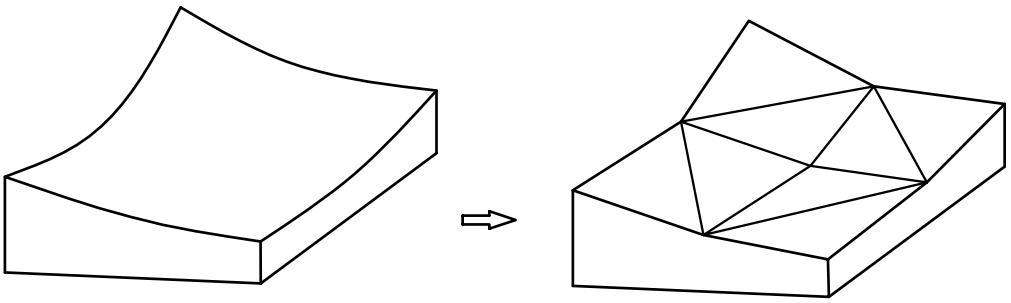


Figura 6.1: Linearização em domínios triagulares

Em cada triângulo, o traçado das isocurvas é feita a partir dos valores calculados nos vértices de cada triângulo, no interior de cada elemento do modelo (figura 6.2).

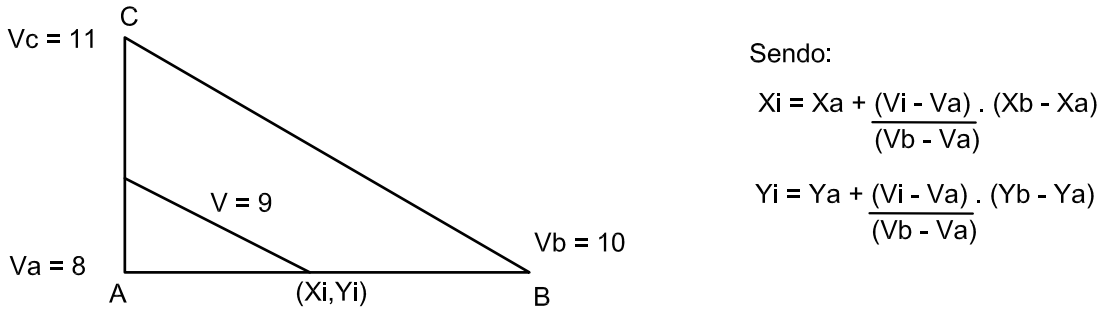


Figura 6.2: Traçado das isocurvas.

6.3 Traçado de Isocurvas pela Técnica da Tangente

A técnica da tangente constitui uma interpolação não-linear, baseada na formulação paramétrica do elemento e na seqüência mapeada para o sistema global do modelo.

Seja a equação da grandeza a ser representada

$$f(\xi, \eta) = N(\xi, \eta) d^{(e)}, \tag{6.1}$$

onde:

$N(\xi, \eta)$ é uma matriz contendo as funções interpoladoras;

$d^{(e)}$ é um vetor com grandezas nodais.

Por definição, a isocurva apresenta valores constantes, logo

$$f(\xi, \eta) = constante \qquad e \qquad df(\xi, \eta) = 0 \tag{6.2}$$

Expandindo a derivada tem-se

$$\frac{\partial f}{\partial \xi} d\xi + \frac{\partial f}{\partial \eta} d\eta = 0 \quad , \tag{6.3}$$

sendo que os valores de $d\xi$ e $d\eta$ são componentes de um passo dl pré-definido. O procedimento pode ser visto na figura 6.3(a).

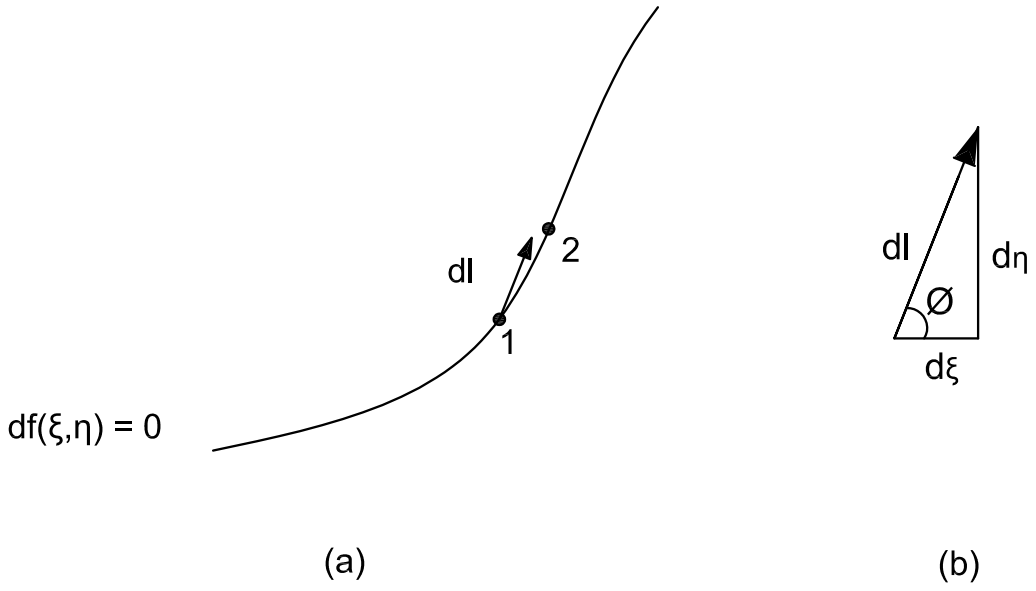


Figura 6.3: Traçado das isocurvas

A partir do passo e da tangente do ângulo ϕ (figura 6.3(b)), pode-se calcular as coordenadas do ponto seguinte.

$$\tan \phi = |\frac{d\eta}{d\xi}| \quad , \tag{6.4}$$

sendo que

$$|\frac{d\eta}{d\xi}| = \frac{\partial f}{\partial \xi}\frac{\partial \eta}{\partial f} \quad . \tag{6.5}$$

Portanto, para o traçado de uma isocurva é necessário: um ponto de partida cujas coordenadas são conhecidas, o valor das funções de interpolação e de suas derivadas no ponto anterior, os valores nodais e o valor do incremento dl.

Esta técnica, embora eficiente, exige um acoplamento do modelo numérico com o pós-processador, uma vez que é necessário usar as funções de interpolação dos elementos do modelo.

6.4 Traçado de Isocurvas pela Técnica da Inversão

A técnica da inversão, assim como a técnica anterior, baseia-se na formulação paramétrica do elemento e em uma interpolação não-linear.

A partir da equação que aproxima a grandeza analisada e da definição de isocurva, $f(\xi, \eta) = constante$, tem-se:

$$f(\xi, \eta) = N(\xi, \eta)d^{(e)} = C = constante \tag{6.6}$$

Portanto, pela inversão do sistema de equações, expressando ξ em função de η, como mostrado na equação 6.7, abaixo, pode-se obter as coordenadas dos pontos da isocurva, caminhando pequenos passos incrementais ($d\eta$), ao longo de η, e calculando o valor de ξ correspondente.

$$\xi(\eta) = N(\eta)d^{(e)} + C \tag{6.7}$$

O método, embora simples, depende da inversão do sistema de equações, o que pode ser um problema, dependendo da dimensão de N.

6.4.1 Traçado de Isofaixas por Subdivisão do Domínio

O traçado de isofaixas por subdivisão do domínio baseia-se na geração de uma nova malha interna aos elementos do modelo, de maneira que, cada sub-elemento, assuma um valor que é calculado a partir dos valores nodais da grandeza a ser aproximada (figura 6.4).

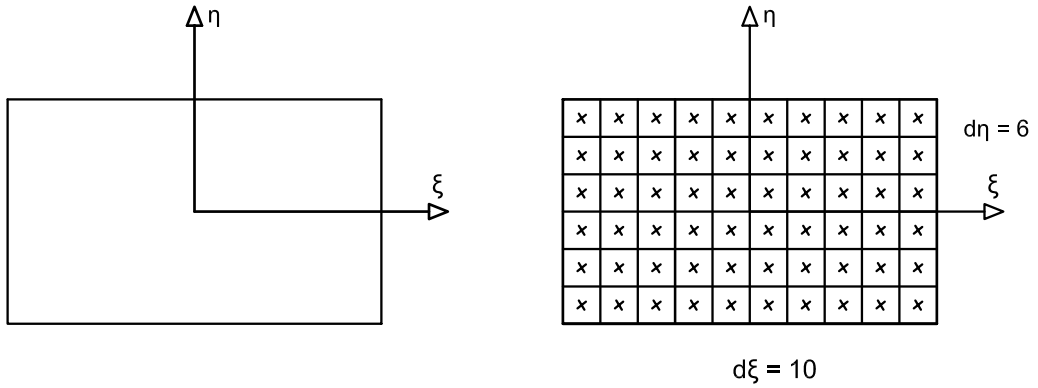


Figura 6.4: Traçado das Isofaixas por subdivisão do domínio.

O valor de cada sub-elemento é calculado no centro do respectivo sub-elemento, e a cada valor é atribuído uma cor. Desta forma a qualidade da representação depende do refinamento da nova malha.

6.5 Traçado de Isofaixas por Rastreamento

A técnica de rastreamento é uma técnica não-linear de interpolação semelhante à técnica da tangente para isocurvas. O procedimento agora consiste na atribuição de uma cor para cada ponto da tela de visualização, contido no domínio do modelo.

Para tanto, são usadas as linhas de rastreamento ou *scan lines*. O traçado das isofaixas por este método ocorre sob três sistemas de coordenadas:

1. O sistema de coordenadas local de cada elemento do modelo (figura 6.5(a));

2. O sistema de coordenadas do gráfico (figura 6.5(b));

3. O sistema de coordenadas da tela de visualização (figura 6.5(c)).

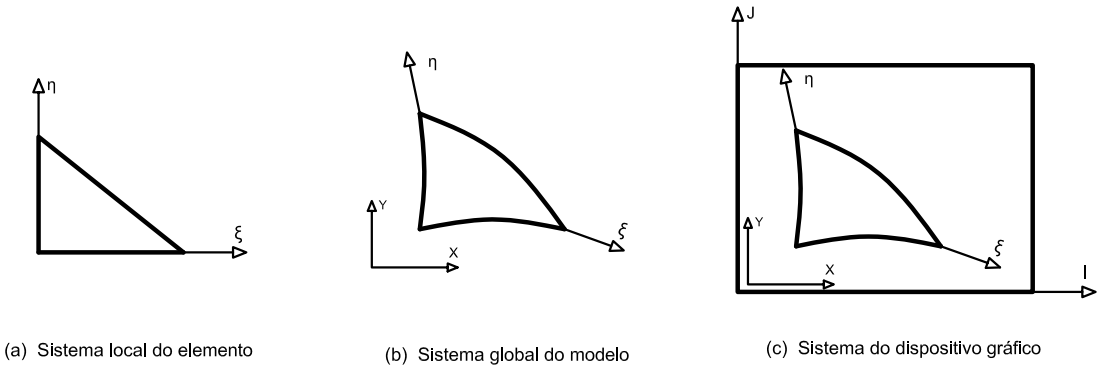


Figura 6.5: Sistemas de coordenadas para traçado de faixas por Rastreamento

Como o problema é paramétrico, as grandezas podem ser representadas no sistema do gráfico pelas próprias funções de interpolação do elemento. Portanto, conhecendo as coordenadas do gráfico (I,J) e o valor da função f, pode-se definir uma cor nos pontos da tela para os valores correspondentes. Desta forma, as coordenadas do ponto (I,J) são dadas por:

$$I(\xi, \eta) = N(\xi, \eta) I^{(e)} \tag{6.8}$$

$$J(\xi, \eta) = N(\xi, \eta) J^{(e)} \tag{6.9}$$

onde:

$N(\xi, \eta)$ é a matriz com as funções de interpolação;

$I^{(e)}, J^{(e)}$ são os valores das coordenadas nodais no sistema do gráfico.

A qualidade da representação depende diretamente do incremento das coordenadas ξ e η no sistema local do elemento. A técnica, da forma em que foi apresentada, também exige um acoplamento entre o modelo numérico e o pós-processador.

6.6 Técnica de Representação de Resultados Implementada na Aplicação

A técnica usada para a representação de grandezas baseia-se no traçado de isofaixas utilizando uma subdivisão de domínios. A subdivisão é feita de acordo com o número de faixas (ou cores) usadas na representação e os valores máximos e mínimos calculados no modelo. O método foi adaptado para o programa a partir da implementação realizada por Martha et al. (1996).

Dada uma malha de elementos finitos, deseja-se visualizar um determinado valor obtido no processamento. Sendo assim, a técnica de representação pode usar o domínio dos próprios elementos finitos para o traçado dos contornos ou usar uma subdivisão triangular, baseada na triangulação de Delaunay, elemento por elemento, para gerar uma segunda malha, usada somente na plotagem dos contornos.

O detalhamento da representação depende diretamente da discretização do modelo. O gradiente de cores e o número de faixas usados também influem na visualização, uma vez que os limites dos valores são calculados tendo em mãos o número de contornos.

O traçado das faixas de valores inicia-se ao ser atribuído o número de faixas e os valores máximo e mínimo da grandeza que se quer representar. Em seqüência, são calculados limites de cada faixa de valor.

Estando os valores limites disponíveis, são percorridos cada um dos subdomínios (sejam os elementos da malha ou os triangulos da subdivisão), fazendo-se uma interpolação de valores, conforme visto na figura 6.2, para cada vértice do respectivo subdomínio. A interpolação destes valores é feita de acordo com os limites de cada faixa de valor obtidos anteriormente. O resultado desta interpolação pode ser vista na figura 6.6.

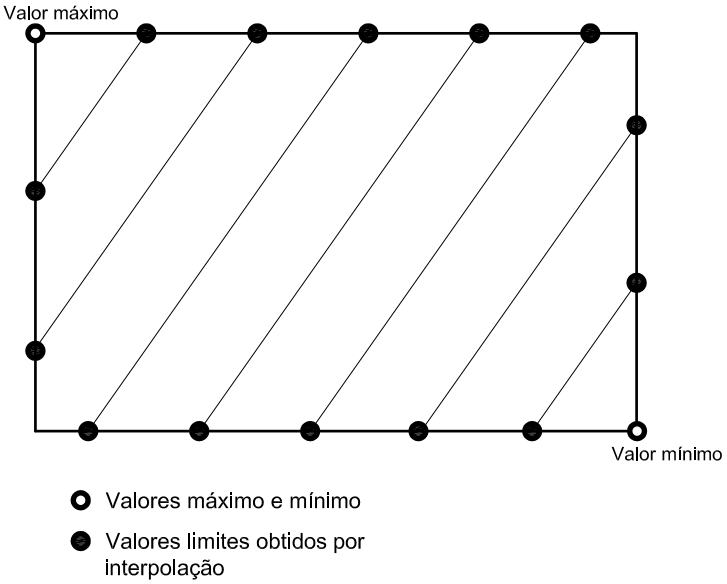


Figura 6.6: Obtenção de faixas do contorno por interpolação de valores.

Após o processo de interpolação, para cada faixa criada, atribui-se uma cor correspondente ao valor limite já determinado.

Capítulo 7

Pós-Processamento e Análise Não-Linear via MEF

7.1 Análise Não-Linear via MEF

Em mecânica estrutural, um problema é dito não-linear se a rigidez depende dos deslocamentos da estrutura. Esta dependência é dita geometricamente não-linear quando a rigidez é afetada por um estado excessivo de deformação, sem influência no comportamento do material. Quando a resposta do material é dependente do estado de deformação a que o mesmo é submetido, o problema é tratado como fisicamente não-linear (Fuina, 2004).

O Método dos Elementos Finitos (MEF) é usado para modelagem e solução de inúmeros problemas de engenharia. A formulação destes problemas resulta em um sistema de equações algébricas lineares ou não-lineares que devem ser solucionados. Os processos de solução são conhecidos como métodos de obtenção de trajetórias de equilíbrio.

As trajetórias de equilíbrio são obtidas através de processos incrementais-iterativos nas variáveis do problema. Tal processo gera resultados do modelo analisado a cada passo de execução.

O método tem seu início estabelecendo-se um incremento do fator de carga, em função de um parâmetro de controle, que varia de acordo com a metodologia usada, obtendo-se assim um vetor de deslocamentos para cada iteração de cada

passo do processo, verificando-se um critério de convergência no final da iteração. O procedimento é finalizado no momento em que a convergência é atingida.

7.2 Pós-Processamento e a Análise Não-Linear

Após a solução não-linear do modelo de elementos finitos, obtida convergência e de posse dos resultados, um programa de pós-processamento deve ser capaz de interpretar os resultados de grandezas variáveis ao longo do tempo e daquelas variáveis a cada passo da análise não-linear. Para tanto, o programa deve ter recursos gráficos interativos capazes de exibir os resultados do problema.

O pós-processamento deve percorrer todos os passos da análise e, através dos dados gerados durante o processamento, ser capaz de exibir os resultados das grandezas nodais e dos elementos para qualquer momento da análise. A visualização das trajetória de equilíbrio ou da variação de qualquer grandeza envolvida no processo também deve ser compreendida na etapa do pós-processamento.

Capítulo 8

PADRÕES DE PROJETO DE SOFTWARES

8.1 Introdução

A arquitetura do sistema INSANE é baseada em uma combinação dos padrões de projeto de software *Model-View-Controller* (MVC), *Command* e *Observer*, como ilustrado na figura 8.1.

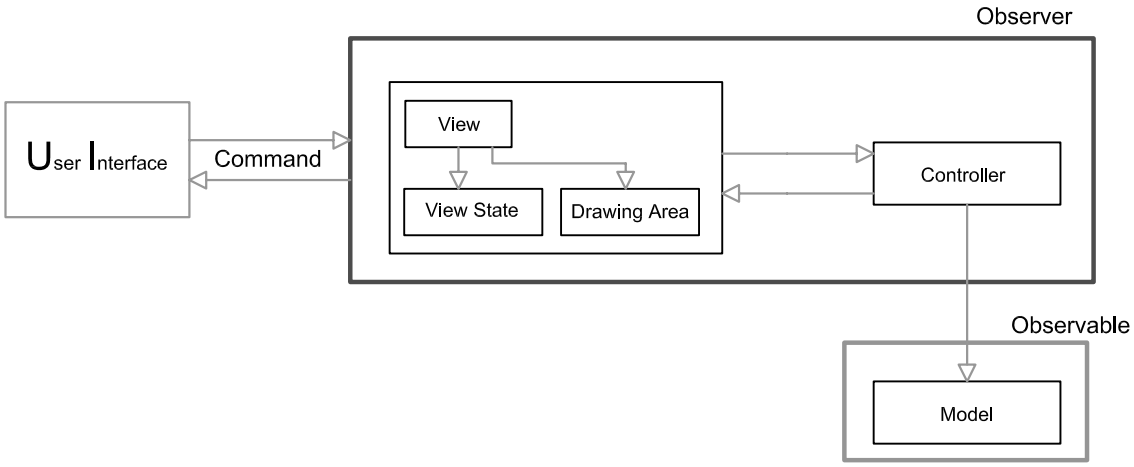


Figura 8.1: Arquitetura do INSANE.

Este capítulo discute cada um destes padrões e, ao final, detalha a combinação dos mesmos, adotada no pós-processador.

8.2 Padrão *Model-View-Controller* (MVC)

Visando separar o modelo de sua representação, a implementação do INSANE é baseada no padrão de projeto *Model-View-Controller* (*MVC*). A utilização desta metáfora de programação permite que o controle da interação com o usuário e a visualização das respostas sejam implementados independentemente do modelo adotado, minimizando as tarefas de manutenção e expansão da aplicação. A implementação segundo o padrão *MVC* permite o aperfeiçoamento gradual da aplicação através de mudança de plataforma, criação de diversas vistas sincronizadas com o modelo, substituição ou atualização das diversas vistas e disponibilização "*on-line*" do sistema.

Existe um ciclo de vida para cada uma das atividades executadas pelo programa. Este ciclo de vida permite que o usuário faça alterações no modelo e visualize o resultado a cada alteração, até que consiga o resultado desejado. O referido ciclo compõe-se de: especificação do usuário, atualização do modelo e visualização.

O padrão de projeto *Model-View-Controller* pode ser usado para a implementação do ciclo de vida de cada atividade. Este padrão divide a aplicação em três componentes: *modelo*, *vista*, e *controlador*. A figura 8.1 ilustra o padrão (Brugiolo, 2004).

O *modelo* contém o núcleo dos dados e da funcionalidade do sistema, sendo independente das saídas e entradas de dados. A *vista* apresenta para o usuário as informações armazenadas no modelo. Cada *controlador* é associado a um componente *vista*, sendo o responsável pela percepção das entradas do usuário e tradução das mesmas em requisições de serviços para os componentes *modelo* e *vista*. Todas as requisições dos usuários devem ser feitas através dos controladores (Buschmann, F., Meunier, R., Rohnert, H., Sommerland, P. & Stal, M., 1995).

8.3 Padrão *Command*

O uso do padrão *Command* na implementação do sistema permite o encapsulamento de rotinas de execução em objetos, a associação destes objetos a elementos de interface gráfica com o usuário (GUI) e dispositivos de entrada (teclado e "*mouse*"), a execução de uma mesma rotina disparada por diferentes elementos de GUI e possibilita um incremento na modularidade de seu código. O encapsulamento das rotinas de execução possibilita também que a realização de alterações nas mesmas não provoque modificações nas classes existentes (Brugiolo, 2004).

O padrão *Command* baseia-se em uma classe abstrata de mesmo nome, a qual declara uma interface para execução de operações. Na sua forma mais simples, esta interface inclui uma operação abstrata **execute()**. As subclasses concretas de *Command* especificam um par receptor-ação através do armazenamento do receptor como uma variável de instância e pela implementação de **execute()**, para invocar a solicitação. O receptor tem o conhecimento necessário para poder executar a solicitação.

Este padrão desacopla o objeto que invoca a operação daquele que tem o conhecimento para executá-la. Isto proporciona grande flexibilidade no projeto da interface de usuário. Uma aplicação pode oferecer tanto uma interface gráfica com *menus* como uma interface gráfica com *botões* para algum recurso seu, simplesmente fazendo com que o *menu* e o *botão* compartilhem uma instância da mesma classe que implementa *Command*. O padrão *Command* possibilita a substituição dinâmica de comandos, o que é muito útil para interfaces gráficas sensíveis ao contexto. Pode-se ainda concatenar comandos para compor comandos maiores, propiciando a redução da complexidade destes. Todos estes recursos são possíveis porque o objeto que emite a solicitação não precisa conhecer a execução da solicitação.

O tempo de vida de um objeto *Command* é independente de sua solicitação original, permitindo armazenar comandos e executar suas rotinas em momentos distintos. A operação **execute()**, de *Command*, pode armazenar estados para que o comando

possa reverter seus efeitos. Para suportar a operação desfazer a interface de *Command* deve acrescentar a operação **undo()**, que reverte os efeitos de uma chamada anterior de **execute()**. Os comandos executados devem ser armazenados em uma lista histórica. O nível ilimitado de desfazer e refazer operações é obtido percorrendo esta lista para trás e para frente, chamando operações **undo()** e **execute()**, respectivamente (Gamma et al., 1995).

A figura 8.2 apresenta os componentes envolvidos com o padrão *Command* e ilustra o relacionamento entre eles. *Receiver* é o componente que sabe como executar as operações associadas a uma solicitação, podendo qualquer classe funcionar como um *Receiver*. A classe *Command* declara uma interface para execução de uma operação. *ConcreteCommand* implementa o processo **execute()** através da invocação das operações correspondentes no *Receiver* e define uma vinculação entre um objeto *Receiver* e uma ação. Quando os comandos podem ser desfeitos, *ConcreteCommand* armazena estados para desfazer o comando antes de invocar **execute()**. O componente *Invoker* é responsável por disparar o processo **execute()** de seu comando associado (Gamma et al., 1995). O componente *Client* cria um objeto *ConcreteCommand*, o associa a um *Invoker* e estabelece o seu receptor (*Receiver*).

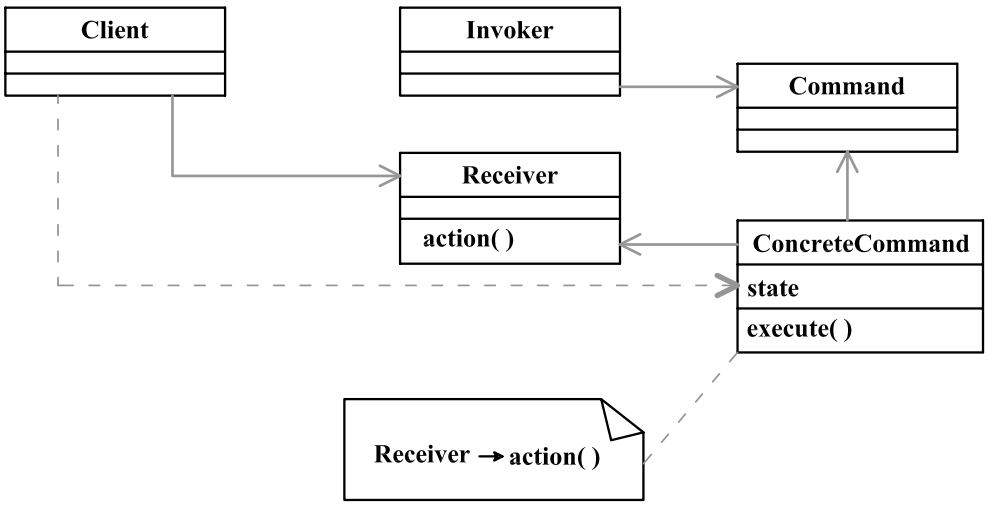


Figura 8.2: Estrutura do padrão *Command* .

8.4 Padrão Observer

O padrão *Observer* fornece um mecanismo de propagação de mudanças que garante a consistência e a comunicação entre os componentes *controlador* e *vista* com o componente *modelo*, do padrão *MVC*. No momento em que um componente *controlador* ou *vista* é criado, o mecanismo de propagação de mudanças efetua seu registro, ligando-o ao modelo do qual ele é dependente. Os componentes *vista* e *controlador* dependem desse registro para serem informados das atualizações do modelo.

O mecanismo de propagação de mudanças é disparado a cada mudança de estado do modelo, acarretando na execução do procedimento de atualização do componente *vista*, que exibe ao usuário a informação atualizada. Cada *vista* é associada a um único *controlador* e fornece a este a funcionalidade necessária para manipular a exibição de dados. A figura 8.3 ilustra o padrão.

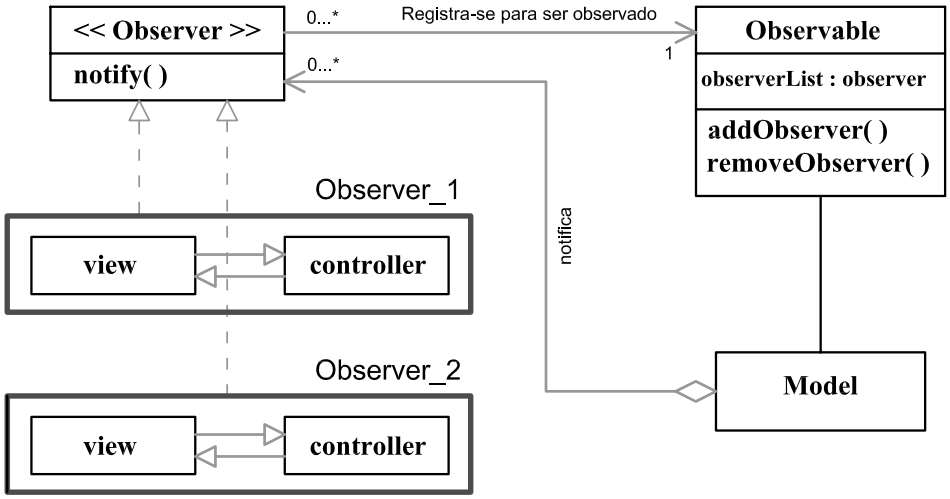


Figura 8.3: Padrão Observer.

8.5 Padrões de Projeto de Software Adotados no Pós-processador

Consistentemente com a arquitetura básica do INSANE (figura 8.1), o pós-processador também utiliza uma combinação dos padrões de projeto *MVC*, *Command* e *Observer*. Esta combinação está mostrada na figura 8.4, que é um detalhamento da figura 8.1, para o caso do pós-processador. Como pode ser visto na figura 8.4, vários pares vista-controlador observam o modelo do pós-processador (PosProcModel). Este, por sua vez, observa o núcleo numérico do INSANE, no caso da execução simultânea do processamento e do pós-processamento.

A utilização da interface *Observer* e da classe *Observable*, disponíveis em JAVA, proporciona os mecanismos de propagação de mudanças necessários para a atualização das vistas do pós-processador a cada mudança no modelo.

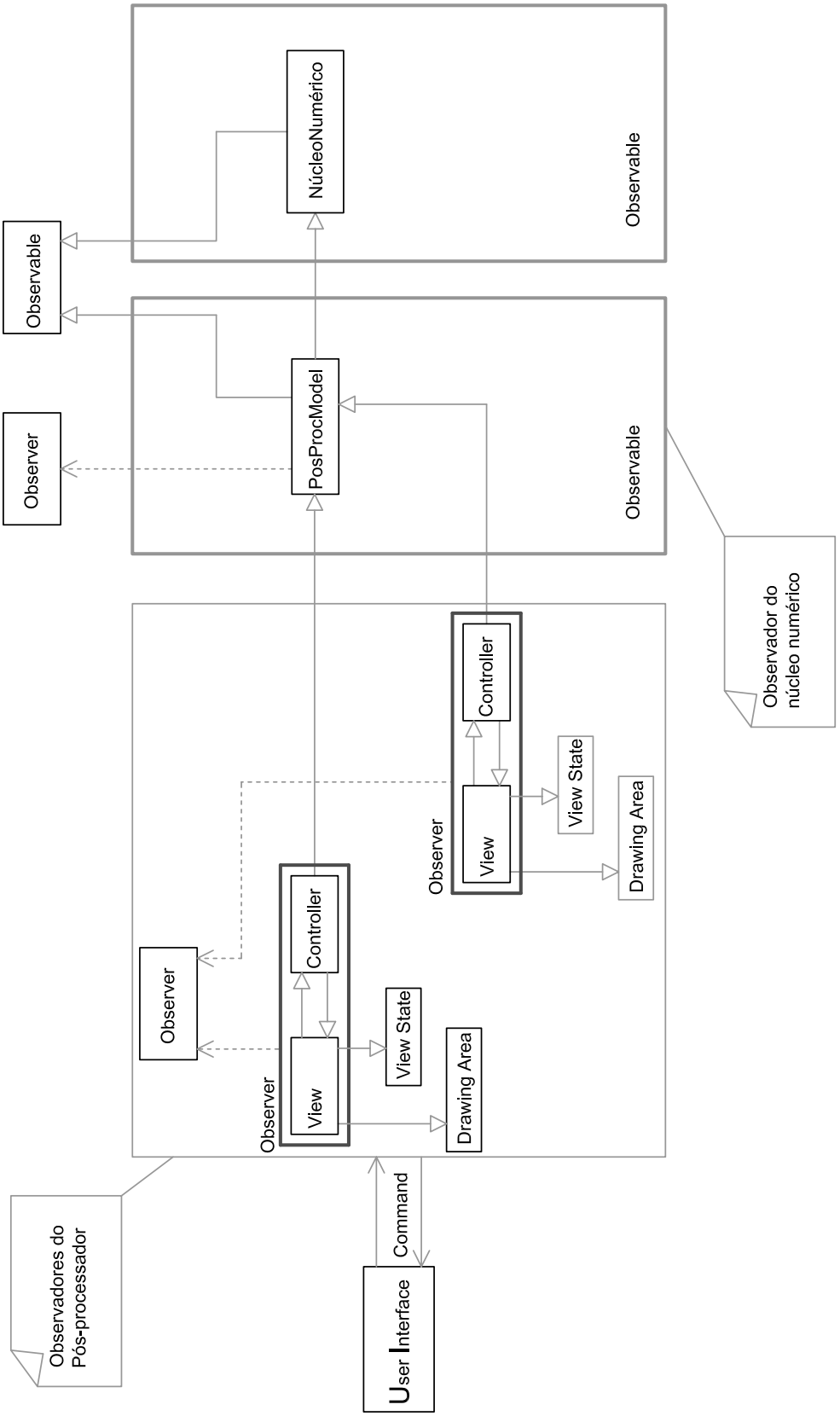


Figura 8.4: MVC-OBSERVER

Capítulo 9

Projeto Orientado a Objetos

9.1 Introdução

O pós-processador parte de uma premissa básica: a independência entre o processamento e o pós-processamento. Esta independência deve ser garantida tanto para o pós-processador de grandezas variáveis ao longo do tempo quanto para aquelas variáveis a cada passo da análise não-linear.

Deseja-se também que o pós-processamento possa ocorrer simultaneamente, ou não, ao processamento. Para o caso do pós-processamento não ocorrer simultaneamente, os dados processados serão passados para o pós-processador através de uma camada de persistência. A persistência destes dados é feita através de arquivos XML podendo, desta forma, ser compartilhada com os demais módulos do sistema. Ocorrendo o processamento simultâneo ao pós-processamento, é feito o uso de múltiplas *threads*. *Threads* são diferentes fluxos de execução do programa que são processadas simultaneamente. O recurso conhecido como *multithreading* ou *multiescalonamento* proporciona a execução do processamento em paralelo à visualização dos resultados no pós-processador.

A proposta de implementação de um pós-processador para o sistema **INSANE** demandou poucas e inevitáveis mudanças na implementação já existente do sistema.

Originalmente, a interface gráfica apresentava uma área de desenho e permitia trabalhar com um único modelo por vez. Os modelos apresentavam visualização

no plano e as aplicações de pré-processamento, processamento e pós-processamento eram integradas em uma aplicação comum.

A primeira modificação realizada foi o desenvolvimento de um aplicativo de pós-processamento independente do programa original. O desenvolvimento de aplicações independentes foi estendida aos demais módulos do sistema.

A modificação da interface com o usuário foi uma das mais significantes e desencadeou diversos aprimoramentos. A interface passou a ser uma aplicação do tipo *Desktop*, que permite visualizar múltiplas vistas simultaneamente. Tal modificação fez com que o padrão MVC (*Model-View-Controller*) fosse usado de forma mais ampla. O padrão passou a ser usado para vários pares vista-controlador, respectivo a um mesmo modelo, ou a modelos distintos. As múltiplas janelas de visualização possibilitam trabalhar com vários modelos ao mesmo tempo, o que é de grande importância também para o pré-processamento.

A implementação de operações de transformações geométricas incorporou características de visualização até então não existentes no sistema. Possibilitou visualizar modelo em planos diferentes ou em perspectivas diferentes simultaneamente, além de permitir que modelos tridimensionais também sejam exibidos.

Apresenta-se a seguir o projeto orientado a objetos do pós-processador. O núcleo numérico também será brevemente exposto. Serão utilizados diagramas UML (*Unified Modelling Language*) para descrever as principais classes, o relacionamento entre elas e a organização geral das aplicações.

9.2 Camada Modelo do Pós-Processador

O pós-processador é baseado em um modelo de semi-arestas, como já apresentado no capítulo 2. A implementação desta estrutura de dados baseou-se na proposta de Mäntylä (1987), mantendo as principais características, sendo adaptada para a linguagem JAVA. Criaram-se classes correspondentes aos níveis da hierarquia da

estrutura de adjacência (capítulo 2) e uma classe para trabalhar com listas de modelos geométricos. Estas classes são compostas por listas duplamente encadeadas, já disponíveis em JAVA, métodos de acesso e referências às respectivas camadas da estrutura de dados.

A classe **LinkedList** implementa todas as operações definidas nas interfaces **Collection** e **List**. Permite a inclusão de elementos nulos e garante que a ordem dos elementos contidos respeitará a ordem de sua inclusão. Dada a implementação interna na forma de uma lista duplamente ligada, as operações de inclusão ou remoção de elementos em qualquer posição tem desempenho superior em relação às listas obtidas com a classe **ArrayList**, também disponível na API **Collection**. Em compensação, as operações de pesquisa e interação têm desempenho mais pobre. O que determina a escolha entre as classes **ArrayList** e **LinkedList** é a natureza das operações sobre seus objetos (Jandl, 2003). No pós-processador foi adotada a classe **LinkedList**.

Nas figuras 9.1, 9.2, 9.3, 9.4, 9.5 e 9.6 são mostradas as classes correspondentes aos respectivos níveis da estrutura de adjacências e suas associações.

A classe **PlanarSubdivision** (figura 9.1) dá acesso às listas de vértices, faces e arestas do modelo geométrico, e uma referência ao modelo de semi-arestas como um todo, que possui uma lista de subdivisões planares. Esta classe representa uma subdivisão planar do modelo de semi-arestas.

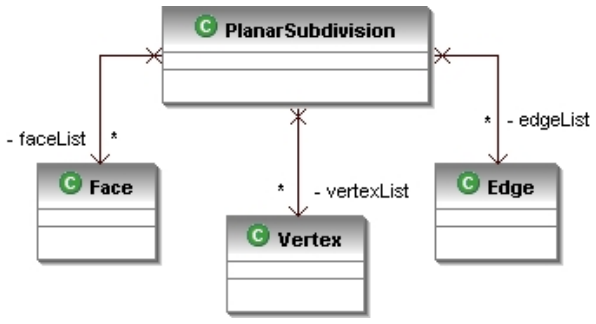


Figura 9.1: Classe para o nível da subdivisão planar.

Já nas classes **Face** (figura 9.2) e **Loop** (figura 9.3) verifica-se as respectivas

referências ao nível topológico superior. A classe **Face** possui acesso à sua lista de "loops" e a classe **Loop** possui acesso à sua lista de semi-arestas.

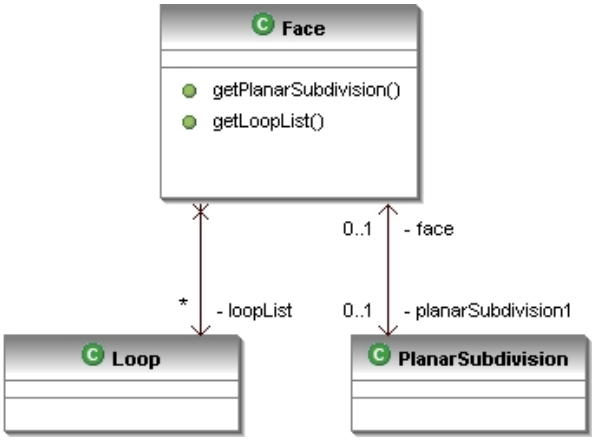


Figura 9.2: Classe para o nível da face.

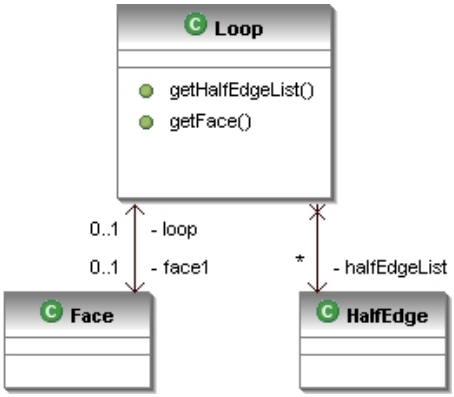


Figura 9.3: Classe para o nível "loops".

Nas figuras 9.4 e 9.5 são vistas as classes **HalfEdge** e **Edge**. A relação entre entre a semi-aresta e aresta é mantida na implementação. Como pode ser visto (figuras 9.4 e 9.5) as semi-arestas possuem acesso à sua respectiva aresta, ao vértice que a compõe e uma referência ao "loop". As arestas são acessadas pela subdivisão planar, como já visto, e possui acesso às semi-arestas, fazendo a conexão entre as faces do modelo. A aresta, através das semi-arestas que a compõem, possui em sua estrutura seus vértices inicial e final.

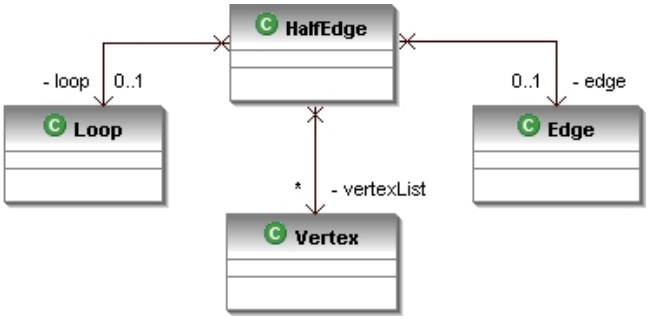


Figura 9.4: Classe para o nível da semi-aresta.

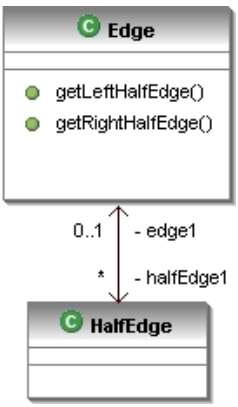


Figura 9.5: Classe para o nível da aresta.

A classe **Vertex** é acessada pelas classes **PlanarSubdivision** e **HalfEdge** (figura 9.6). A principal característica dos vértices é guardar as informações relativas às suas coordenadas.

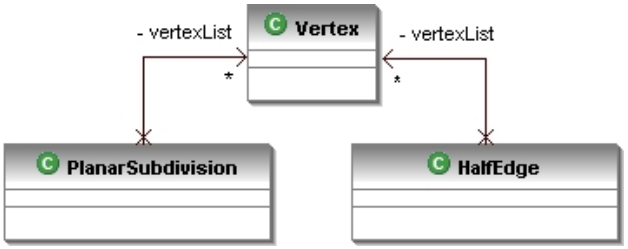


Figura 9.6: Classe para o nível da semi-aresta.

A classe **HalfEdgeModel** representa toda a estrutura de adjacência, como visto na figura 9.7. Para tanto, ela possui acesso à todas as listas encadeadas da estrutura.

Nesta classe são implementados os métodos necessários à obtenção das informações contidas na estrutura de dados.

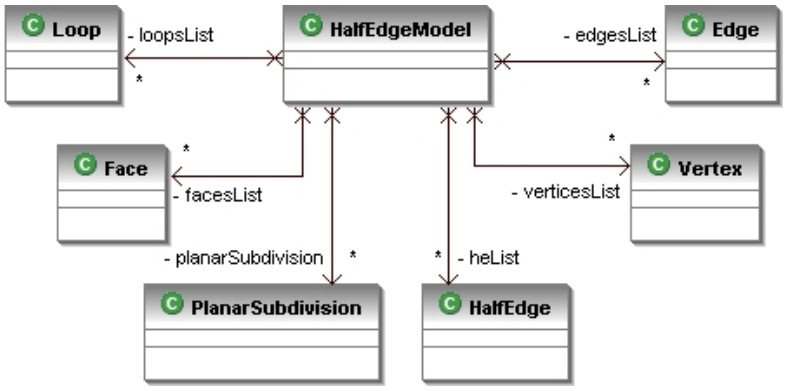


Figura 9.7: Classe **HalfEdgeModel**.

O pós-processador, além das informações geométricas, depende de dados do modelo de elementos finitos como, por exemplo, valores de grandezas nodais. Portanto, os atributos dos nós, elementos e do modelo em geral são também guardados na estrutura de semi-arestas.

Como a aplicação implementada toma como premissa básica a independência entre o modelo de elementos finitos e o pós-processamento, os dados do modelo de elementos finitos são passados à estrutura de adjacência e são alocados em um **HashMap**, vinculando uma chave e um valor a cada posição do mapa.

A classe **HashMap** é a representante básica dos mapas disponibilizados no *framework* de coleções, sendo construída por meio de uma tabela de *hash*, isto é, uma tabela organizada em pares chave-valor (Jandl, 2003).

9.3 Camada Vista do Pós-Processador

A vista do pós-processador é composta pela classe **PostpView** que é derivada da classe **IView**, como mostra a figura 9.8. Esta classe possui uma área de desenho, representada pela classe **DrawingArea**, um estado da vista, representada pela classe **ViewState**, e uma instância do controlador. A vista, ao ser inicializada, também

inicializa o seu respectivo controlador, formando um par vista-controlador. Vários pares vista-controlador podem ser criados.

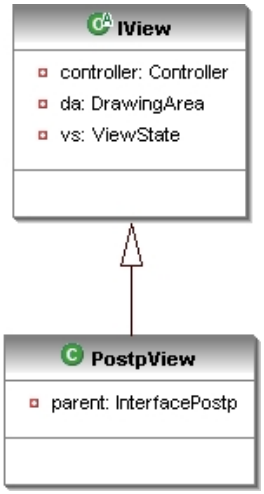


Figura 9.8: Herança da classe **IView**.

A classe **DrawingArea** é responsável pela representação de objetos **Graphics2D** na tela do programa. A classe **Graphics2D** é uma subclasse abstrata da classe **Graphics**. Um objeto **Graphics** gerencia um contexto gráfico e desenha pixels na tela, que representam outros objetos gráficos. Estes objetos contém métodos para desenhar, manipular fontes, manipular cores e outros (Deitel, H. M., Deitel P.J., 2006). Conforme mostra a figura 9.9, a classe **DrawingArea** é derivada da classe **PrintableGridCanvas**, que implementa a interface **Printable** e estende **JComponent**. A classe **JComponent** contém um método *paintComponent()* que pode ser utilizado para desenhar imagens gráficas. Tal método recebe como argumento um objeto **Graphics** e este objeto é usado para redesenhar os componentes gráficos do sistema.

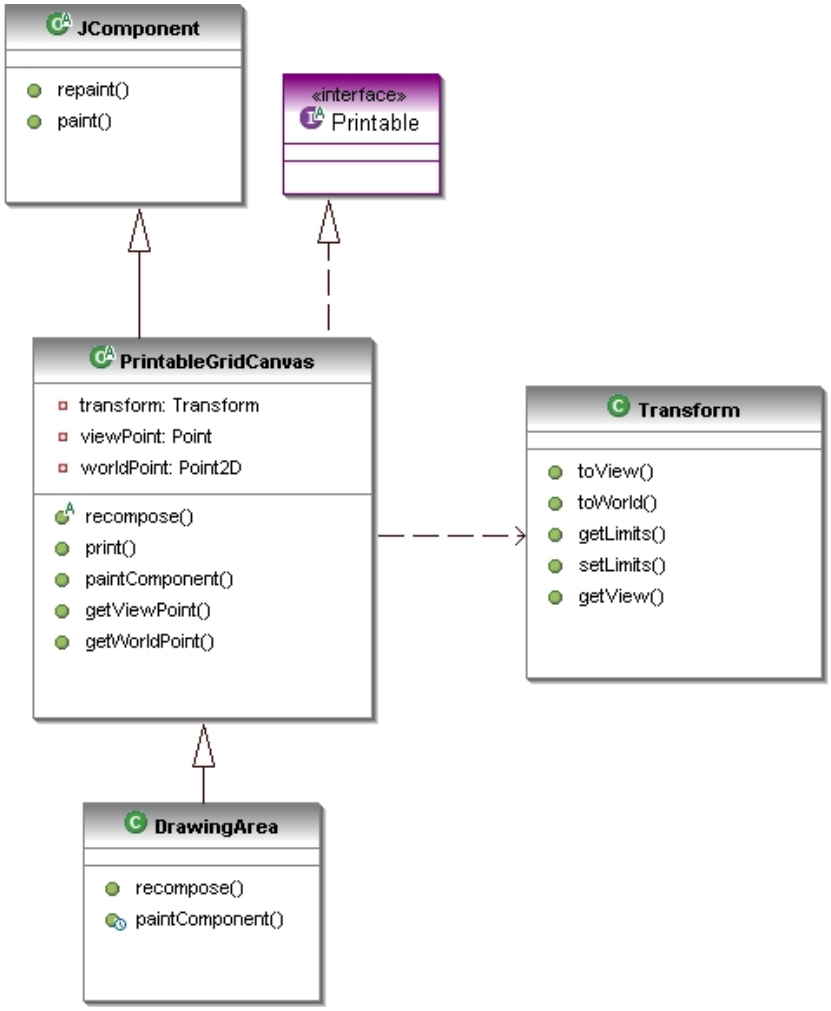


Figura 9.9: Caracterização das classes **DrawingArea** e **PrintableGridCanvas**.

Verifica-se na figura 9.9 uma dependência da classe **PrintableGridCanvas** com a classe **Transform**, que é responsável pelas transformações do sistema de coordenadas do dispositivo gráfico para as coordenadas da vista e ou para as coordenadas do mundo e por estipular os limites do desenho. Destacam-se alguns métodos da classe **PrintableGridCanvas**, como *getViewPoint()*, que retorna um ponto em coordenada da vista; *getWorldPoint()*, que retorna um ponto em coordenada do mundo; *recompose()*, que compõe e redesenha toda a área de desenho.

O estado da vista do pós-processador é representado pela classe **PostpViewState**, que é uma classe derivada de **ViewState**, como mostra a figura 9.10.

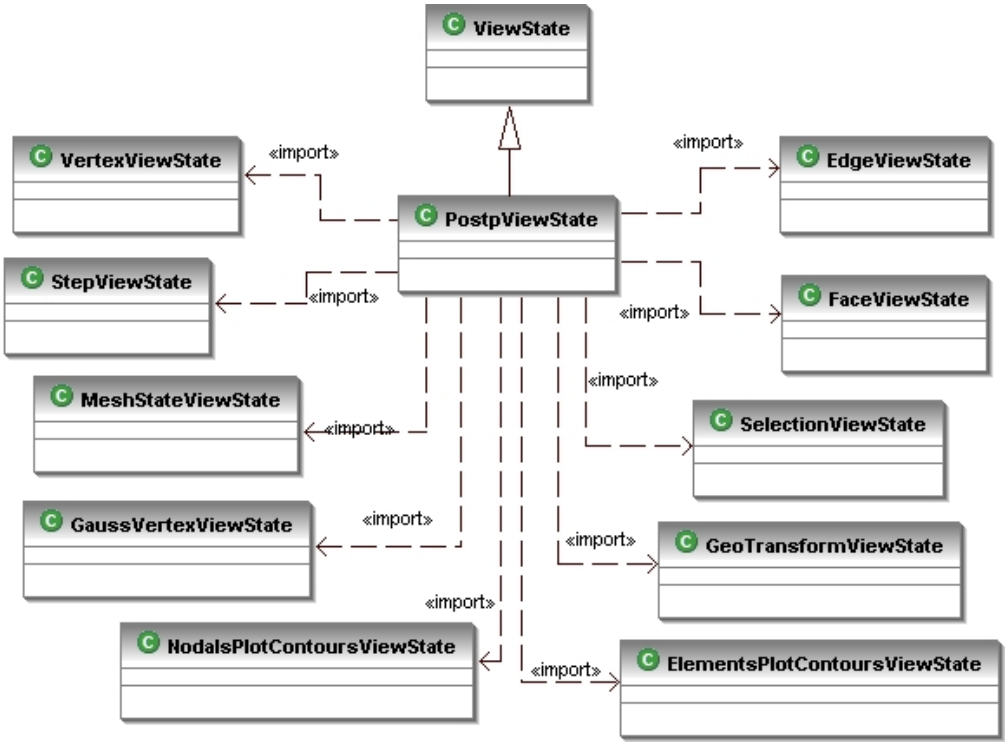


Figura 9.10: As classes componentes do estado da vista.

Na figura 9.10 também estão ilustradas as principais dependências da classe **PostpViewState**. Todas as classes ilustradas são partes componentes do estado da vista.

9.4 Camada Controlador

A classe **PostpController** implementa a interface **Controller** (figura 9.11) e possui referências para os objetos das classes da estrutura de semi-arestas, oriundas do modelo do pós-processador, e instâncias de objetos de desenho, que irão representar o modelo na tela do computador.

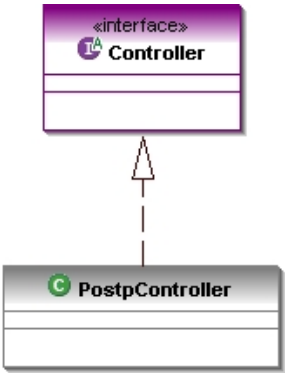


Figura 9.11: Herança da classe **PostpController**.

O controlador tem a função de consultar o modelo e transcrever para a vista todas as informações contidas nele. Desta forma, a classe **PostpController** acessa aa instâncias das classes correspondentes à vista e ao modelo, como ilustra a figura 9.12. Além do acesso ao modelo e à vista, outras classes são necessárias para que o controlador desempenhe todas as funções de desenho e representação.

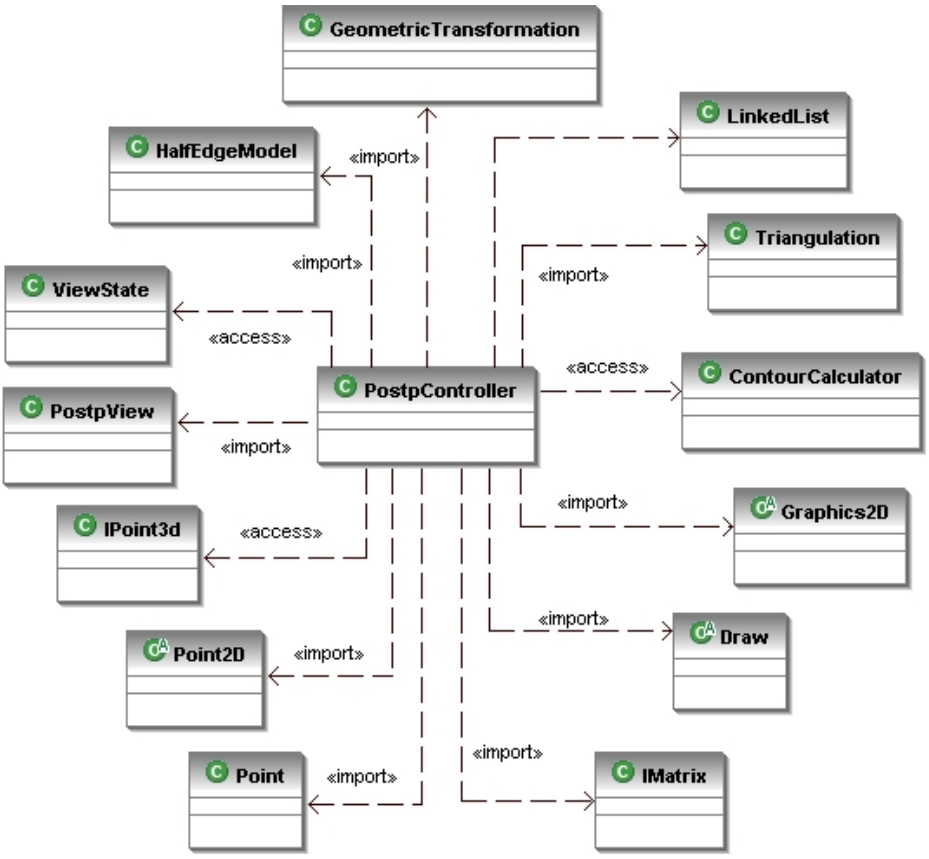


Figura 9.12: Dependências da classe **PostpController**.

O pós-processador trabalha com múltiplos controladores associados a múltiplas vistas, podendo cada par vista-controlador ser atribuído ao mesmo modelo ou a um modelo diferente. A expansão da parte gráfica do sistema pode ser feita de forma simples com a implementação de novos controladores. Novos pares vista-controlador e mais atributos ao estado da vista podem ser incorporados ao aplicativo de forma direta, sem a necessidade de reescrever o código existente.

9.5 Integração entre as Classes

A integração entre as classes das camadas modelo, vista e controlador, como já descrito no capitulo 8, é feita através de uma composição dos padrões de projeto *MVC*, *Observer* e *Command*. A figura 9.13 ilustra esta integração de classes compondo o aplicativo de pós-processamento. São mostrados dois pares vista-controlador, um do pós-processador e um do aplicativo para gráficos no plano, detalhado na seção 9.9.

A possibilidade de exibição de múltiplas áreas de visualização simultâneas facilita a comparação de diferentes resultados e não impede que os resultados sejam visualizados separadamente. As múltiplas janelas se associam cada uma a um par vista-controlador. Como exemplos de outros pares vista-controlador disponibilizados, citam-se:

1. Visualização da malha de elementos finitos;
2. Visualização da triangulação de Delaunay;
3. Visualização do modelo geométrico;
4. Visualização das isofaixas;
5. Visualização de gráficos em geral;

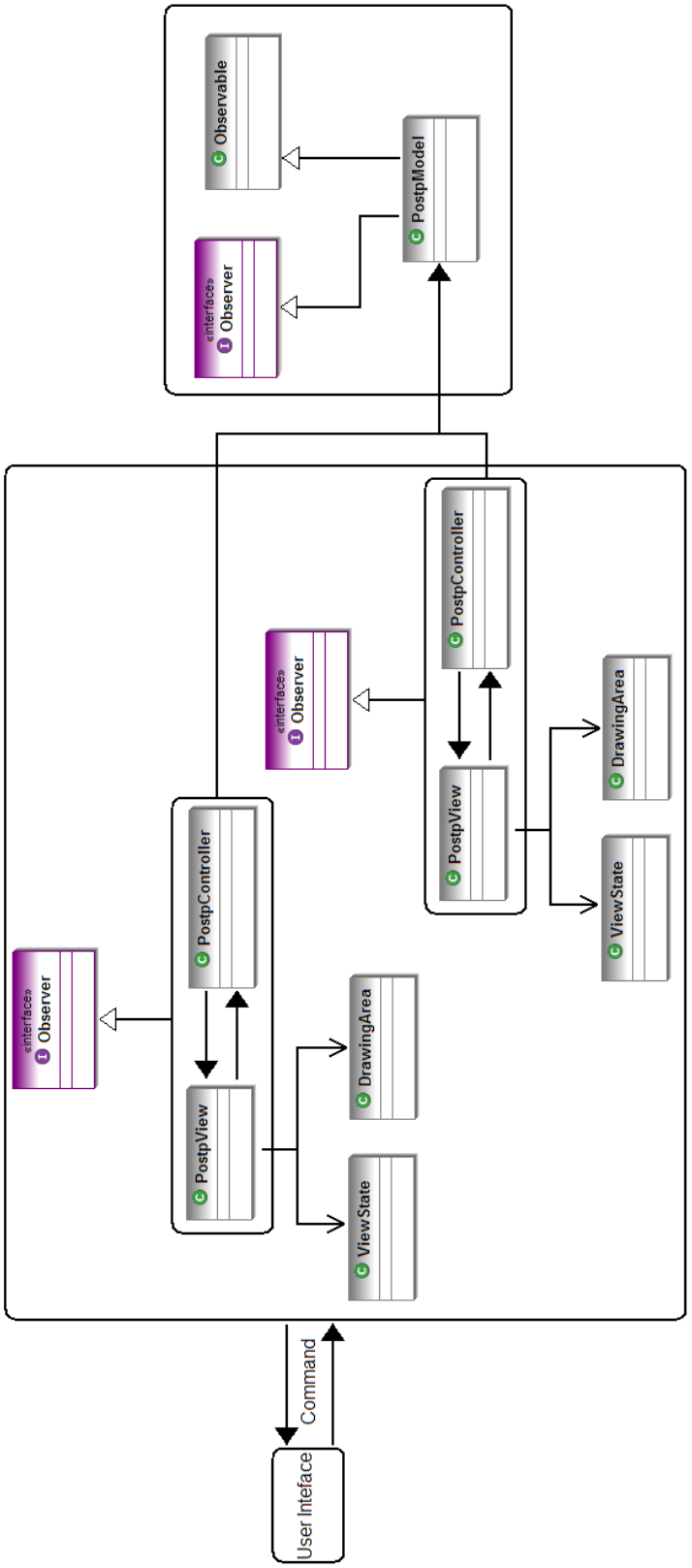


Figura 9.13: Integração entre as camadas modelo, vista e controlador no pós-processador

9.6 Classes para Extrapolação de Grandezas

As classes implementadas para a extrapolação de grandezas podem ser vistas na figura 9.14.

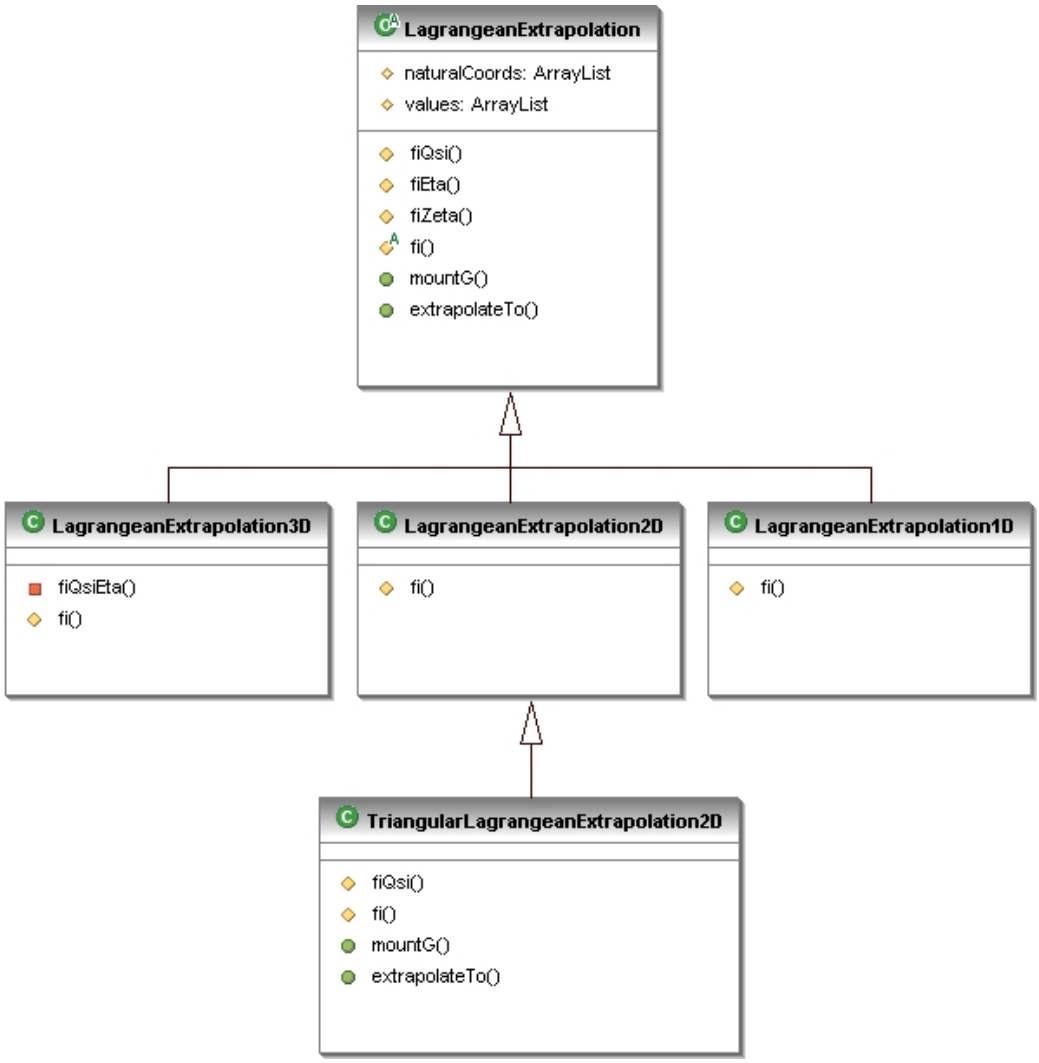


Figura 9.14: Classes para extrapolação de grandezas.

A classe **LagrangeanExtrapolation** possui classes derivadas especializando a extrapolação para os casos unidimensionais, bidimensionais e tridimensionais. Há também uma especialização para domínios triangulares, uma vez que o polinômio para a extrapolação muda e, conseqüentemente, os métodos para obtenção da matriz de extrapolação devem ser sobrescritos.

Na classe **LagrangeanExtrapolation**, os métodos *fiQsi()*, *fiEta()*, *fiZeta()*,

mountG() e *extrapolateTo()* são comuns às demais subclasses. Já o método *fi()* é abstrato em **LagrangeanExtrapolation** e deve ser implementado nas classes derivadas, pois este define o polinômio usado em todo o processo de obtenção da matriz **G**, que é a matriz de extrapolação a ser usada.

9.7 Classes para Geometria Computacional

9.7.1 Triangulações

O projeto orientado a objetos para a triangulação possui uma organização de classes simples que implementam somente a triangulação de Delaunay, embora tenha sido criada uma classe genérica que controla o cálculo da triangulação. A classe genérica é a **Triangulation** que possui apenas dois métodos, o método *compute()*, que invoca o processo de cálculo da triangulação, e o método *trianglesToHalfEdge()*, que se encarrega de organizar a triangulação em uma estrutura de semi-arestas (figura 9.15). A expansão desta classe para outros métodos de triangulação, ou até mesmo para algoritmos diferentes da triangulação de Delaunay, pode ser feita de forma simples pela implementação das novas metodologias de cálculo em classes derivadas da classe **Triangulation**.

A classe **Delaunay**, vista na figura 9.15, faz uso de outras classes representativas de uma estrutura de dados interna da triangulação. Na figura 9.16 tem-se a associação da classe **Delaunay** com as classes **TrianglesD**, **EdgeD** e **NodesD** que são, respectivamente, classes representando um triângulo de Delaunay, uma aresta de um triângulo de Delaunay e um vértice da triangulação.

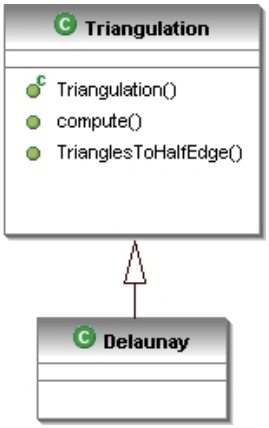


Figura 9.15: Classe **Triangulation**.

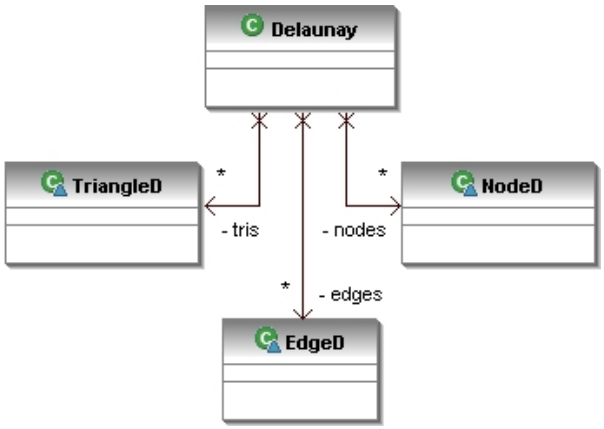


Figura 9.16: Classe **Delaunay**.

9.7.2 Transformações Geométricas

As operações geométricas descritas no capítulo 4 sugerem uma divisão baseada em duas classes bem distintas, uma relativa às transformações geométricas e uma relativa às projeções. A figura 9.17 mostra as subclasses da classe **GeoTransform**. A classe **Projection** representa as projeções e a classe **Transformation** representa as transformações.

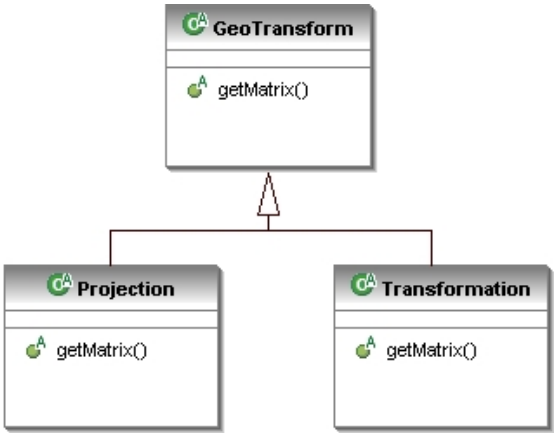


Figura 9.17: Herança da classe **GeoTransform**.

A classe **GeoTransform** possui o método abstrato *getMatrix()*, que é implementado nas classes que definem o tipo de operação. O método tem como finalidade retornar a matriz de transformação desejada. Assim, este método é implementado pelas classes **Projection** e **Transformation**, ambas derivadas de **GeoTransform** (ver figura 9.17). Na figura 9.18 são mostradas todas as classes derivadas de **Projection** que foram implementadas. Na figura 9.19 é ilustrada a herança das classes de transformações geométricas.

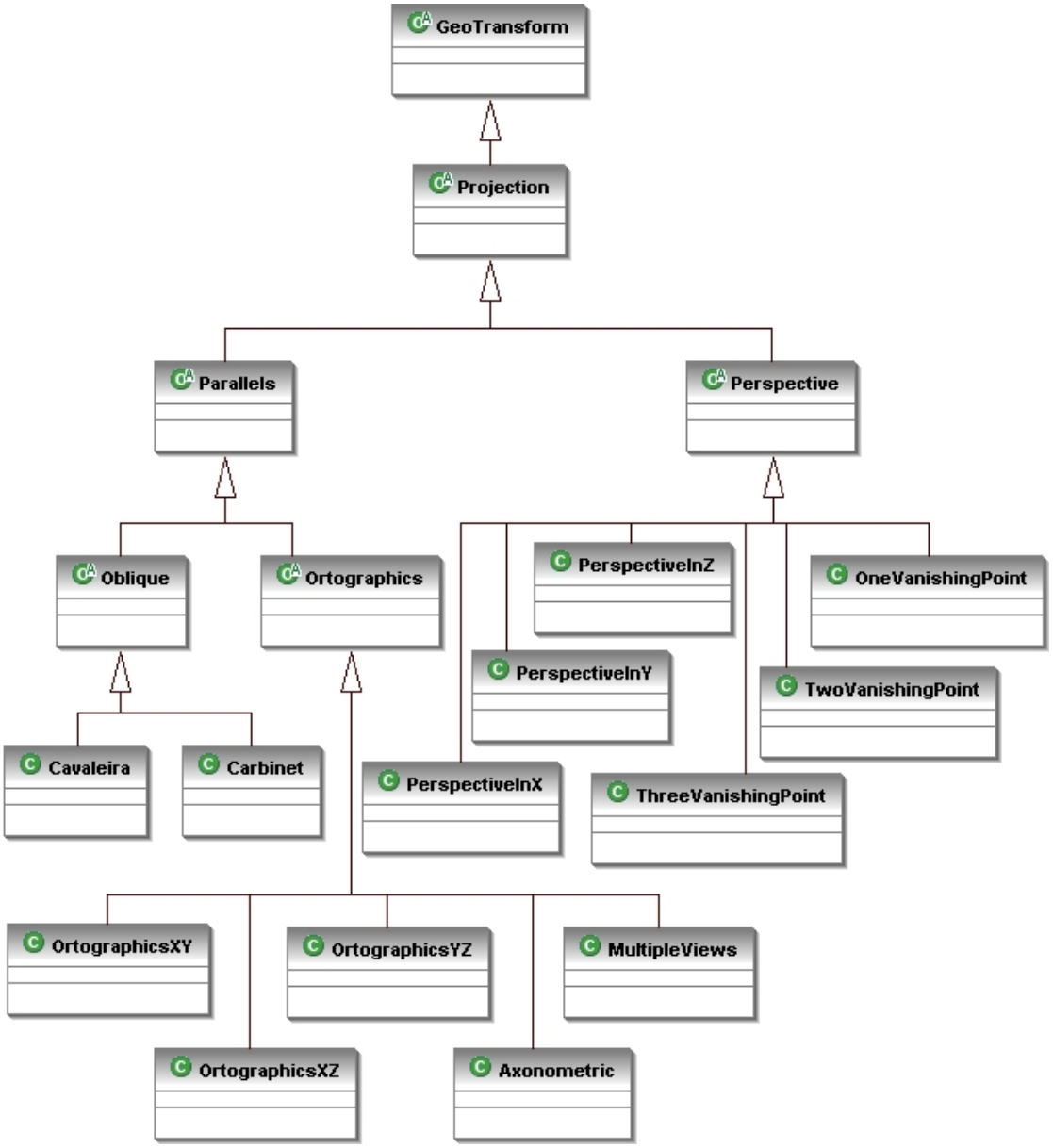


Figura 9.18: Estrutura das classes herdeiras de **Projection**.

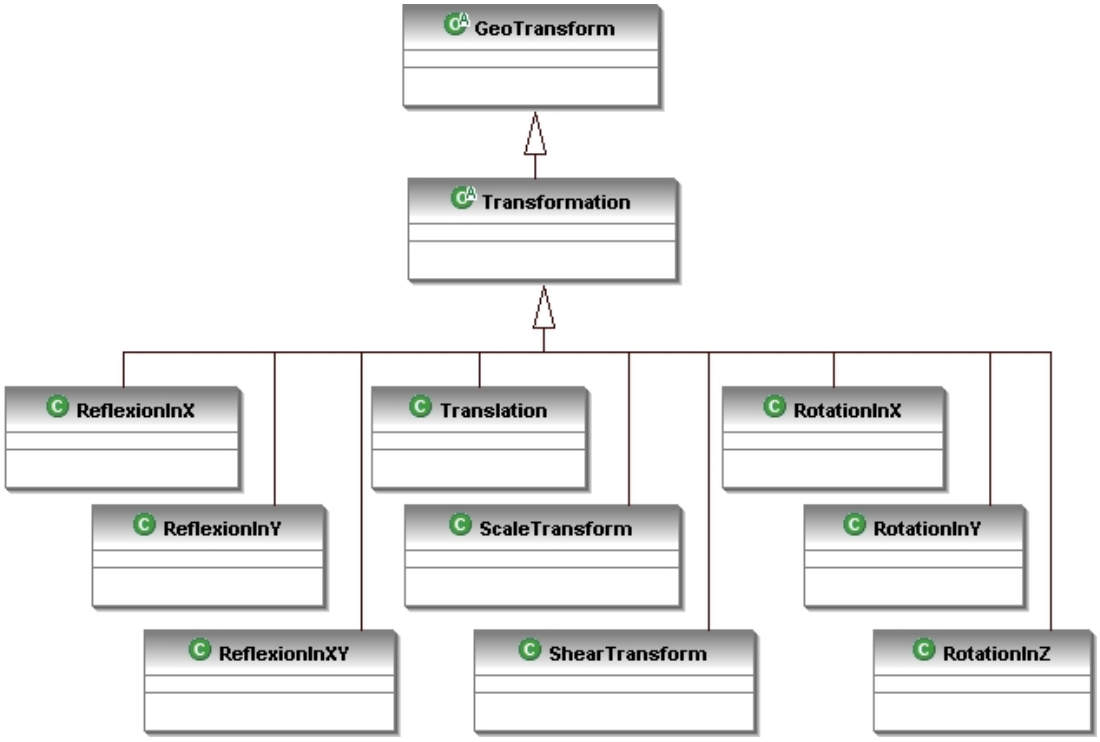


Figura 9.19: Estrutura das classes herdeiras de **Transformation**.

9.8 Classe para o Cálculo das Isofaixas

O cálculo das isofaixas depende apenas dos polígonos oriundos da subdivisão de domínios ou dos próprios elementos do modelo do MEF, dos limites (máximo e mínimo) da grandeza a ser representada e da escala de cores. Partindo destes dados, a classe **ContourCalculator** (figura 9.20) se encarrega de calcular as faixas de valores e, por interpolação, cria os contornos para cada subdomínio. Estes contornos são representados pela classe **Contour** e são guardados em uma lista, para desenho posterior.

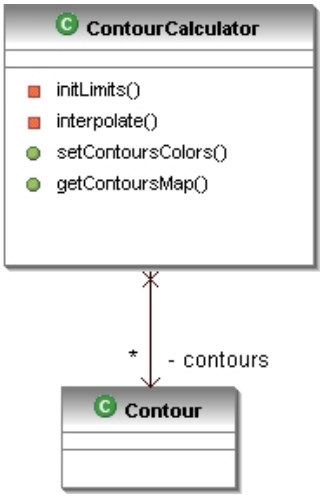


Figura 9.20: Classes para o cálculo das isoFaixas de valores.

Na classe **ContourCalculator** são destacados alguns métodos. O método *initLimits()* calcula os valores de cada faixa da representação. Para tanto, o método se baseia na quantidade de cores da representação e nos valores máximo e mínimo da grandeza. A partir daí são obtidos intervalos de valores para cada cor. O método *interpolate()* avalia os valores vinculados aos vértices do contorno do subdomínio e, através de uma interpolação, obtém os pontos do contorno. O método *setContoursColors()* atribui uma paleta de cores, na qual o processo de cálculo se baseia. Após todo o cálculo, o método *getContourMap()* pode ser evocado para desenho das isofaixas.

9.9 Aplicativo para Visualização de Gráficos

A organização do aplicativo gráfico do pós-processador segue a mesma do sistema, segundo o padrão MVC.

O modelo gráfico usado foi o de dispersão de pontos, composto por seqüências de valores e eixos de coordenadas. O modelo, portanto, é composto por três classes: **ISequence**, **Axi**s e **XYPlotModel**. O diagrama de associação das classes do modelo é representado na figura 9.21.

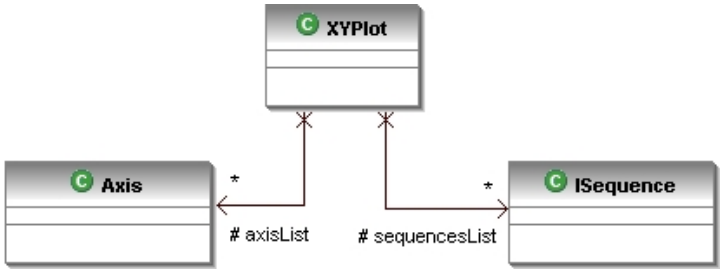


Figura 9.21: Classes do modelo.

A classe **ISequence** é usada para caracterizar uma seqüência do gráfico, sendo composta basicamente por uma lista de pontos e métodos que operam sobre os valores da seqüência. A classe **Axis** trata da caracterização de um eixo correspondente a um gráfico, com variáveis para modificar os limites do eixo, a separação de marcas principais e secundárias e a escala do eixo. A classe **XYPlotModel** é responsável pela composição do modelo, recebendo as listas de seqüências e os eixos do gráfico. Os métodos implementados nesta classe permitem manipular os valores do gráfico, alterando as listas de seqüências e eixos.

A vista do aplicativo é composta pelas classes **IView** e **ViewState**. A classe **IView** é uma classe abstrata e possui referências a uma área de desenho, a um controlador, e ao estado da vista. De forma geral, a classe **IView** possibilita a implementação de outras vistas para o modelo, ou até mesmo a implementação de outras aplicações visuais. A classe **ViewState** possui parâmetros gerais para a configuração do estado da vista, e as classes derivadas de **ViewState** garantem que os componentes da vista do gráfico tenham seus respectivos estados passíveis a mudanças.

Conforme descrito, e visto na figura 9.22, a vista é composta pela classe **XYPlotView**, derivada de **IView**, que possui uma área de desenho, uma referência ao estado da vista e outra ao controlador. Analogamente, **XYPlotViewState** é uma classe derivada de **ViewState** (figura 9.23) e possui referências de outras classes que configuram o estado da vista, como mostrado na figura 9.24.

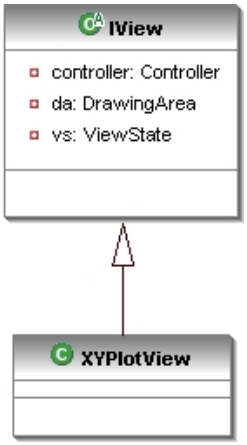


Figura 9.22: Hierarquia da classe **IView**.

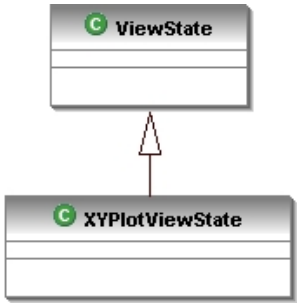


Figura 9.23: Hierarquia da classe **ViewState**.

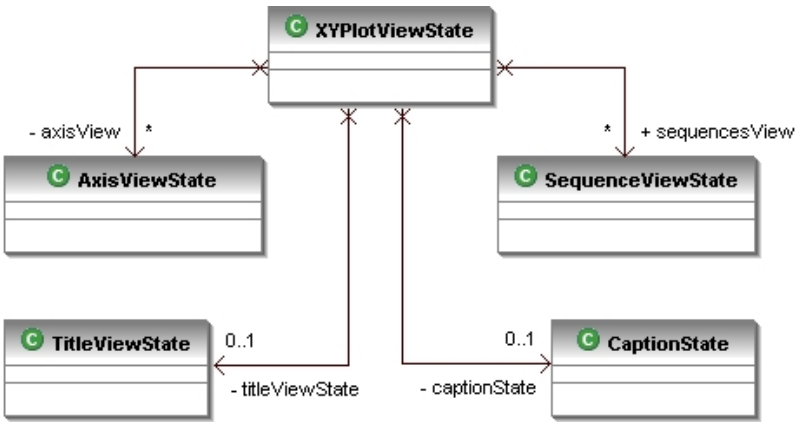


Figura 9.24: Associação de instâncias da classe **XYPlotViewState**.

O controlador compõe a terceira camada da arquitetura e cabe a ele realizar as interações entre o modelo e vista. A classe **XYPlotController**, implementa a interface **Controller** (figura 9.25) e é o controlador do aplicativo gráfico. No

caso de outros modelos ou outras vistas, devem ser implementados outros controladores independentes do já existente. O controlador possui um objeto da classe **XYPlotModel**, representativo do modelo, e uma referência à classe **IView** que é especializada pela classe derivada **XYPlotView**. O acesso do controlador à vista e ao modelo é feito por estas referências, como visto na figura 9.26.

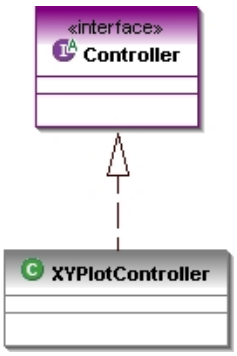


Figura 9.25: Hierarquia do controlador.

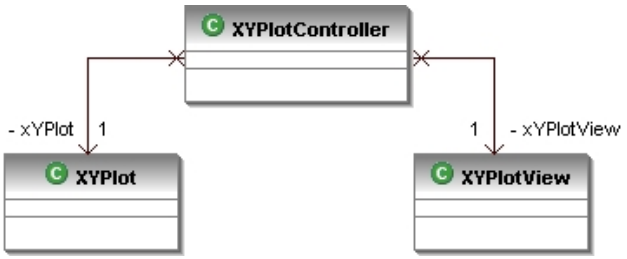


Figura 9.26: Associação da classe **Controller** com as camadas da vista e do modelo.

O relacionamento entre as classes no momento da execução do programa se dá segundo o padrão *Command*. Um comando, ao ser executado, aciona o controlador da respectiva ação que, por sua vez, está habilitado a acessar o modelo, para a manipulação de informações, e a vista, modificando a forma de visualização.

9.10 Pós-Processamento Sincronizado com o Processamento

O uso de múltiplos *Threads* associados a diversos pares vista-controladores e principalmente ao padrão *Observer* propiciou o pós-processamento de modelos de elementos finitos em tempo de processamento. O processamento sincronizado permite ao usuário acompanhar o desenvolvimento do pós-processamento ao longo de uma análise não-linear e interagir com as grandezas representadas pelo programa. Este sincronismo só foi possível graças à combinação de padrões de projeto mostrada figura 9.27.

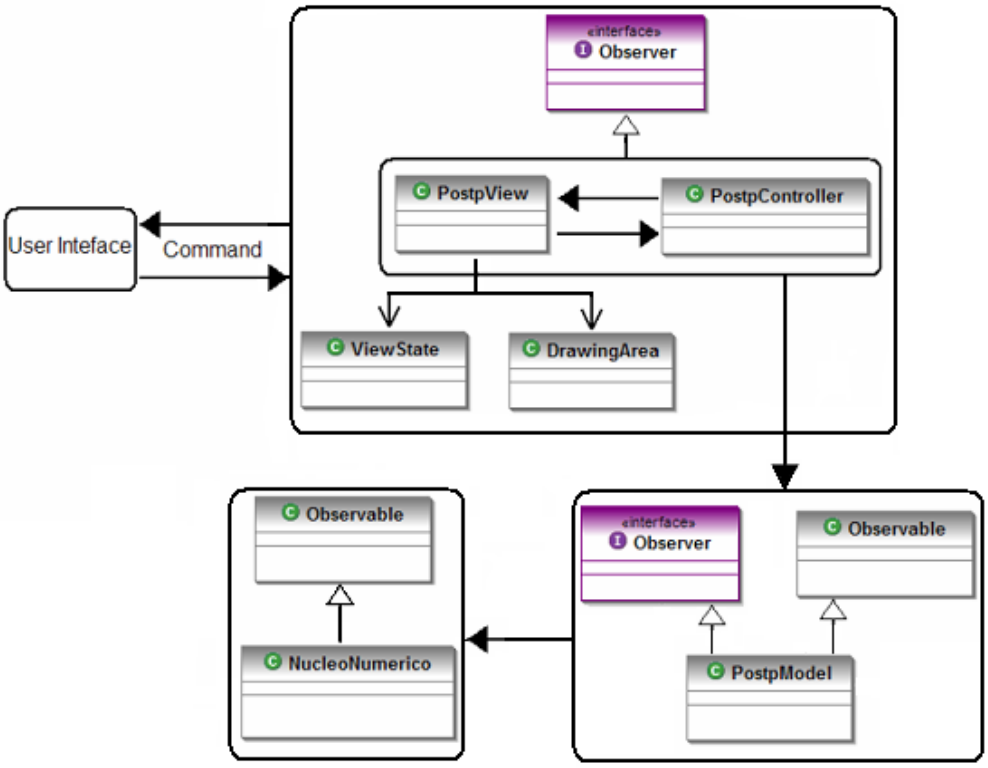


Figura 9.27: Organização do pós-processamento sincronizado.

9.11 Generalização do Pós-Processador

No intuito de generalizar o pós-processador, habilitando-o a ser usado para qualquer modelo discreto, foram implementadas classes para descrever os dados dos modelos na forma de entidades geométricas simples. A figura 9.28 mostra a associação da classe **GeoPostpModel** com as classes **PointModel**, **Patch** e **Boundary**.

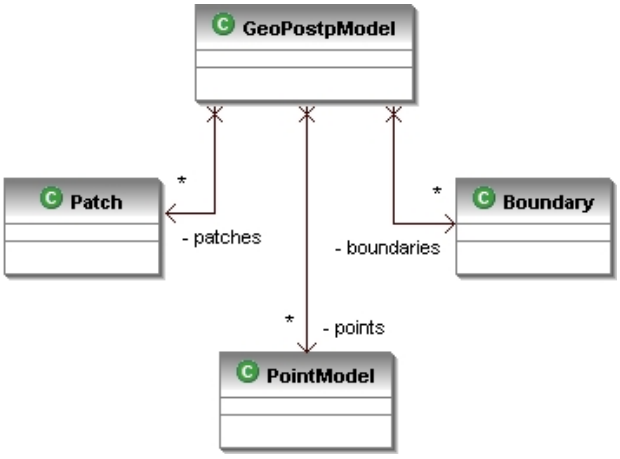


Figura 9.28: Classe **GeoPostpModel**.

A classe **PointModel** representa um ponto geométrico genérico, que além de suas coordenadas, recebe também informações que são guardadas em um mapa. A classe **Patch** recebe informações dos **PointModels** que a compõe. Em outras palavras, seria a incidência do respectivo "*patch*". A classe **Boundary** recebe informações, relativas aos **PointModels**, que correspondem ao fecho convexo de todos os pontos descritos no modelo.

Um modelo passa a ser descrito através de objetos **GeoPostpModel**. Isto é possível através de uma camada criada com a finalidade de interpretar tal modelo e preencher instâncias da classe **GeoPostpModel** e, na seqüência, transcrevê-las em estrutura de semi-arestas. Esta camada é chamada de *Parser* e, cada modelo implementado no sistema INSANE necessita de uma implementação da mesma.

Para a análise não-linear, a descrição de um passo, como um conjunto de objetos **GeoPostpModel**, é feita através da classe **StepPostp** e o conjunto de passos é

representado pela classe **PostpModel**, conforme ilustra a figura 9.29.

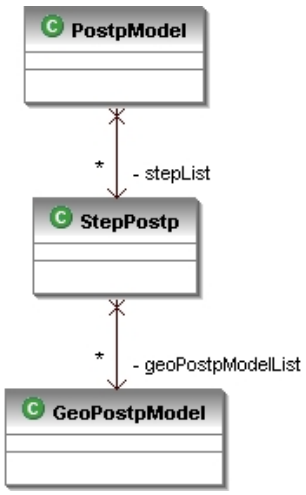


Figura 9.29: Associação das classes do **PostpModel**.

Para a criação de uma nova camada *Parser*, deve-se implementar a interface **ParserPostpModel**, como ilustrado na figura 9.30. Para o modelo de elementos finitos do INSANE (representado pela classe **FemModel** e detalhado no apêndice A), implementou-se a classe **FemModelToPostpModel**, conforme mostra a figura 9.30. A classe utiliza **NodalFemToGeoPostpModel** e **GaussFemToGeoPostpModel**, uma para as grandezas nodais e outra para as grandezas nos pontos de integração (ver figura 9.30).

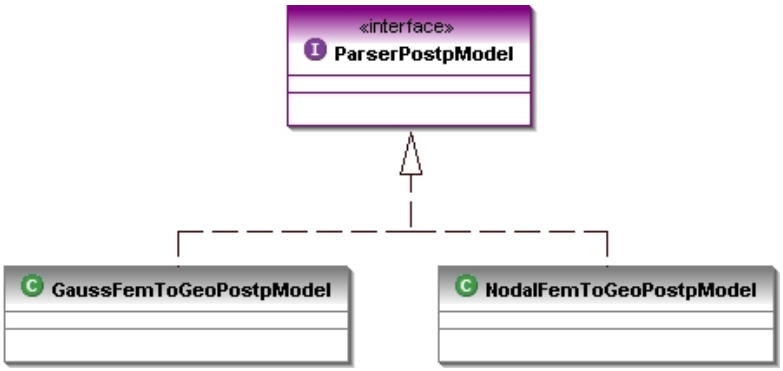


Figura 9.30: Interface **ParserPostpModel**.

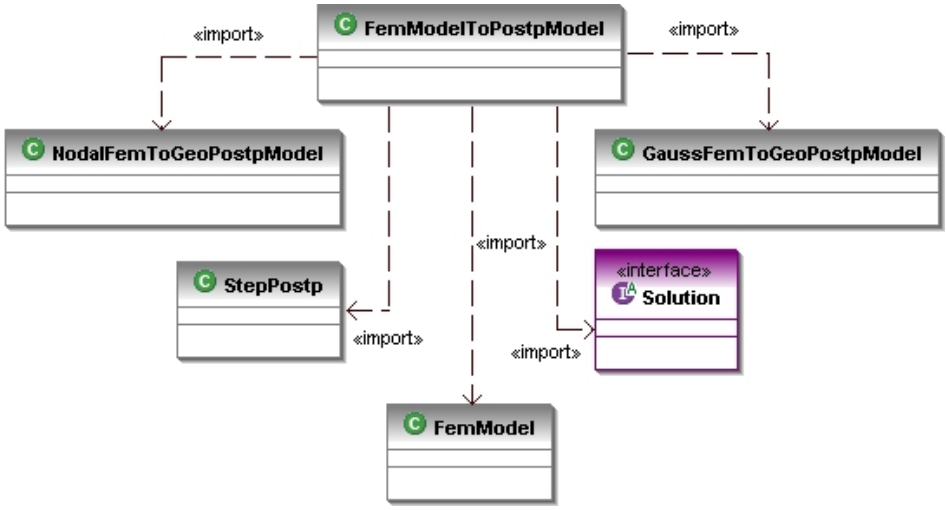


Figura 9.31: Classe **FemModelToPostpModel**

Após o preenchimento do **PostpModel**, é usada a classe **ParserHalfEdge**, que interpreta os dados do modelo genérico e o organiza em um modelo de semi-arestas, que é passado para o pós-processador. A figura 9.32 mostra as dependências de **ParserHalfEdge**.

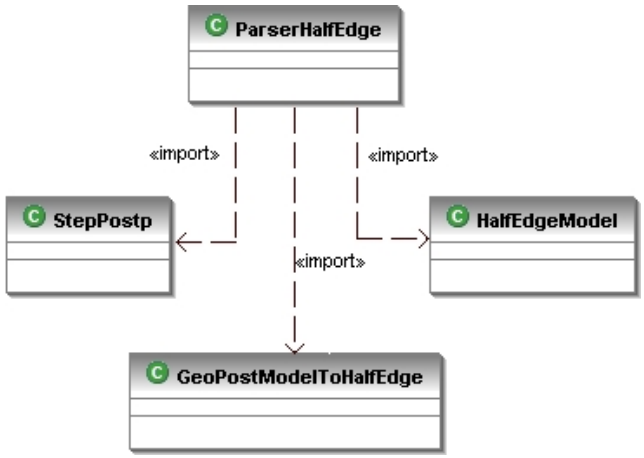


Figura 9.32: Dependências do **ParserHalfEdge**.

9.12 Persistência dos Dados do Pós-Processador

O pós-processador se comunica com os demais módulos do programa (pré-processador e processador) através da camada da persistência. As classes que implementam a

interface **Persistence** têm métodos capazes de interpretar os dados do modelo e exportá-los para arquivos XML (como as classes **PersistenceAsXML** e **PostpPersistenceAsXML**) ou para arquivos binários contendo o objeto JAVA correspondente (classe **PersistenceAsInsane**). As implementações da interface **Persistence** estão mostradas na figura 9.33.

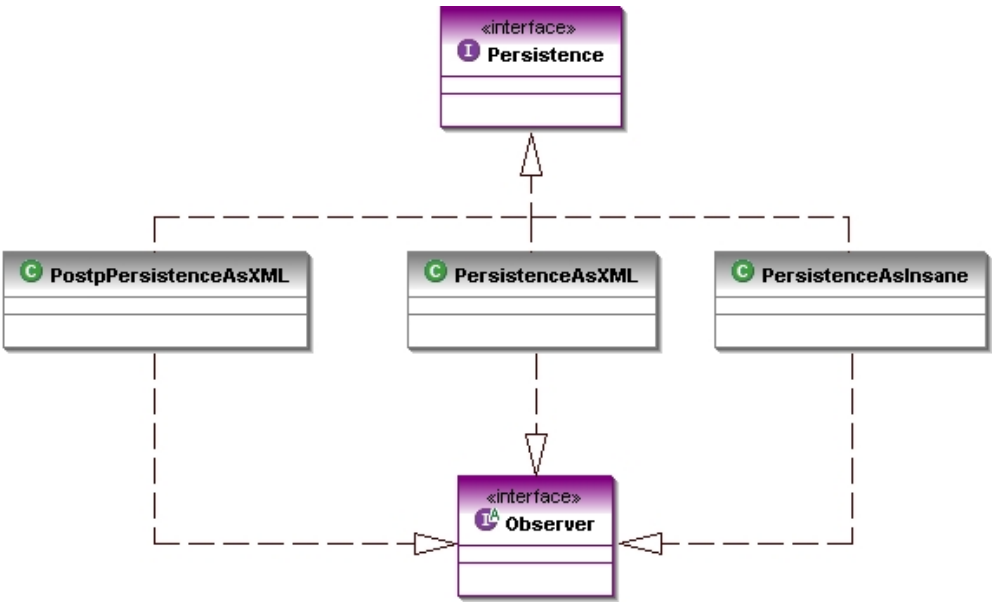


Figura 9.33: Implementação da interface **Persistence**.

A classe **PostpPersistenceAsXML** trabalha com dados do modelo do pós-processador, persistindo-os segundo as definições de marcação detalhadas no apêndice B.

Capítulo 10

Recursos Disponibilizados e Funcionamento do Programa

10.1 Introdução

Neste capítulo são apresentados os recursos disponíveis no pós-processador enfatizando o funcionamento do programa e suas principais características, tais como a visualização de resultados por isofaixas, pós-processamento sincronizado e a visualização de resultados em múltiplas vistas.

Adota-se um modelo como base para a descrição das características do programa. No capítulo 11 outros exemplos para vários tipos de modelos são mostrados, de forma que os recursos de visualização do programa, tratados neste capítulo, sejam ressaltados. O modelo aqui adotado é o de uma viga bi-apoiada com uma carga concentrada no meio do vão, mostrado na figura 10.1.

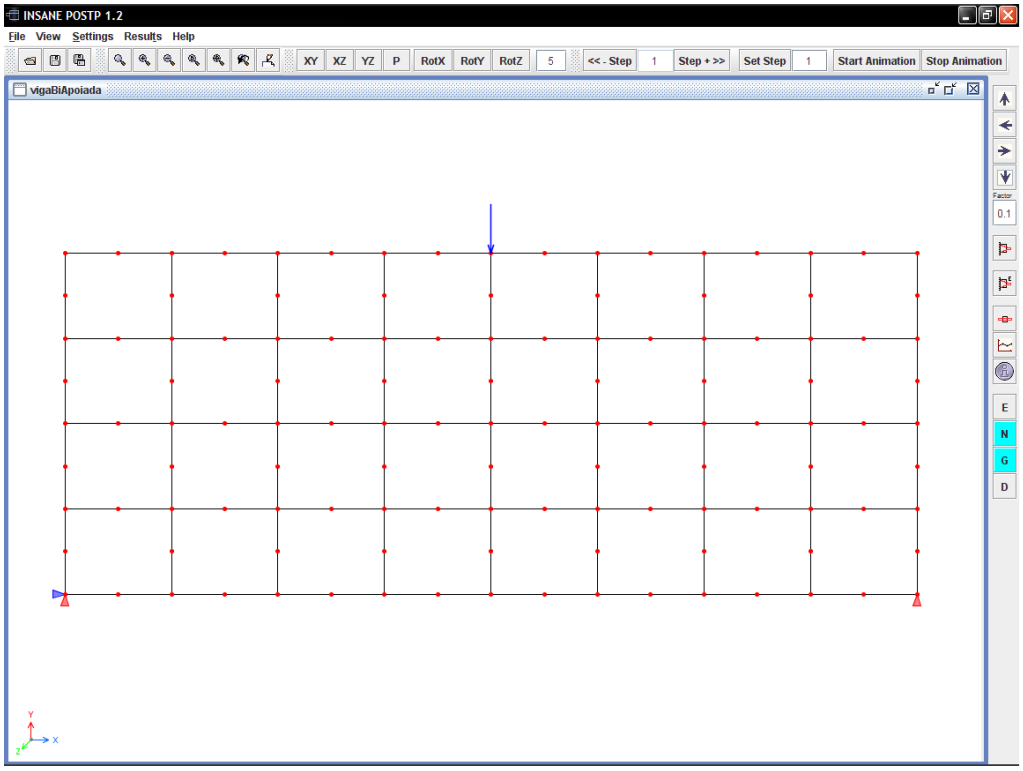


Figura 10.1: Viga bi-apoiada.

10.1.1 Inicialização do Pós-Processamento

O pós-processador ao ser inicializado exibe um **Desktop** vazio com os menus **File** e **Help** ativados. Neste ponto, o programa está preparado para carregar qualquer modelo compatível. A partir do menu **File** podem ser abertos arquivos do pós-processador (extensão .ipp) e arquivos de um **FemModel**(extensão .isn), que são arquivos do modelo de elementos finitos persistidos como objetos JAVA durante o processamento. Arquivos XML nos formatos **PostpModel** (com dados do modelo do pós-processador) e **FemModel** podem ser "importados" para serem lidos pelo programa. É possível exportar imagens da área de desenho nos formatos JPEG, PNG ou POSTSCRIPT. Os dados do modelo de pós-processador também podem ser exportados para um arquivo XML. Para os dados de uma análise não-linear tem-se a possibilidade de carregar dados de arquivos XML de um **FemModel** ou **PostpModel** e arquivos de objetos originais do **FemModel**. Os passos de uma

análise podem ser todos carregados na memória do computador ou podem ser usados a partir do disco.

10.2 Visualização da Malha e dos Atributos do Modelo

A primeira visualização de um modelo, ao ser carregado, é sua malha, de forma que suas características geométricas sejam bem ilustradas. A partir da visualização da malha, suas características, e as definições de cores e fontes podem ser configuradas por meio de diálogos com o usuário.

A visualização dos atributos da malha é acionada pelo diálogo **Display**, no menu **View**. Neste diálogo a exibição, por exemplo, dos nós, dos elementos da malha, dos pontos de integração podem ser ativadas. Este diálogo está mostrado na figura 10.2.

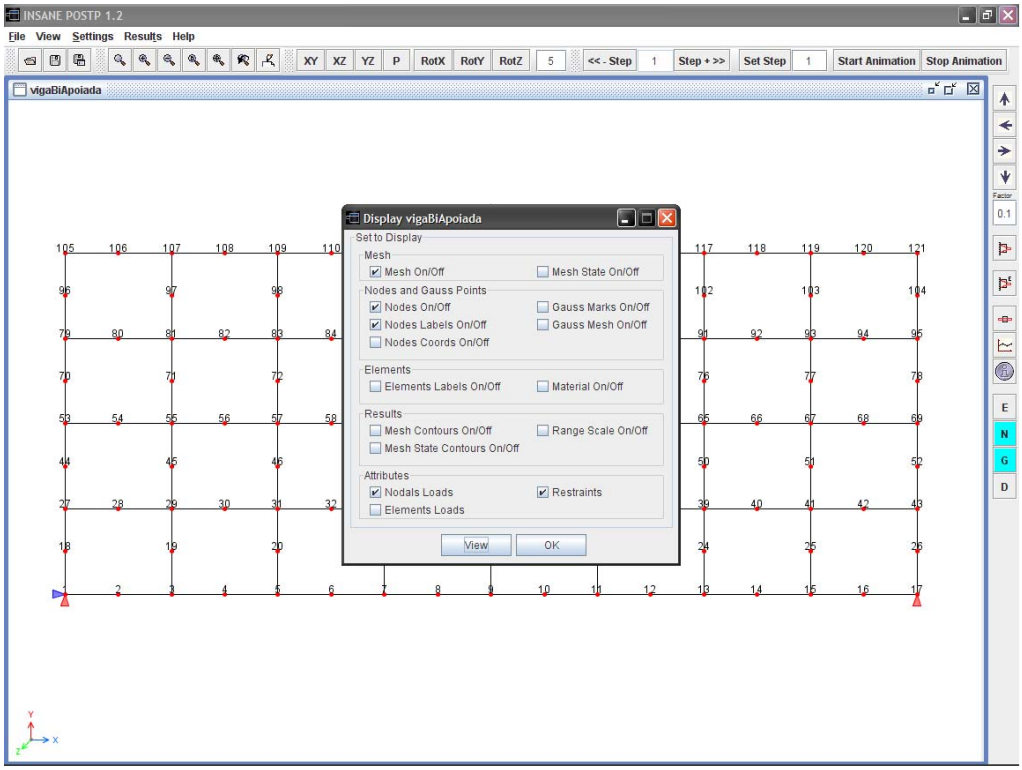


Figura 10.2: Visualização de atributos.

10.3 Gerenciamento de Modelos e Múltiplas Vistas

No menu **View** tem-se a possibilidade de seleção e visualização de modelos diversos através da opção **Model Selection**. O diálogo correspondente apresenta uma lista de modelos que podem ser selecionados e exibidos. Estes modelos ficam armazenados na memória do computador e o diálogo funciona como um gerenciador de dados. Os comandos vinculados ao gerenciamento dos modelos permitem exibir um novo modelo ou uma nova vista. Há também comandos para apagar da memória um modelo e fechar as vistas do **Desktop** principal. A figura 10.3 apresenta o diálogo exibindo vários modelos guardados.

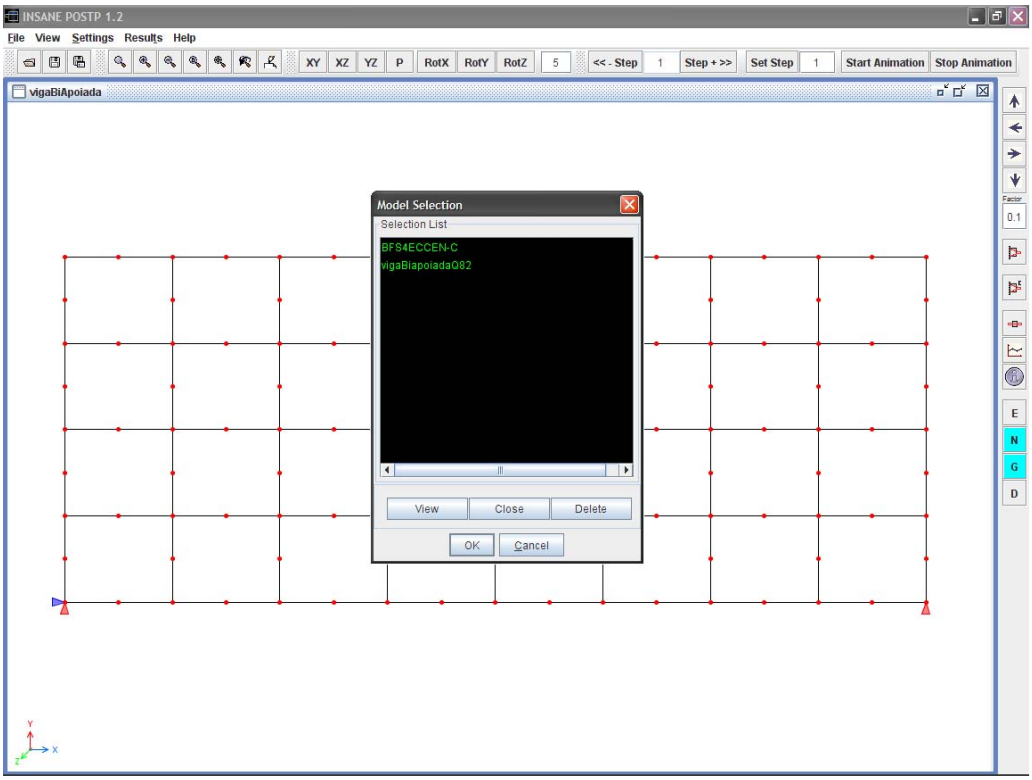


Figura 10.3: Gerenciamento de modelos e vistas.

Trabalhar com múltiplas vistas pode ser importante, pois permite observar de diversos ângulos uma determinada malha, ou a malha de um mesmo modelo com

atributos e informações diferentes em cada vista. A figura 10.4 apresenta um mesmo modelo com quatro vistas diferentes, com informações diversas em cada uma delas. Neste exemplo, visualiza-se ao mesmo tempo a malha, a numeração dos nós e elementos e isofaixa de valores do modelo, facilitando a análise de resultados. Modelos distintos, como ilustrado na figura 10.5, também podem ser exibidos ao mesmo tempo.

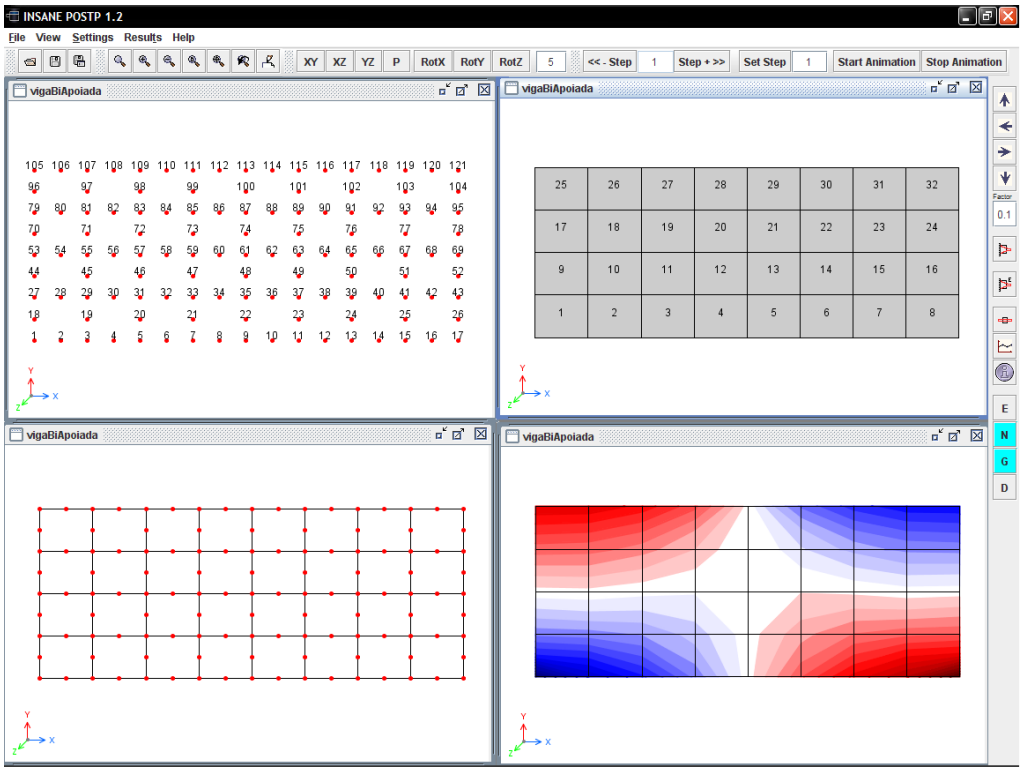


Figura 10.4: Exibição de diferentes vistas simultâneamente.

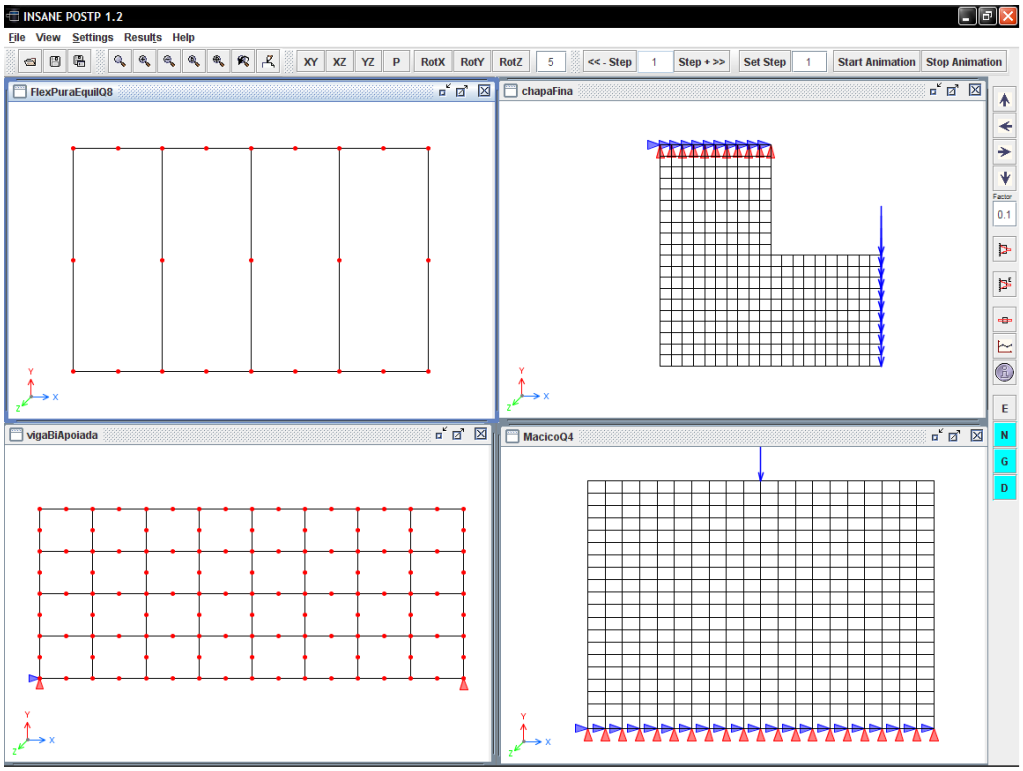


Figura 10.5: Múltiplos modelos.

10.4 Transformações Geométricas

As transformações geométricas implementadas podem modificar a visualização através dos comandos postos na barra de ferramentas da interface gráfica. Existem comandos para projeções nos planos XY, XZ e YZ, visualização em Perspectiva e Rotações em torno dos eixos X, Y e Z. O ângulo para uma rotação pode ser configurado no campo “Angle Step”. A figura 10.6 ilustra múltiplas vistas de um modelo. São mostradas projeções nos planos XY, XZ, YZ e uma perspectiva.

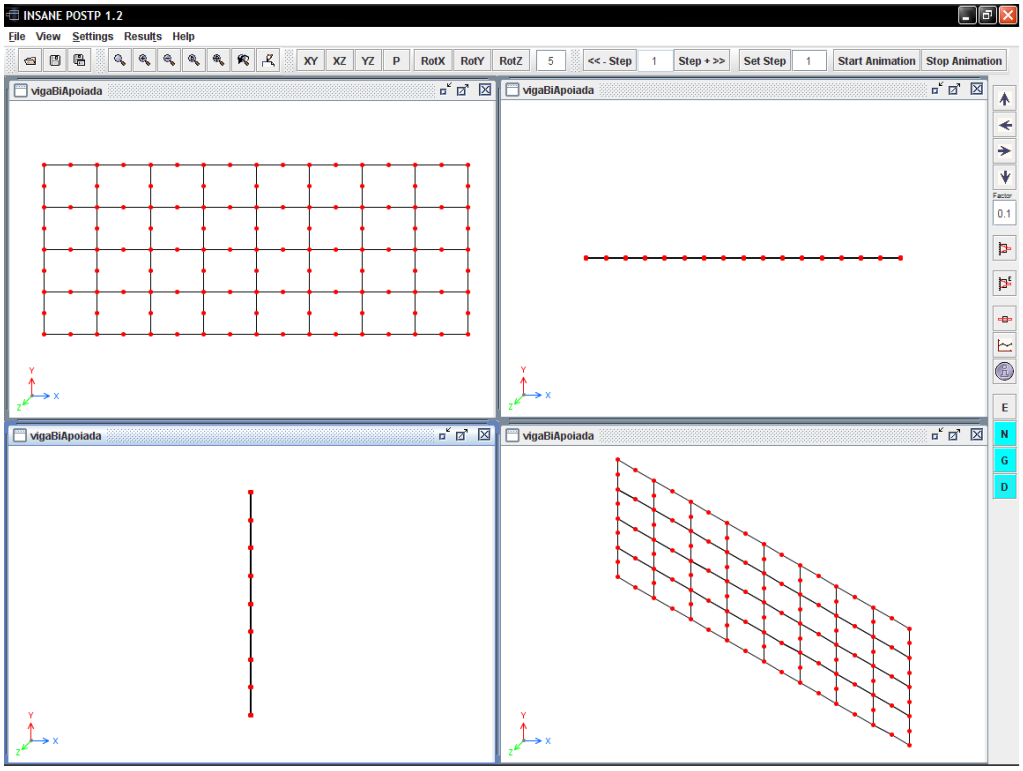


Figura 10.6: Visualização do modelo e suas projeções em múltiplas vistas.

Além das operações de projeções e rotações é possível visualizar o modelo transladado. No menu **View**, o dialogo **Move** apresenta comandos que permitem configurar e usar as translações. Como na rotação o passo da translação é estipulado pelo usuário também é possível aplicar uma escala para uma melhor visualização do modelo. O diálogo correspondente pode ser visto na figura 10.7

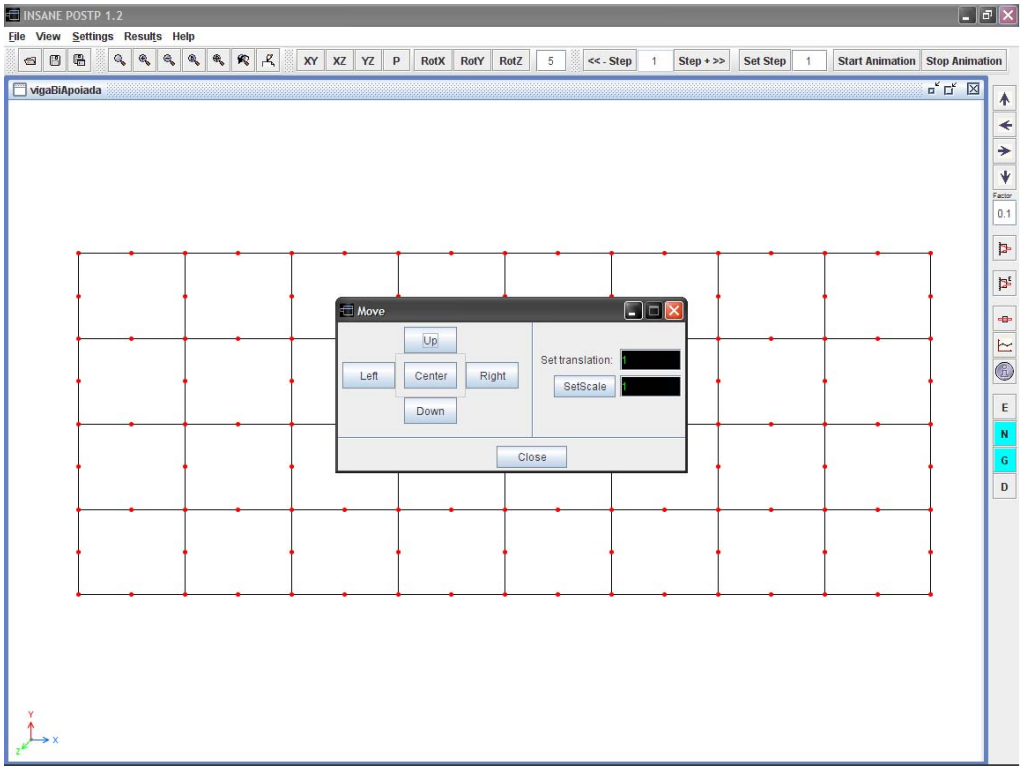


Figura 10.7: Translação e escala do modelo.

10.5 Visualização da Configuração Deformada

Para a visualização da configuração deformada da malha, no menu **View Results**, tem-se a chamada para o diálogo **Mesh State**. Para a exibição da malha deformada é necessário selecionar, de uma lista, as grandezas originadas no processamento relativas ao estado deformado da malha, como mostrada a figura 10.8. O resultado da malha deformada é visto na figura 10.9. A figura 10.10 mostra diversas vistas da malha deformada.

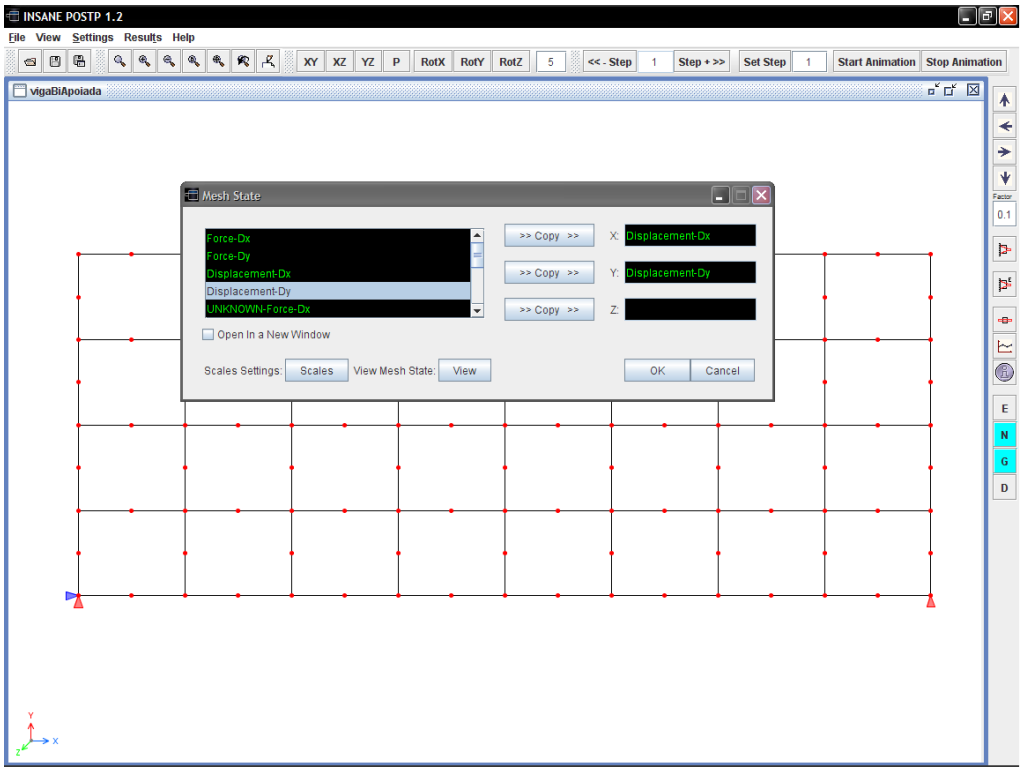


Figura 10.8: Seleção das grandezas do estado deformado da malha.

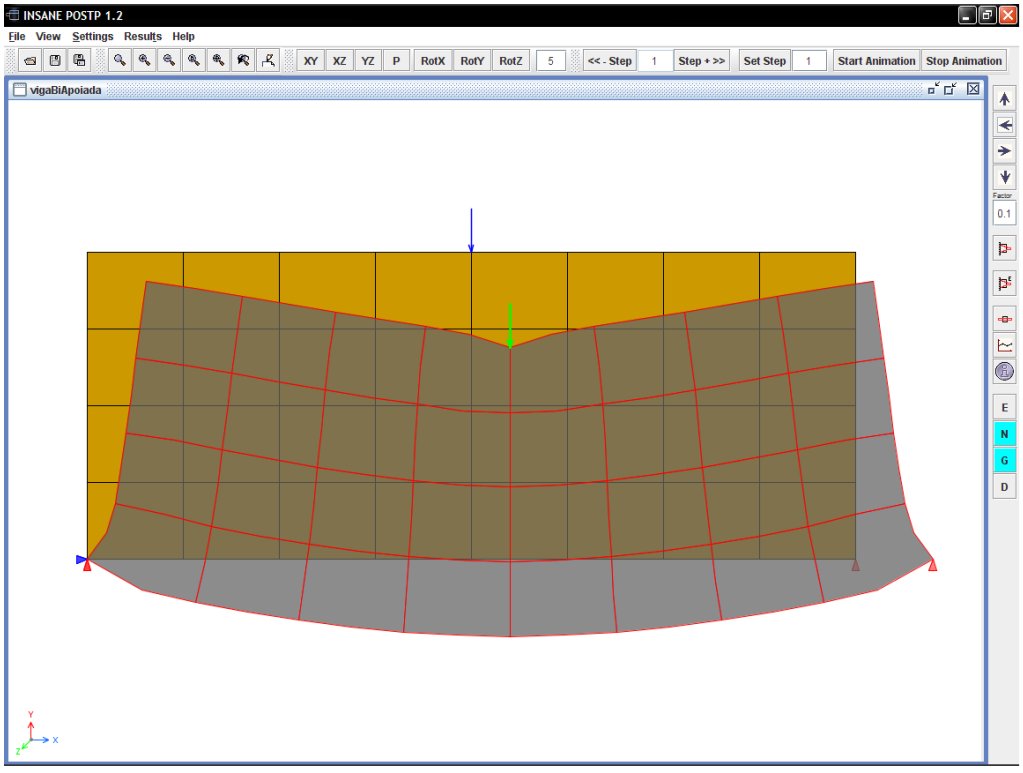


Figura 10.9: Malha deformada.

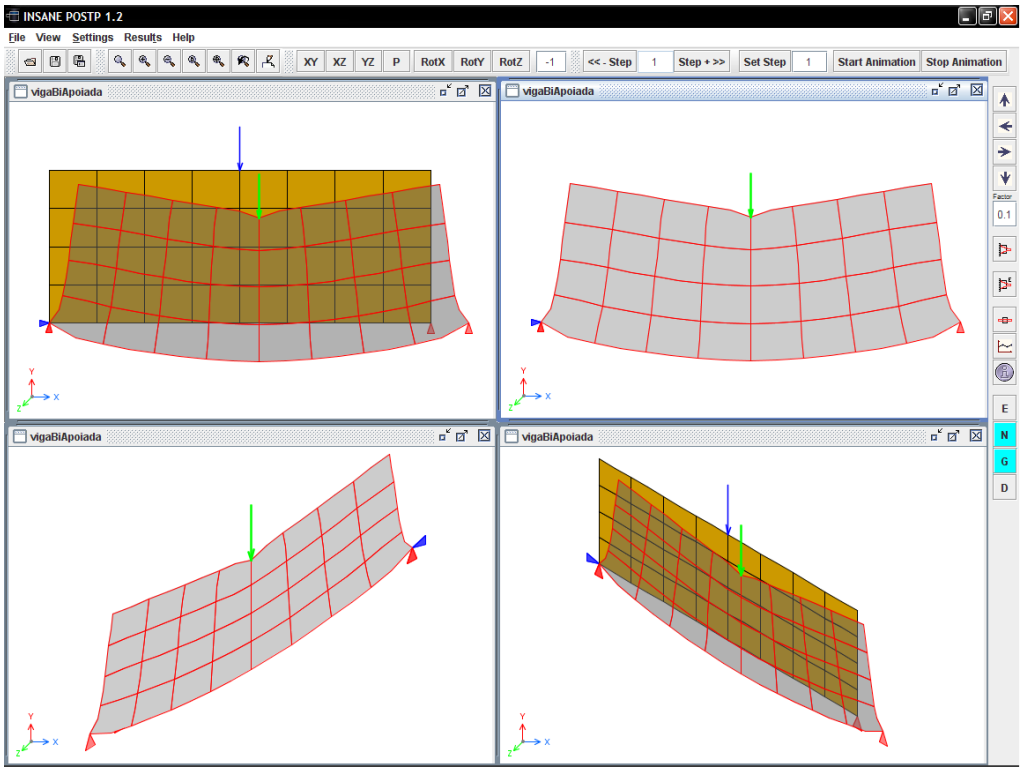


Figura 10.10: Visualização do estado deformado da malha em múltiplas vistas.

10.6 Visualização de Resultados por Isofaixas

A visualização das isofaixas pode ser relativa a valores calculados no nós ou nos elementos, usando a malha do modelo ou a malha gerada pela triangulação, como base para a representação.

A escolha da representação é feita pelo usuário da forma mais conveniente para a análise. É recomendável que ao se utilizar malhas muito refinadas as isofaixas sejam traçadas a partir da própria malha, sem a subdivisão triangular de domínios, pois, assim, a representação é mais fiel ao modelo.

Grandezas nodais (por exemplo, deslocamentos e forças) são calculadas durante o processamento do modelo e passadas diretamente ao pós-processador. Já as grandezas internas (como tensão e deformação) são calculadas no momento em que o pós-processador é carregado.

O pós-processador também permite a composição e posterior visualização de grandezas vetoriais e tensoriais.

10.6.1 Isofaixas para Grandezas Nodais

Os graus de liberdade do modelo não são as únicas grandezas que possuem representação nos nós. Grandezas internas também podem ser calculadas nos nós. Estas grandezas são calculadas após o processamento e atribuídas aos vértices do modelo de semi-arestas.

A exibição das isofaixas requer a escolha do valor a ser representado, dentre as diversas grandezas disponíveis no modelo, a partir do diálogo mostrado na figura 10.11.

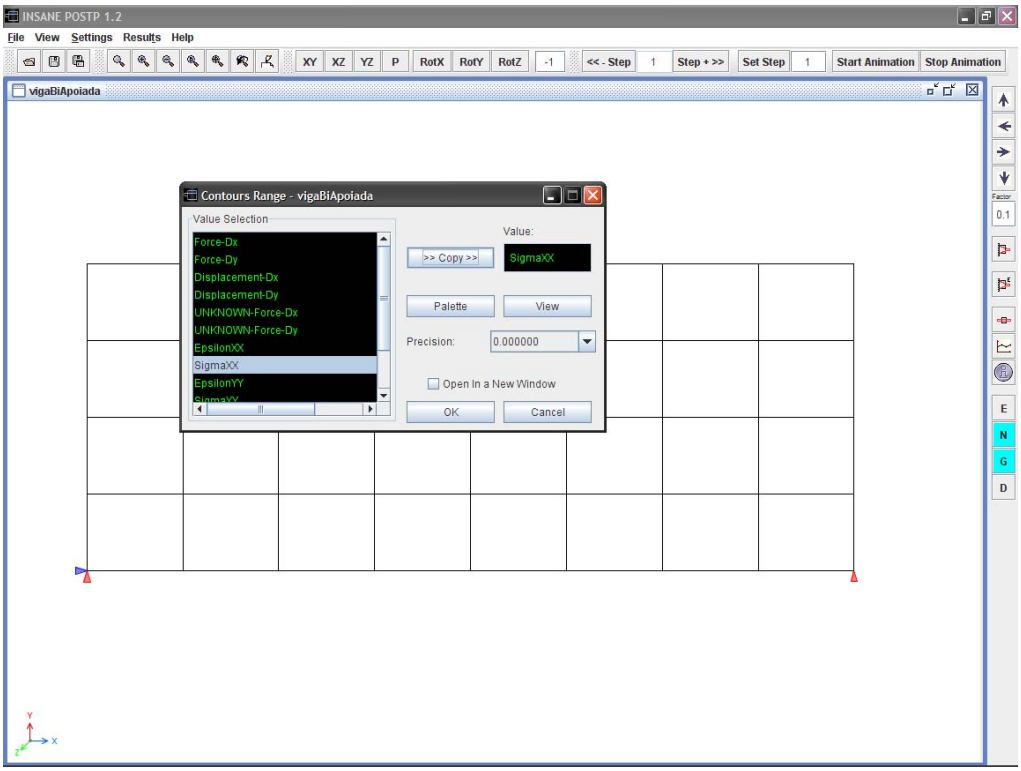


Figura 10.11: Escolha das grandezas.

Para ilustração, a figura 10.12 mostra isofaixas para os deslocamentos na direção X da viga e a figura 10.13 mostra isofaixas para os deslocamentos na direção Y.

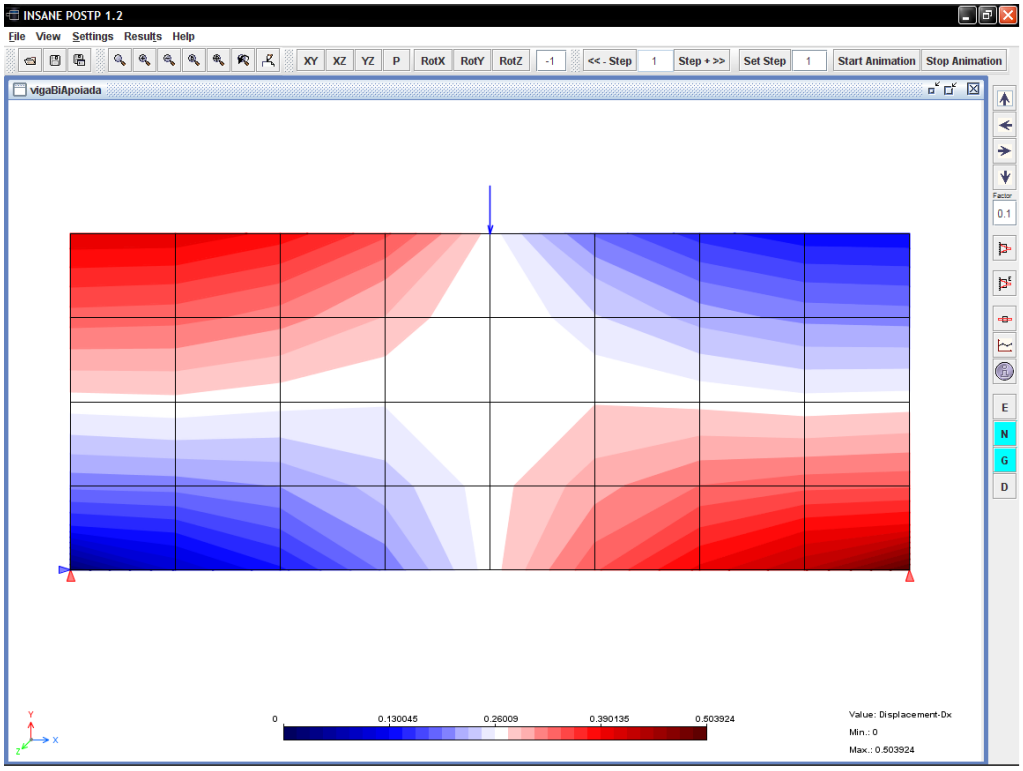


Figura 10.12: Isofaixas dos deslocamentos na direção X.

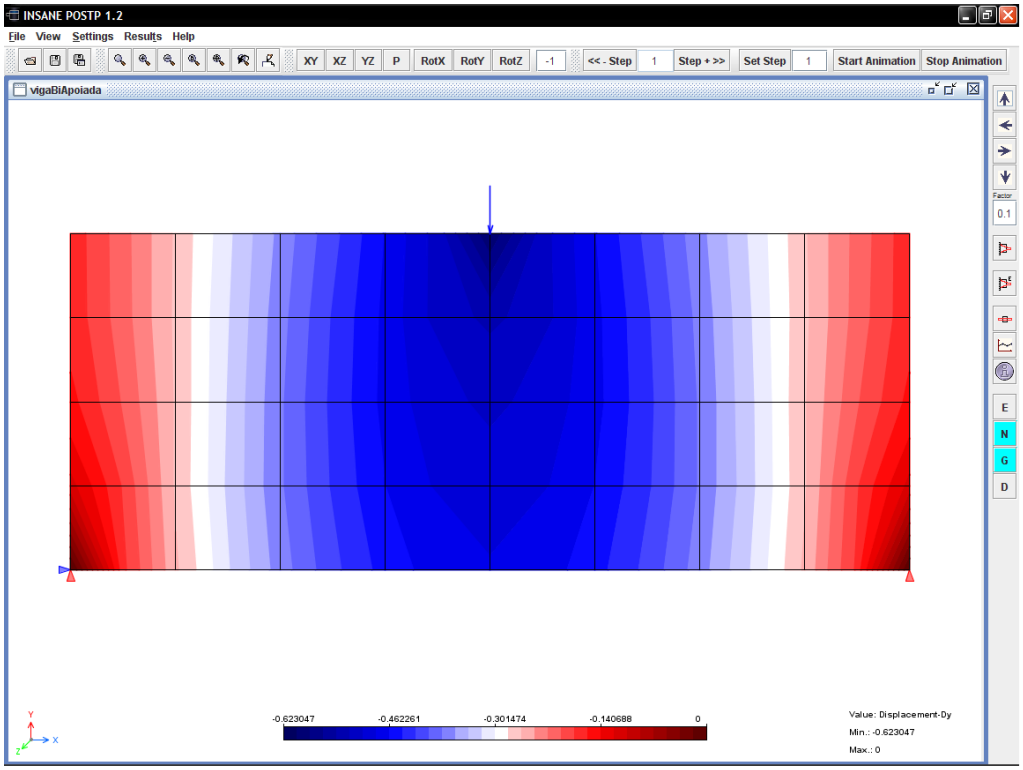


Figura 10.13: Isofaixas dos deslocamentos na direção y.

As isofaixas podem ser plotadas tanto na malha indeformada, como visto nas figuras 10.12 e 10.13, quanto na malha deformada como mostram as figuras 10.14 e 10.15.

Ao compor a configuração deformada da malha, é possível visualizar somente o efeito dos deslocamentos horizontais ou verticais separadamente. A figura 10.16, a seguir, apresenta quatro visualizações das grandezas de deslocamento. As duas primeiras apresentam, respectivamente, as isofaixas para os deslocamentos X e Y com o estado da malha correspondente. Nas duas vistas seguintes, são exibidas, respectivamente, as isofaixas da composição vetorial dos deslocamentos X e Y e a malha deformada juntamente com a indeformada.

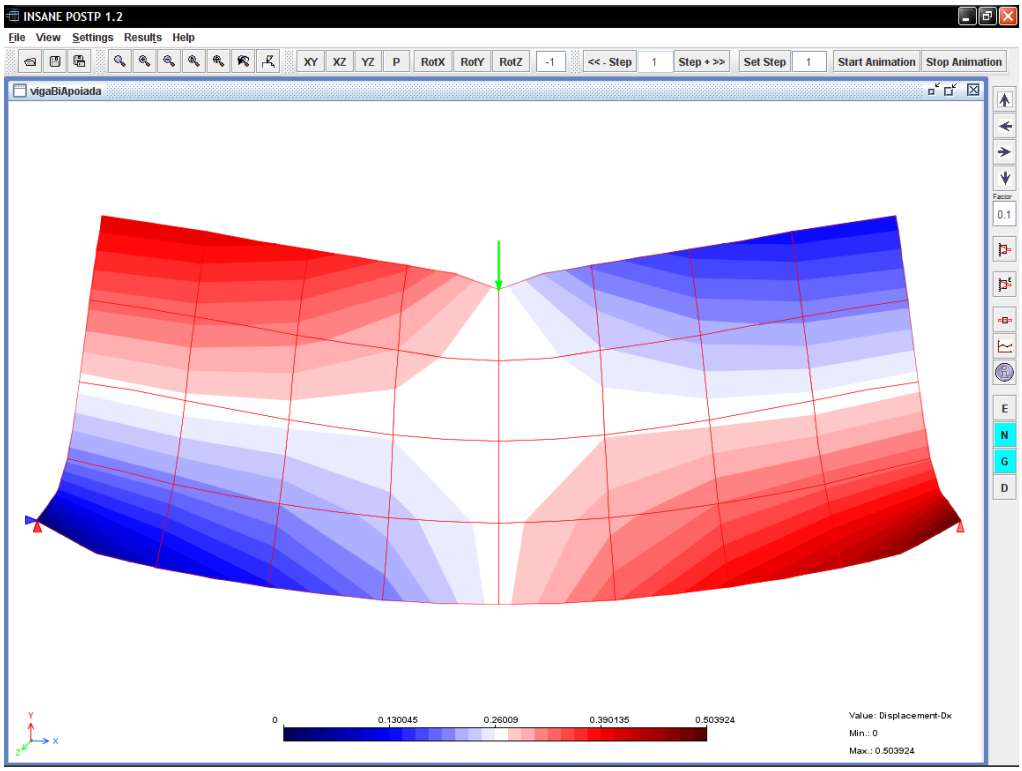


Figura 10.14: Isofaixas dos deslocamentos horizontais na malha deformada.

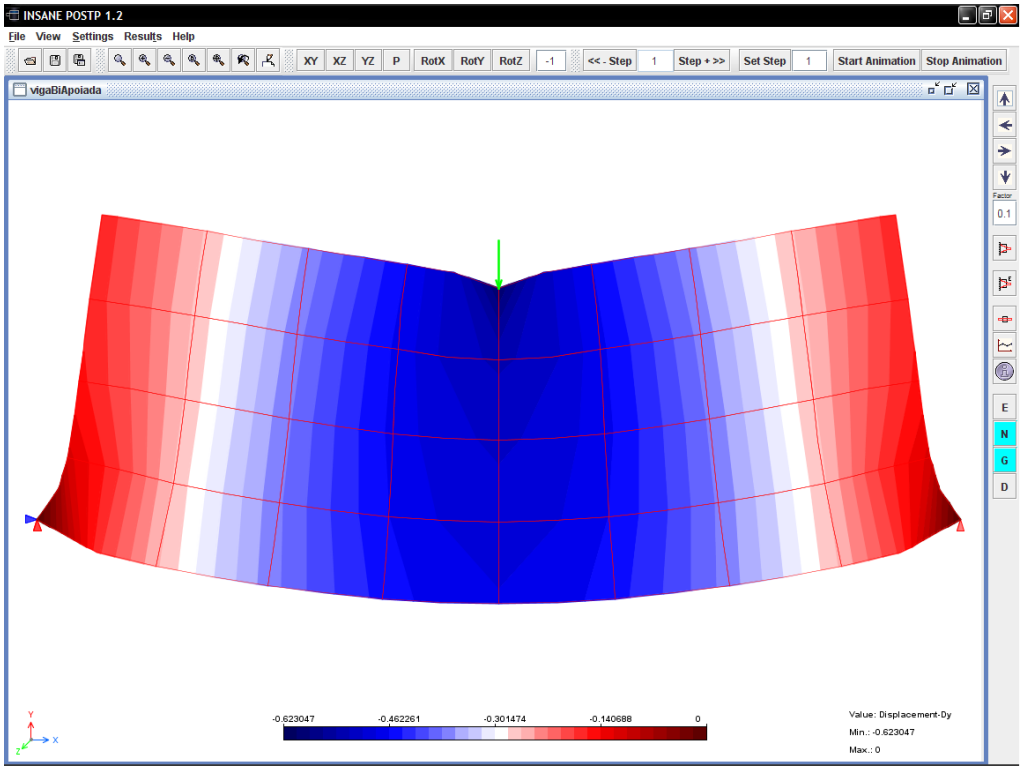


Figura 10.15: Isofaixas dos deslocamentos verticais na malha deformada.

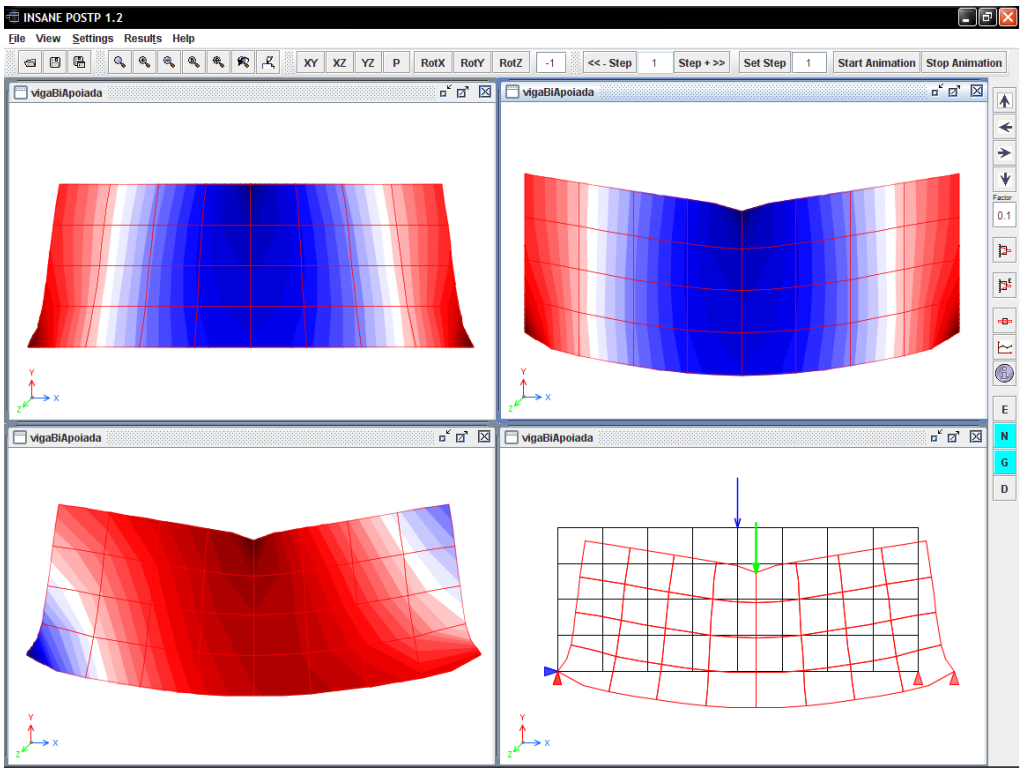


Figura 10.16: Visualização de múltiplas vistas para o estado da malha e as isofaixas.

10.6.2 Isofaixas para Grandezas Avaliadas nos Elementos

As grandezas internas nos elementos podem ser representadas de três maneiras diferentes: a primeira usa a malha de elementos finitos como base para a representação, a segunda usa uma triangulação de Delaunay, nos pontos de Gauss, separadamente em cada elemento, como base para a representação e a terceira usa uma triangulação de Delaunay, nos nós e pontos de Gauss, elemento por elemento, como base para a representação.

A seleção das grandezas nos elementos é feita através de um diálogo com uma lista de grandezas possíveis de serem exibidas, podendo-se optar por suavizar ou não os resultados.

A figura 10.17 mostra a variação das tensões σ_{xx} usando a subdivisão triangular do domínio respectivo aos pontos de integração em cada elemento. Esta subdivisão está mostrada na figura 10.18. Neste caso as grandezas são calculadas nos pontos de Gauss e, portanto, a opção de suavização não é válida.

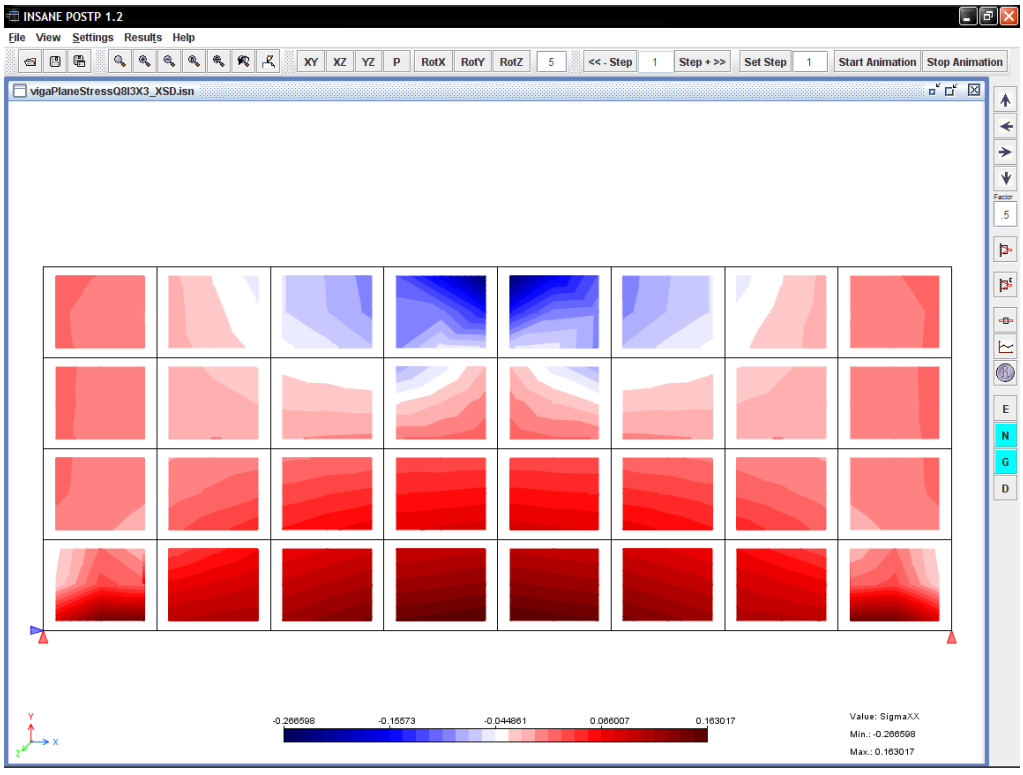


Figura 10.17: Isofaixas para σ_{xx} para a triangulação nos pontos de Gauss.

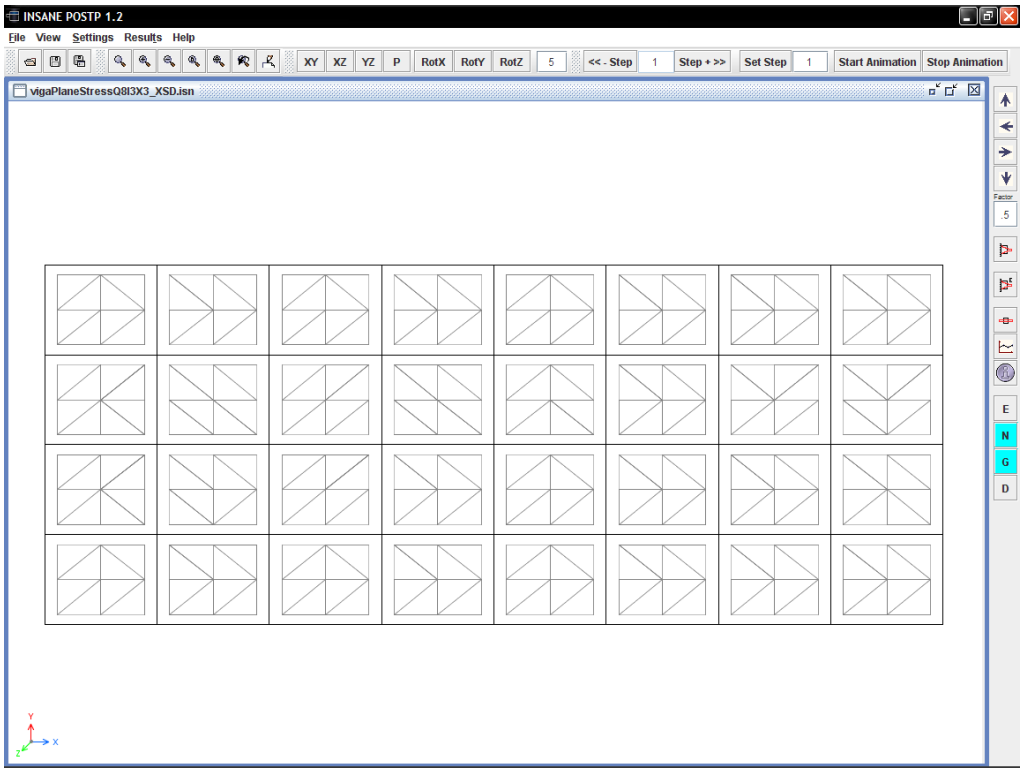


Figura 10.18: Triangulação nos pontos de Gauss.

A figura 10.19 mostra a variação das tensões σ_{xx} usando a malha de elementos finitos para a representação. Neste caso foram usados os valores calculados diretamente no nós, sem a opção de suavização. A representação dos resultados suavizados está mostrada na figura 10.20. A figura 10.21 mostra a variação das tensões σ_{xx}, novamente usando a malha de elementos finitos para representação. Neste caso, entretanto, os valores nodais forma calculados a partir da extrapolação dos valores dos pontos de integração. A representação destes valores com a opção de suavização está mostrada na figura 10.22.

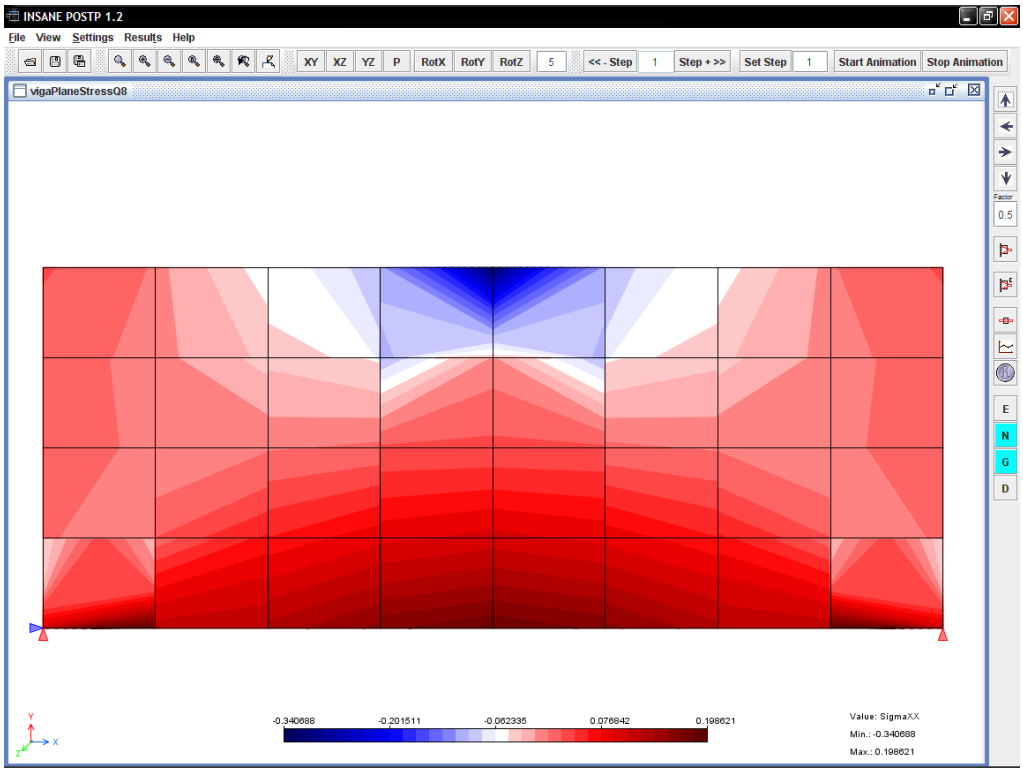


Figura 10.19: Resultados descontínuos para σ_{xx}.

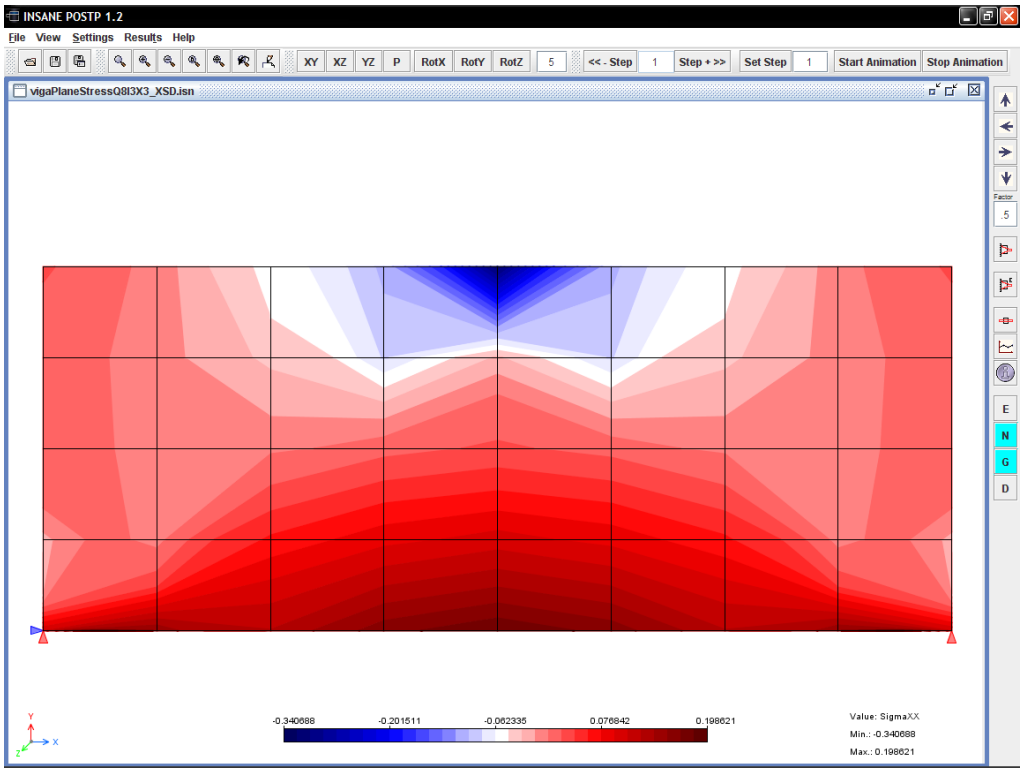


Figura 10.20: Isofaixas para σ_{xx} na malha de elementos.

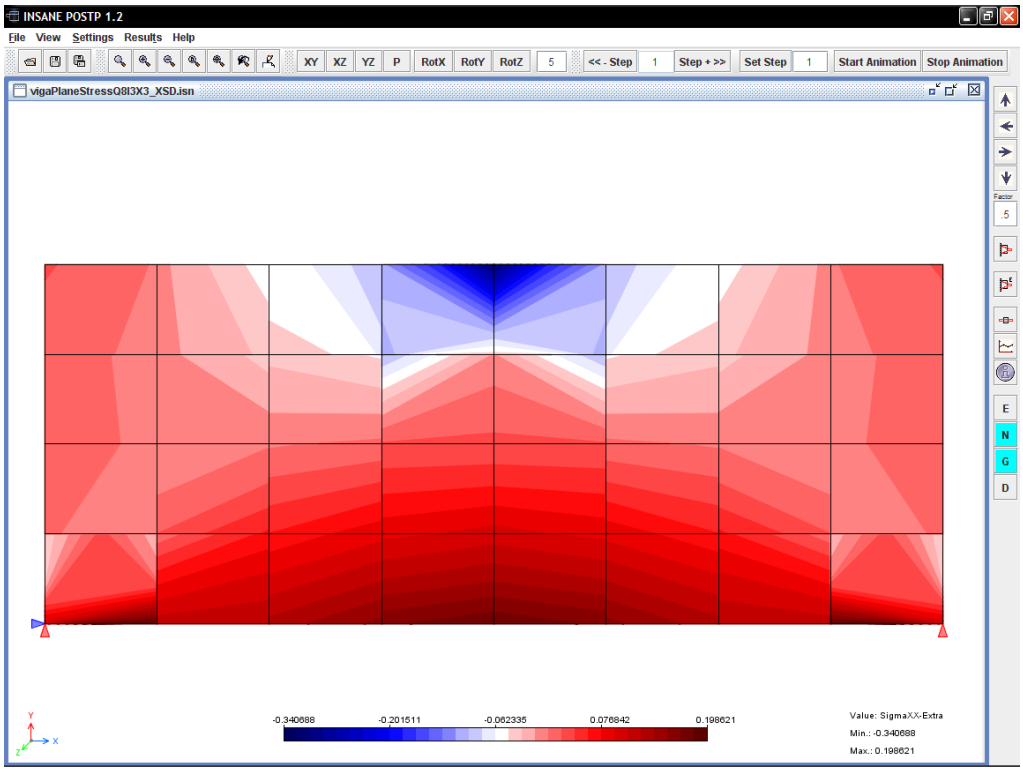


Figura 10.21: Resultados descontínuos para σ_{xx} obtidos por extrapolação.

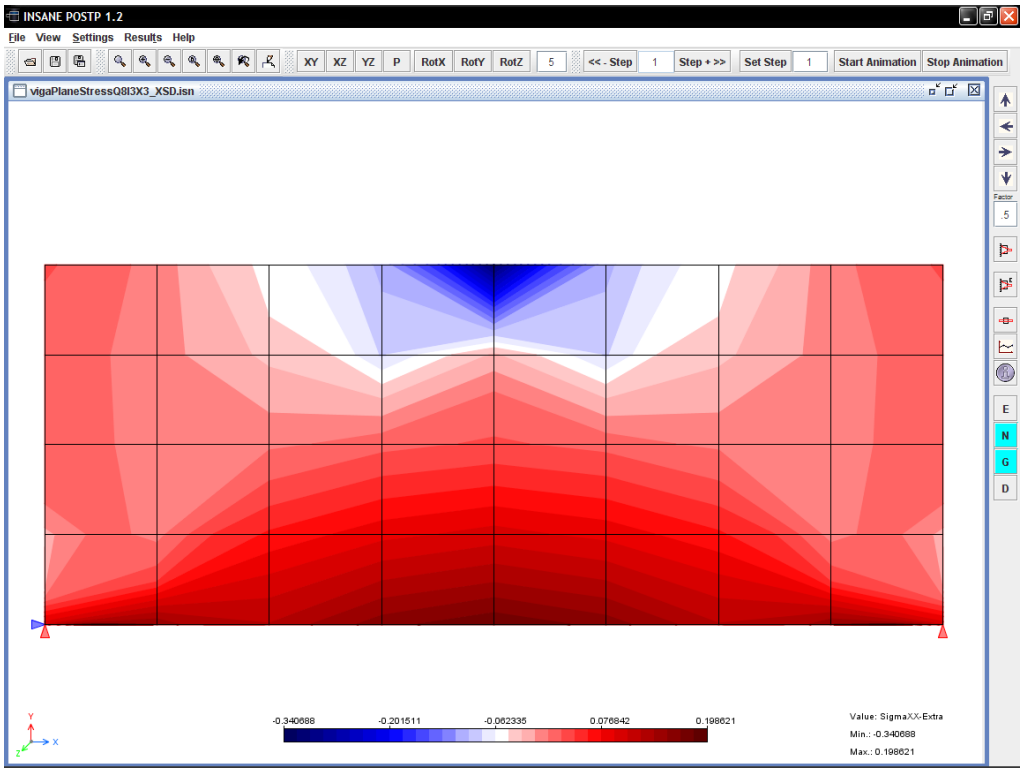


Figura 10.22: Isofaixas para σ_{xx} obtido por extrapolação na malha de elementos.

Na figura 10.23 tem-se a variação das tensões para uma triangulação, elemento por elemento, usando valores nos pontos de Gauss e nos nós. Neste caso os valores adotados para os pontos nodais foram calculados, em cada elemento, diretamente nos nós (sem extrapolação) e, posteriormente, suavização. A visualização da respectiva triangulação está mostrada na figura 10.24

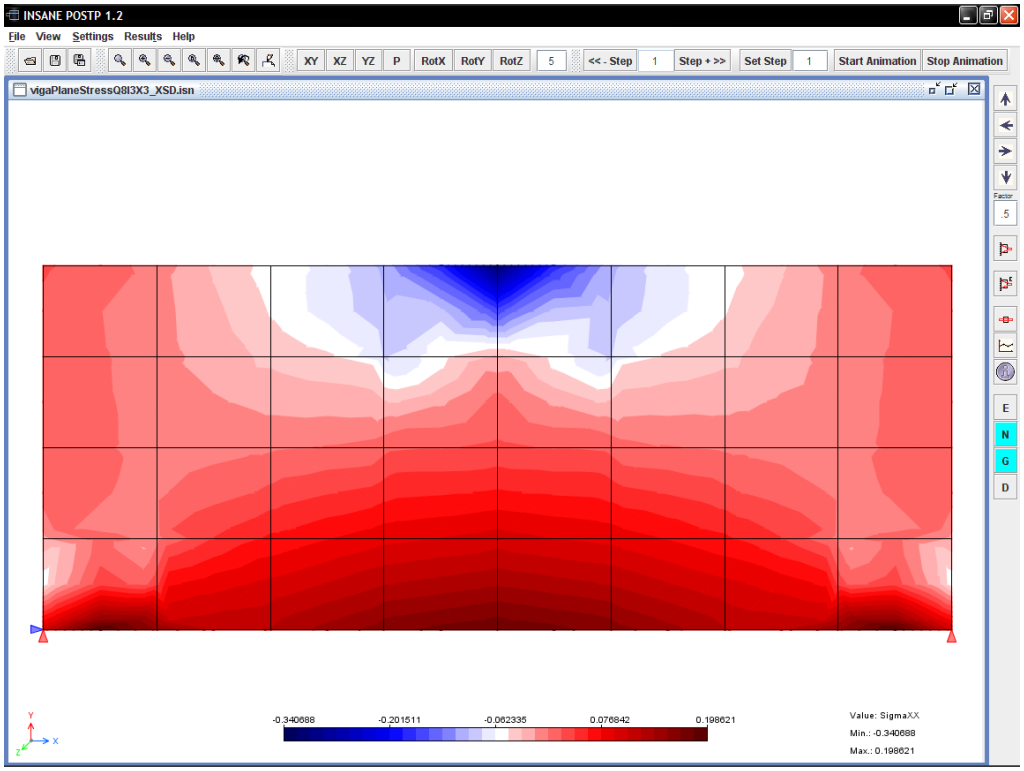


Figura 10.23: Isofaixas para σ_{xx} com subdivisão triangular de toda a malha.

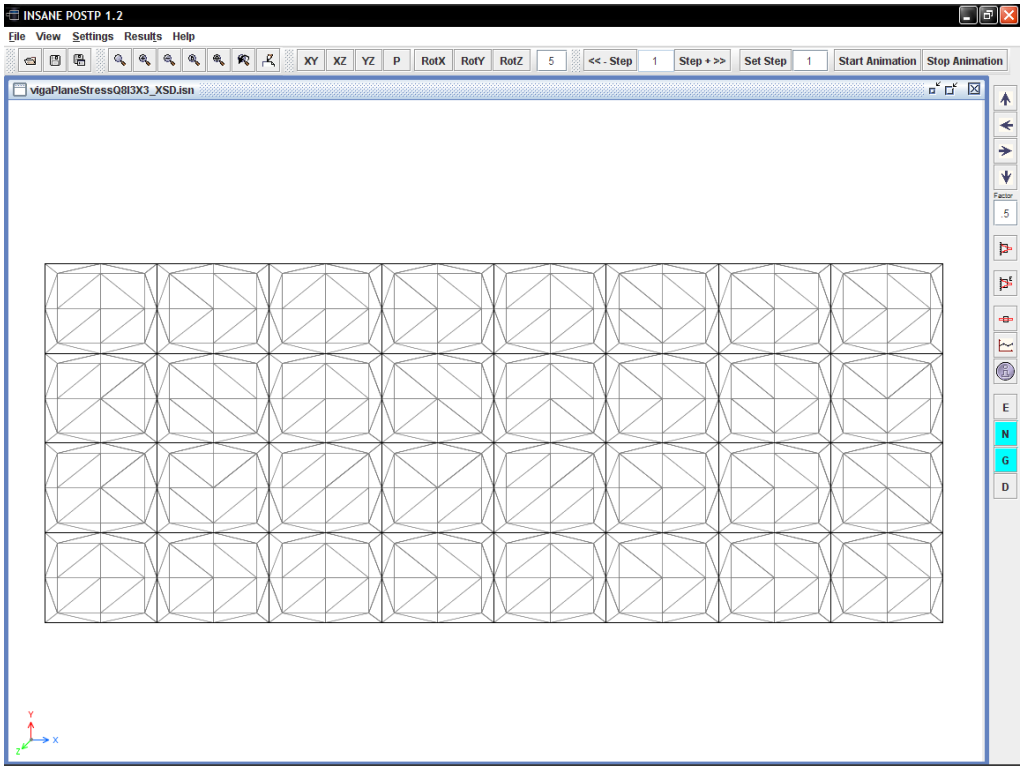


Figura 10.24: Triangulação de toda a malha.

10.6.3 Composição de Resultados para Grandezas Vetoriais e Tensoriais

A composição de grandezas permite a representação de magnitude de vetores e de valores principais de tensores. As grandezas são compostas pelo usuário a partir dos valores nodais ou internos, listados e disponibilizados para a composição.

A figura 10.25 mostra o diálogo para a composição de grandezas vetoriais e a figura 10.26 ilustra o resultado da composição do vetor de deslocamentos nodais.

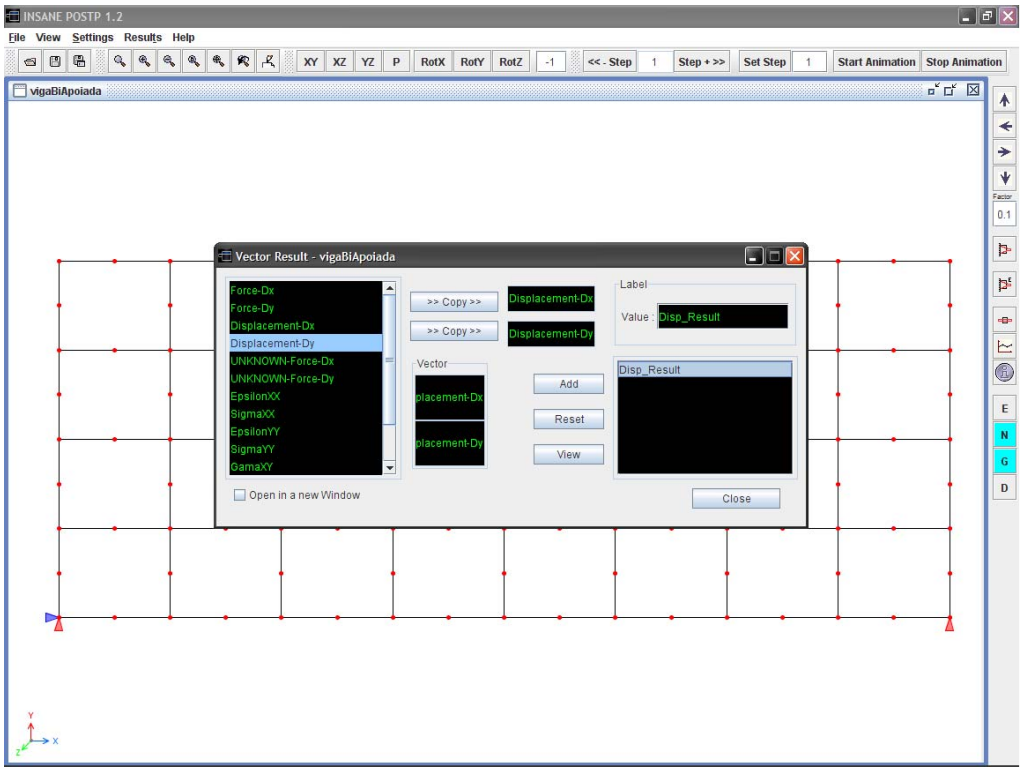


Figura 10.25: Diálogo para a composição de grandezas vetoriais.

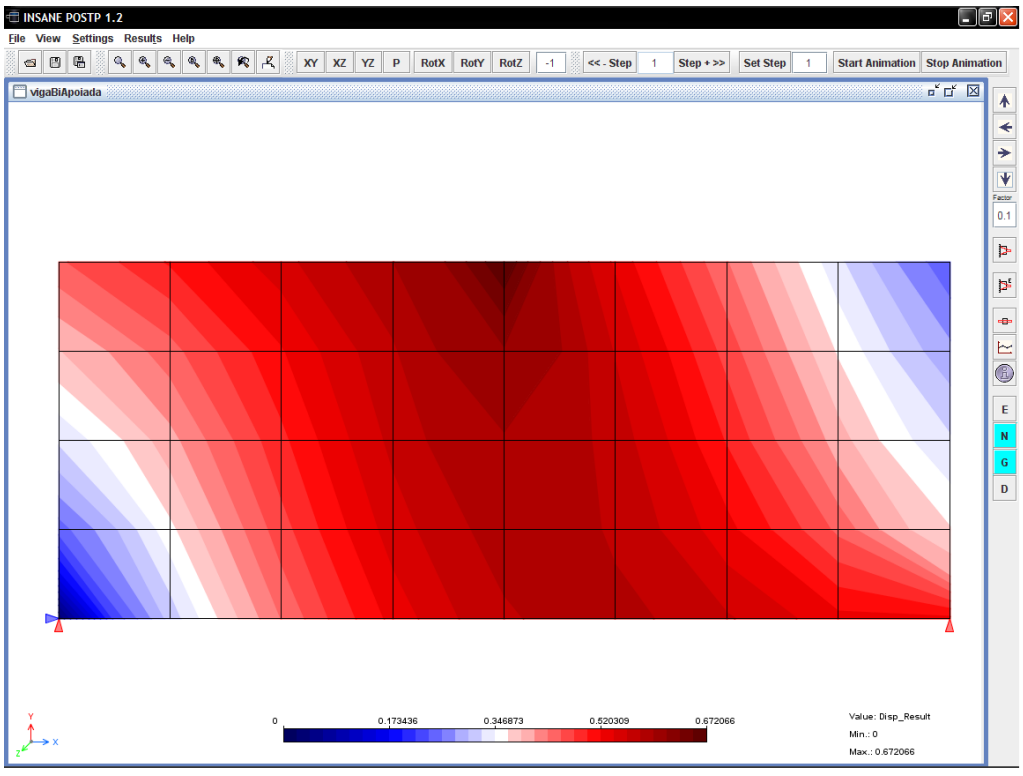


Figura 10.26: Composição de grandezas vetoriais dos deslocamento nas direções x e y.

De forma análoga pode-se compor um tensor e obter a representação dos valores principais deste tensor. A figura 10.27 mostra o diálogo para a composição do tensor. Na figura 10.28 tem-se as isofaixas para o menor autovalor do tensor composto pelas tensões σ_{xx}, σ_{yy} e τ_{xy}.

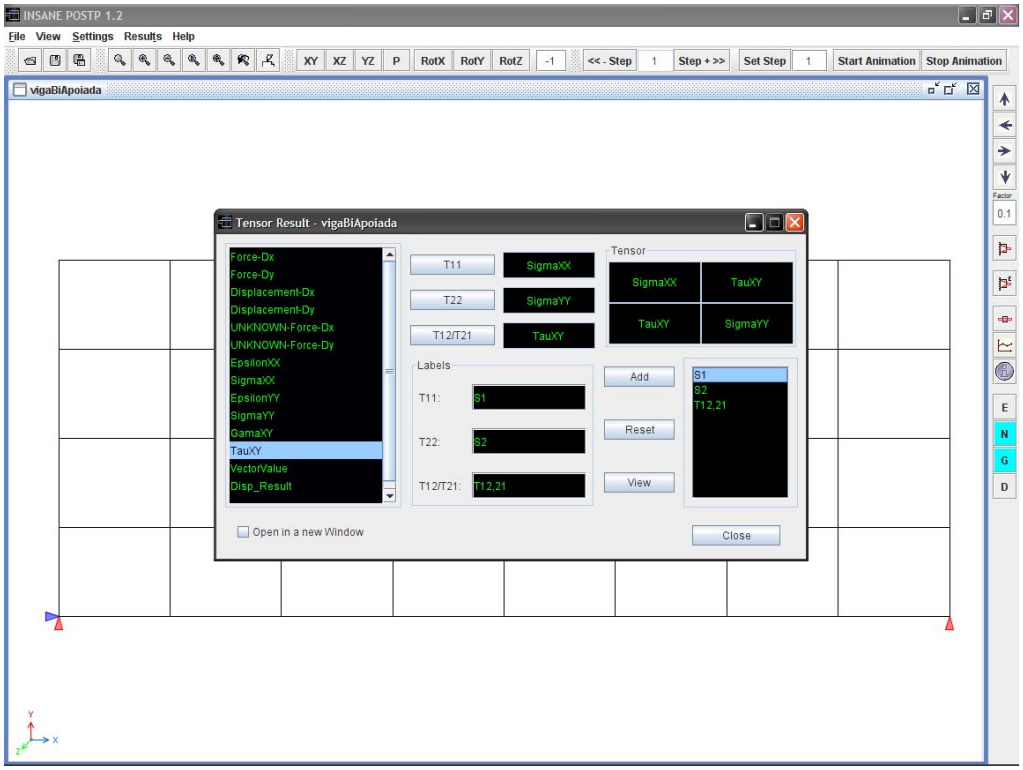


Figura 10.27: Diálogo para a composição de um tensor.

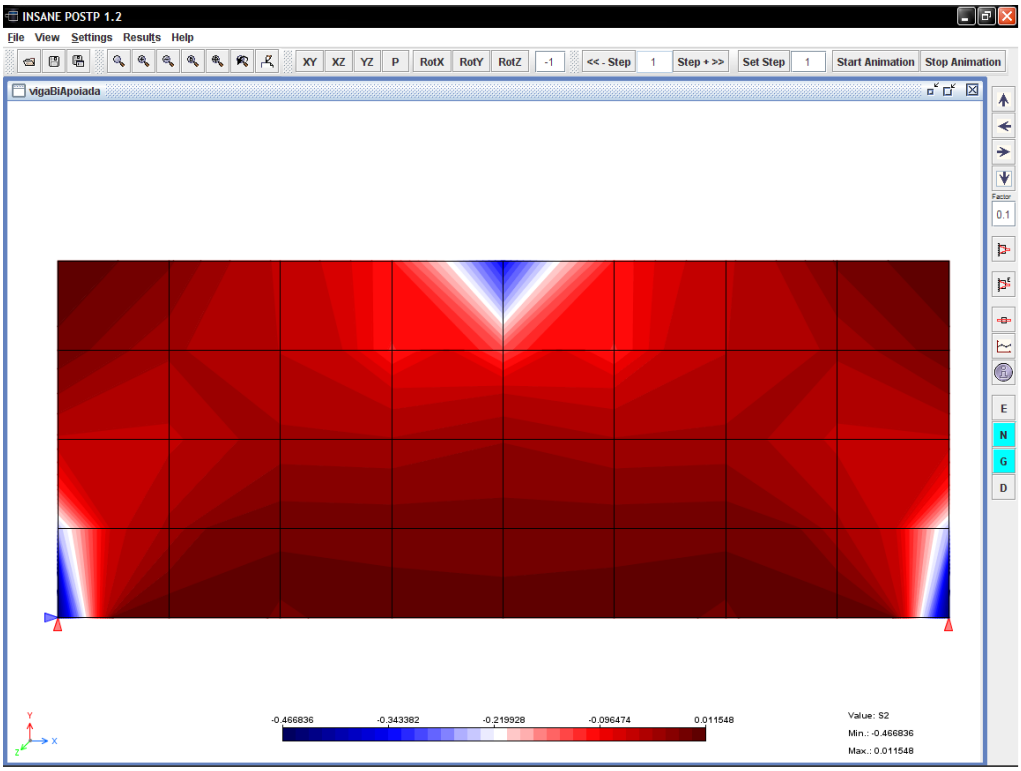


Figura 10.28: Isofaixas dos autovalores do tensor.

10.7 Gráficos das Grandezas ao Longo do Histórico da Análise Não-Linear

A análise não-linear se desenvolve em passos, logo a variação dos resultados do modelo ao longo do histórico da análise é melhor representada através de gráficos. Para tratar os resultados dos passos durante a análise, o pós-processador possui um aplicativo capaz de gerar gráficos para representar o histórico de grandezas.

Os gráficos podem ser para quaisquer grandezas que apresentam variações ao longo da análise. A composição dos gráficos é feita pelo usuário a partir de valores listados e disponíveis para a escolha. Na figura 10.29 tem-se uma visualização de trajetórias de equilíbrio dos nós da seção no meio do vão da viga. Na figura 10.30 tem-se o dialogo para a criação de gráficos.

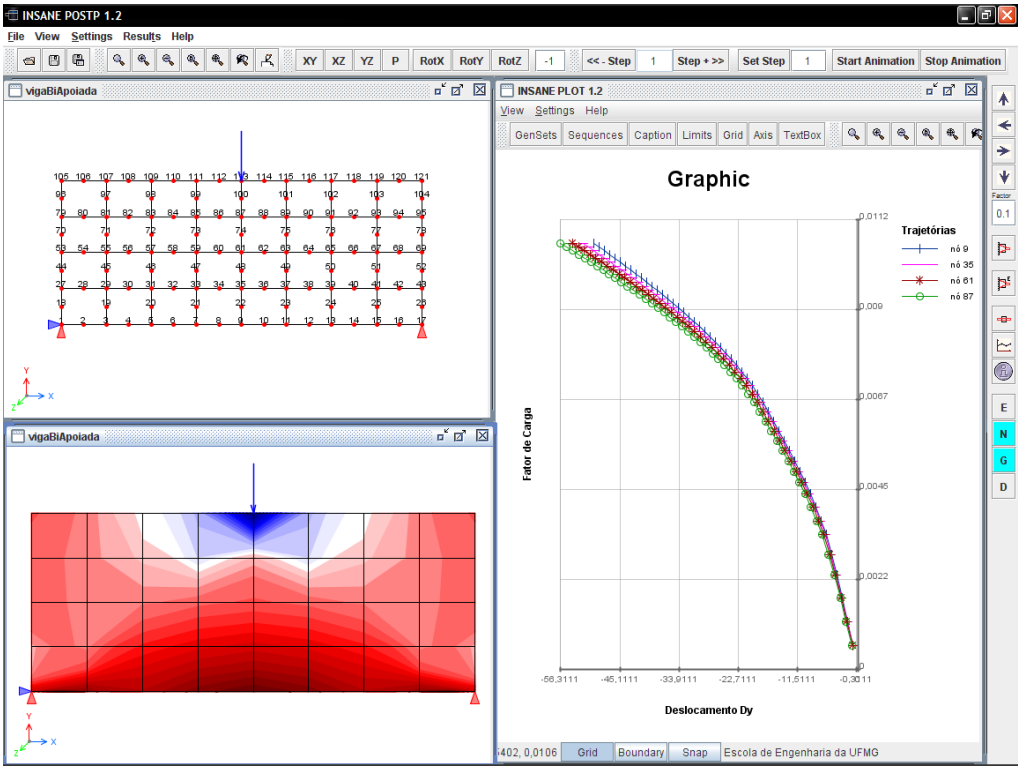


Figura 10.29: Aplicativo gráfico integrado ao pós-processador.

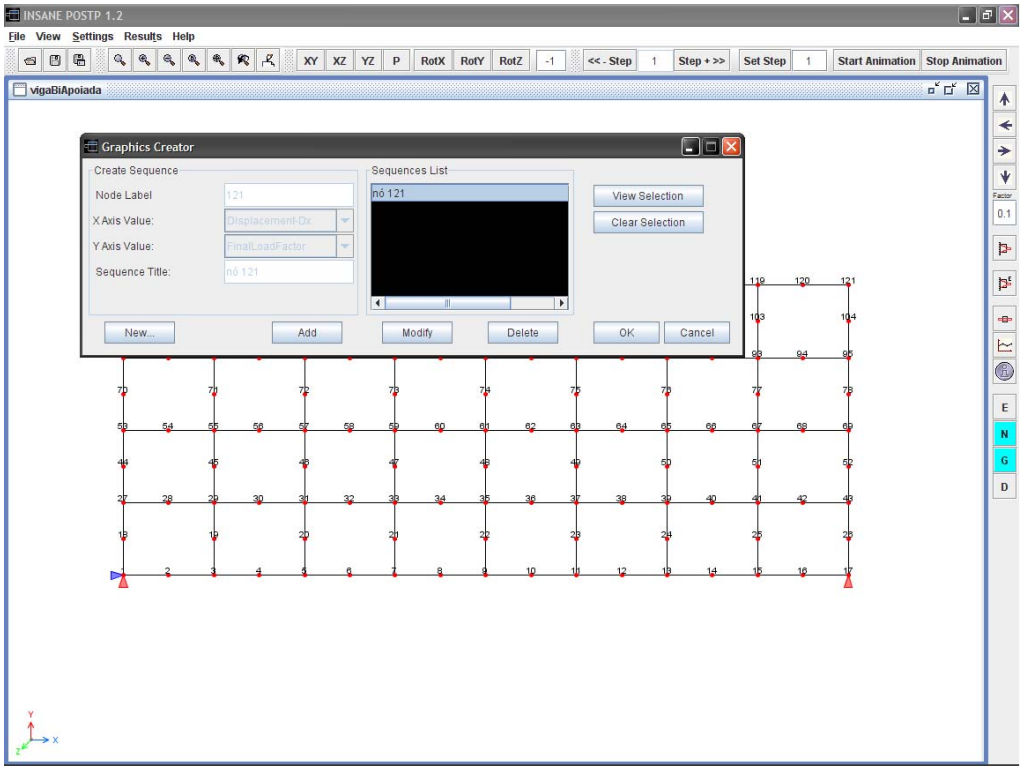


Figura 10.30: Diálogo para a criação de gráficos.

A criação de gráficos inicia-se pela definição do nó que se deseja estudar. Em seguida, as grandezas associadas aos eixos X e Y devem ser escolhidas, dentre aquelas listadas,criando-se uma seqüência. Por fim, a seqüencia é adicionada em uma lista. Para exibir o gráfico seleciona-se a seqüência (ou as seqüências) desejada(s) e acionando o botão **View** do dialogo (figura 10.30), o gráfico será exibido. A figura 10.31 apresenta o gráfico tensão σ_{xx} por deformação ε_{xx} para o nó 9.

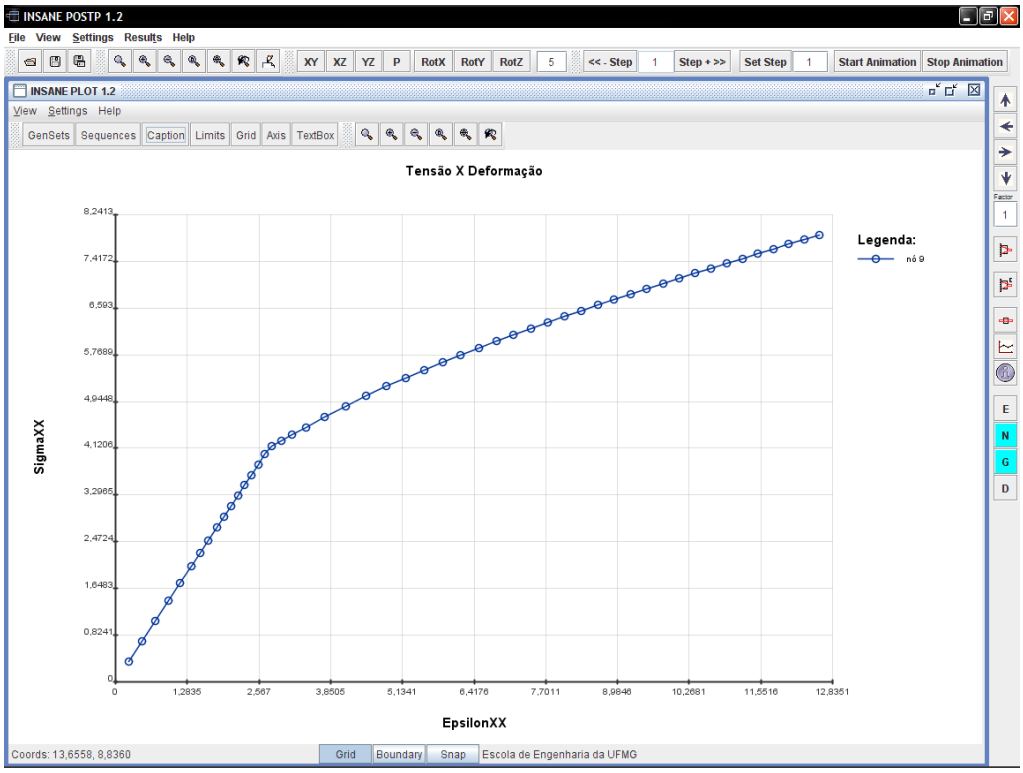


Figura 10.31: Gráfico da variação da Tensão x Deformação do nó 9 ao longo da análise.

Vários gráficos podem ser criados a partir dos dados obtidos na análise. O gráfico mostrado na figura 10.32 representa as trajetórias de equilíbrio, para os deslocamentos na direção x, dos nós situados nos extremos superiores da viga (nós 121 e 105). Na figura 10.33 tem-se múltiplas vistas com diferentes gráficos de resultados do modelo.

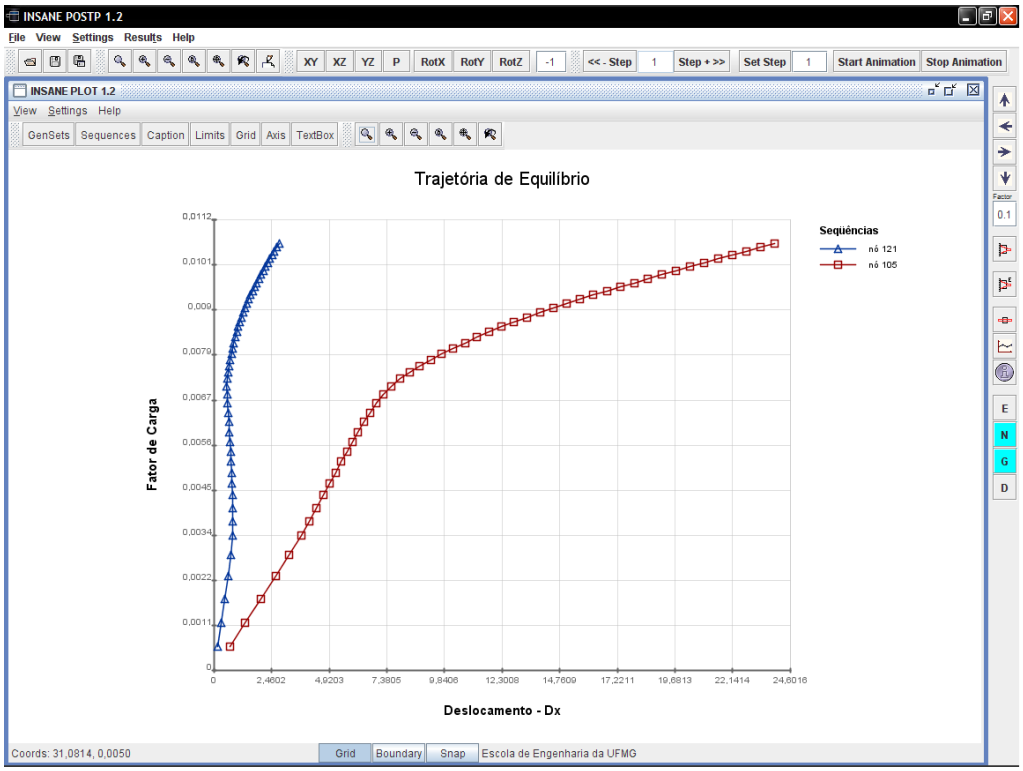


Figura 10.32: Trajetórias de equilíbrio dos nós 105 e 121.

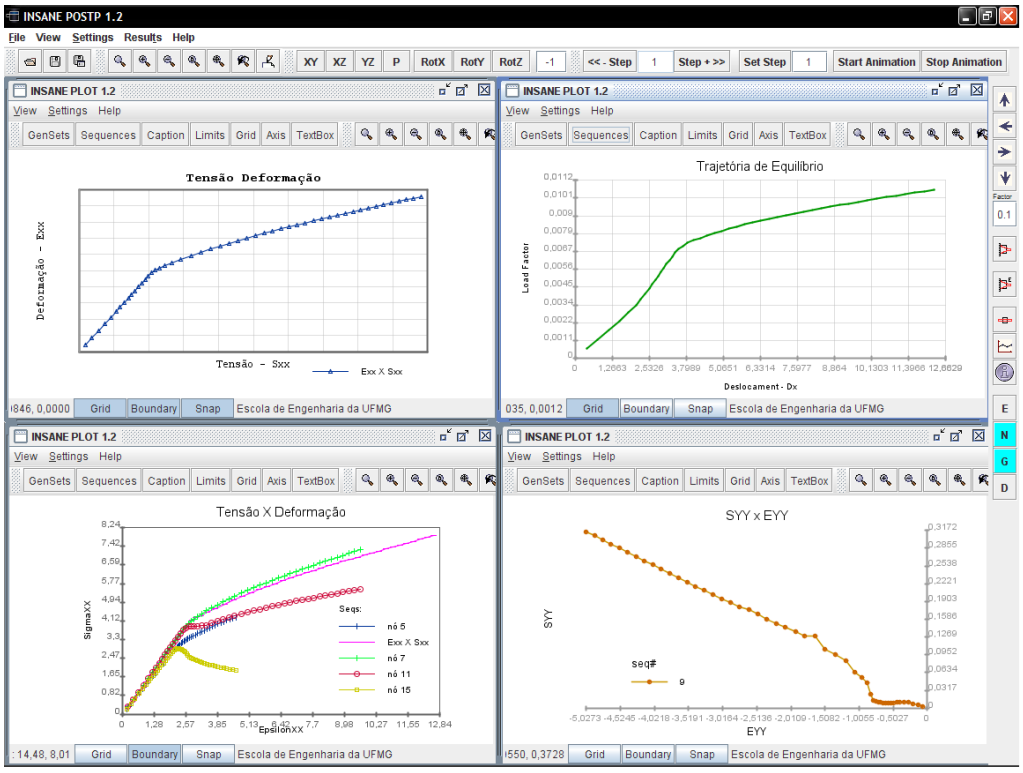


Figura 10.33: Gráficos em múltiplas vistas.

10.8 Pós-Processamento Sincronizado ao Processamento

As funcionalidades até agora descritas são válidas também para o pós-processamento sincronizado ao processamento. O sincronismo permite a visualização dos resultados em tempo de processamento do modelo.

O sincronismo se dá a partir da configuração das grandezas que se deseja visualizar durante a execução. A figura 10.34 mostra o diálogo para a seleção das grandezas a serem exibidas durante o processamento.

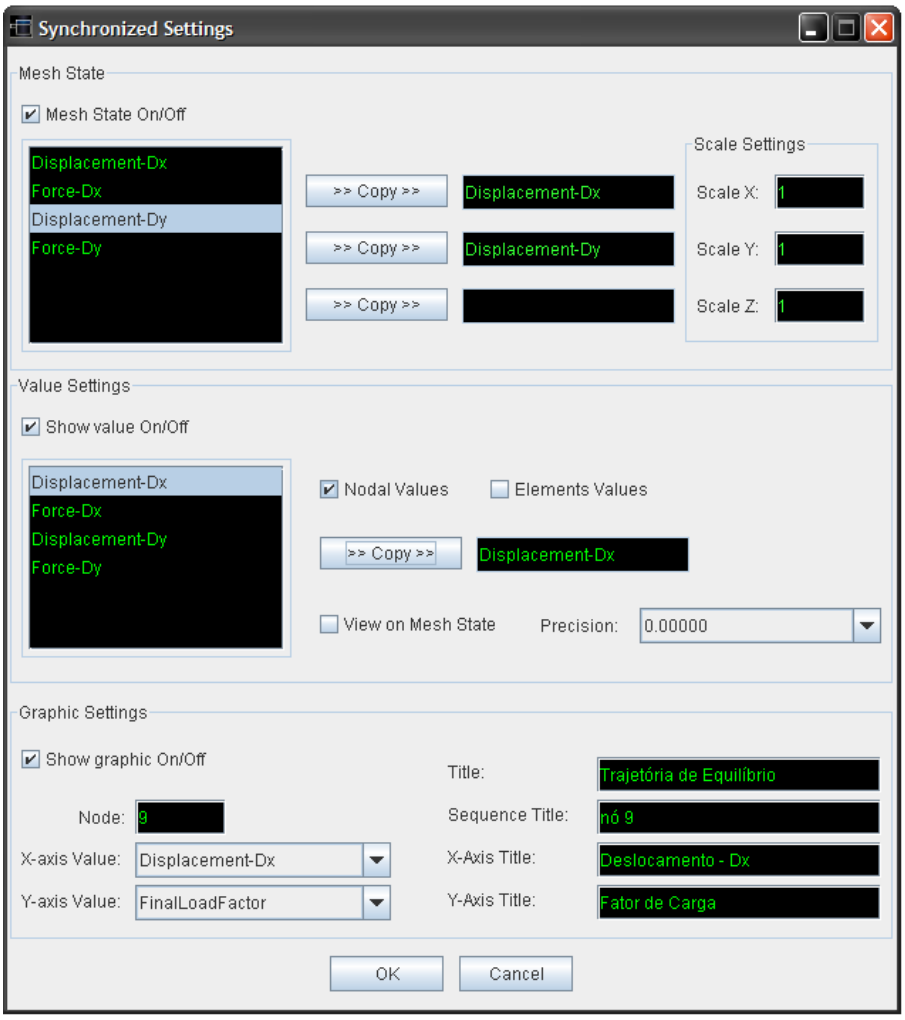


Figura 10.34: Diálogo para configuração do sincronismo.

Durante o processo de sincronismo podem ser visualizadas qualquer uma das

grandezas nodais ou dos elementos, o estado deformado da malha e um gráfico com a variação das grandezas do processo, como ilustra a figura 10.35. Além das grandezas, também pode ser visto um console com a seqüência dos passos processados (figura 10.35).

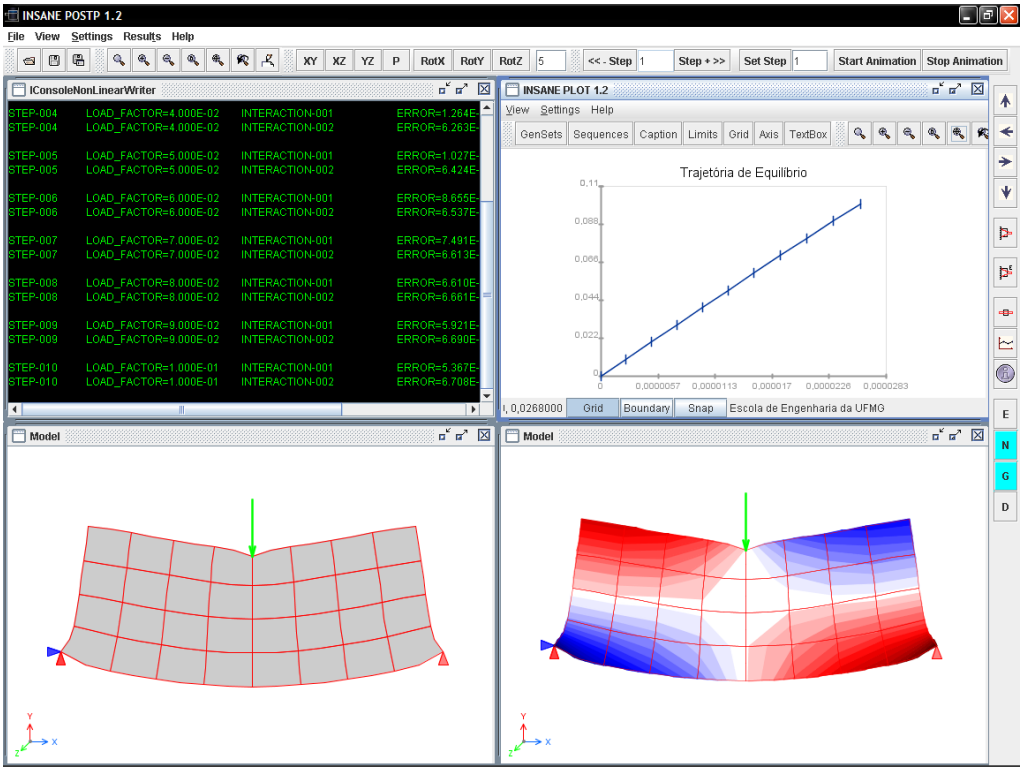


Figura 10.35: Vistas em sincronismo.

10.9 Modelo Genérico e Persistência para o Pós-Processamento

O modelo genérico do pós-processador, como apresentado na seção 9.11, pode ser melhor entendido agora. Seja, por exemplo, um modelo de elementos finitos unidimensionais. Neste caso, a forma de representação das grandezas e a natureza dos resultados são diferentes de modelos bidimensionais, de modo que resultados como tensão e deformação não são visualizados. Entretanto, os deslocamentos, assim como a geração de gráficos do histórico da análise, podem ser exibidos como nos modelos

bidimensionais. Assim sendo, os modelos podem ser informados ao programa por meio de arquivos XML. A figura 10.36 ilustra um pórtico plano, visualizado a partir dos dados de um arquivo XML. No apêndice B podem ser vistos com maiores detalhes os arquivos de definição de dados (DTD) e o arquivo XML referente ao exemplo de pórtico mostrado na figura 10.36.

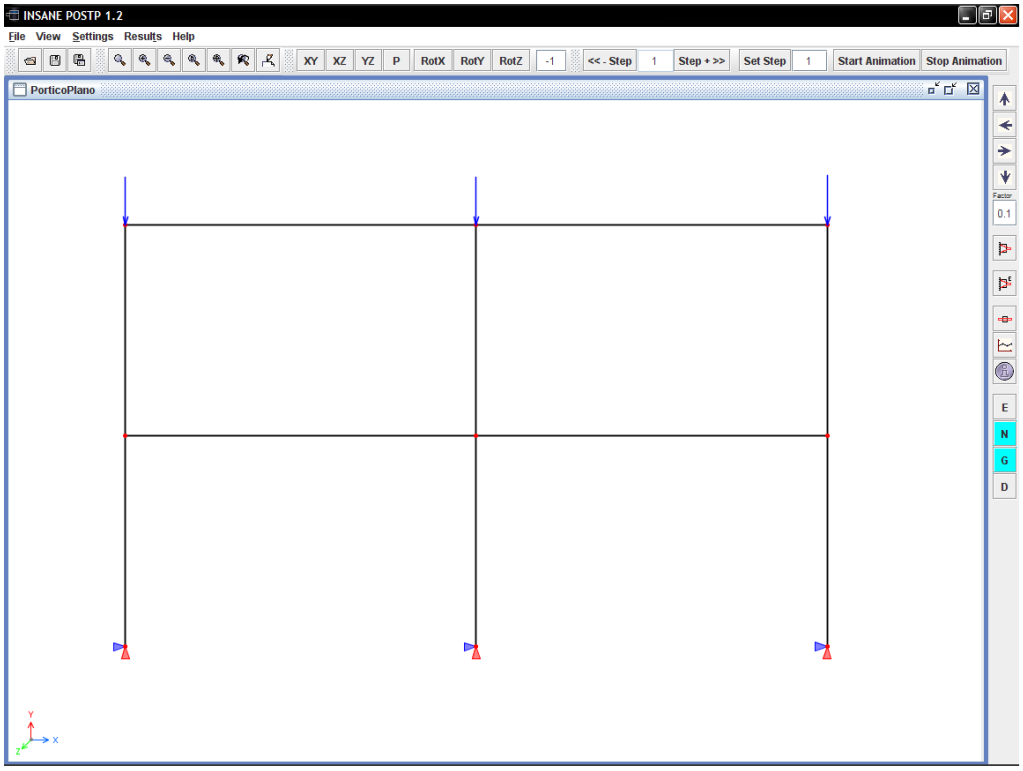


Figura 10.36: Pórtico Plano - Modelo genérico de persistência de dados.

Modelos bidimensionais também podem ser lidos de arquivos XML, resultados obtidos em qualquer programa, desde que persistidos no formato do pós-processador, para que possam ser visualizados. A figura 10.37 apresenta um modelo simples com os resultados carregados a partir de um arquivo XML genérico. O arquivo XML do exemplo pode ser visto no apêndice B.

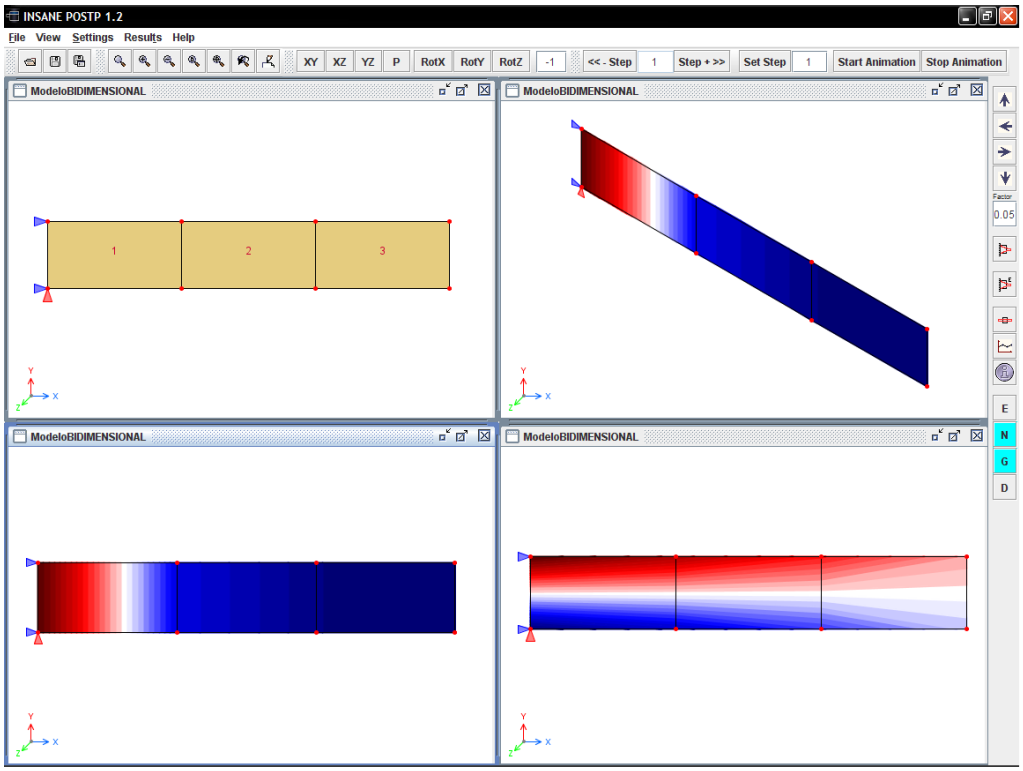


Figura 10.37: Modelo Bidimensional - Modelo genérico de persistência de dados.

Capítulo 11

Exemplos

11.1 Introdução

A fim de explorar mais os recursos e as características do programa, diferentes modelos são apresentados neste capítulo. Para o pré-processamento e o processamento foram usados os recursos atualmente disponíveis no **INSANE**.

Os exemplos aqui apresentados irão se ater aos detalhes da representação dos resultados e dos recursos gráficos interativos do pós-processador.

11.2 Viga Bi-apoiada com Balanço

Neste exemplo apresenta-se uma viga bi-apoiada com um balanço submetida a cargas concentradas, como pode ser visto na figura 11.1. O processamento se desenvolveu em 100 passos com uma estratégia de iteração por controle de carga. No fim do processamento, o modelo passa para o estado deformado mostrado na figura 11.2.

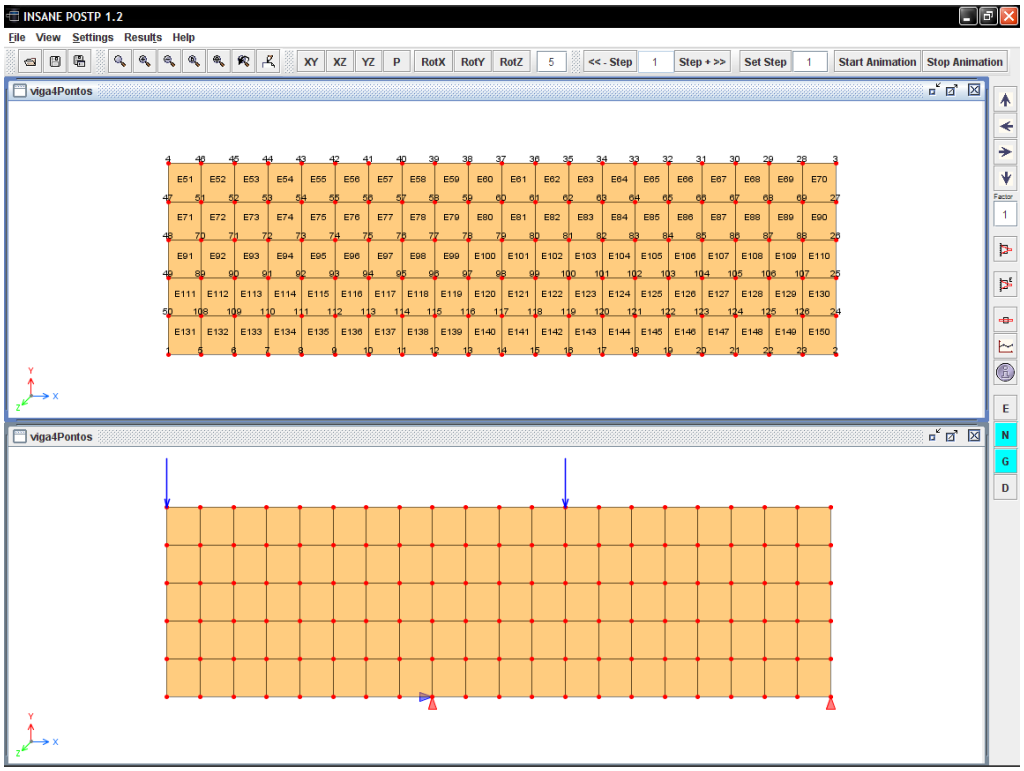


Figura 11.1: Configuração do modelo.

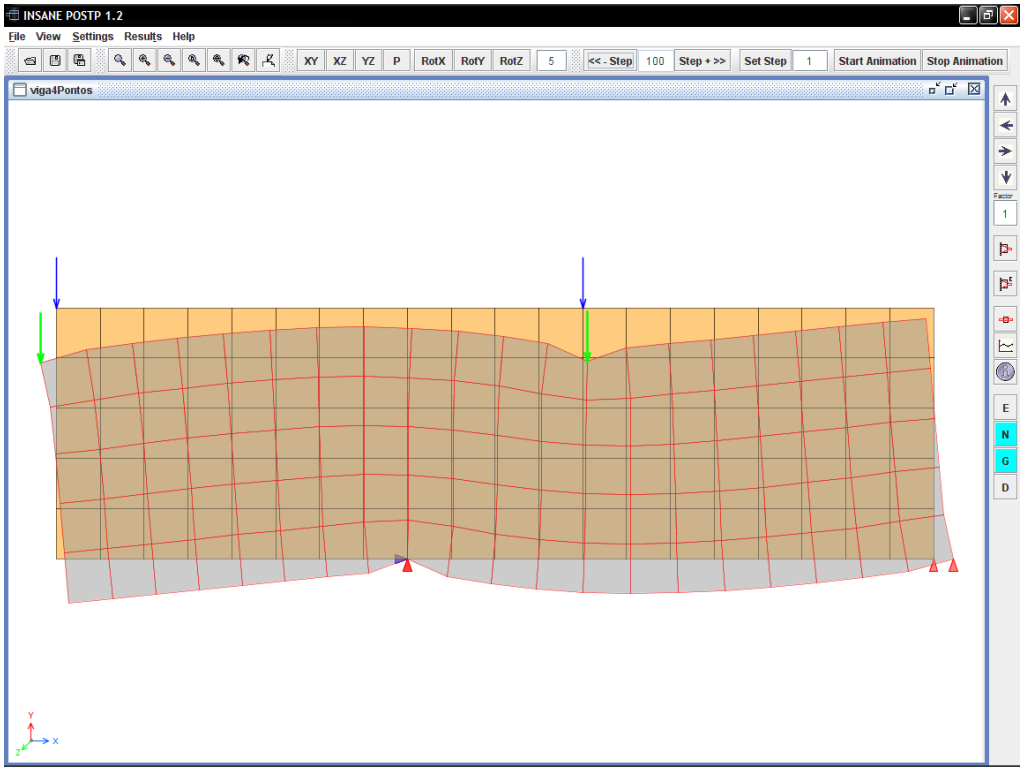


Figura 11.2: Deformada do modelo.

Os resultados podem ser vistos ao longo dos vários passos da solução. A figura 11.3 exibe a variação das deformações γ_{xy} para os passos 1, 35, 70 e 100 da análise.

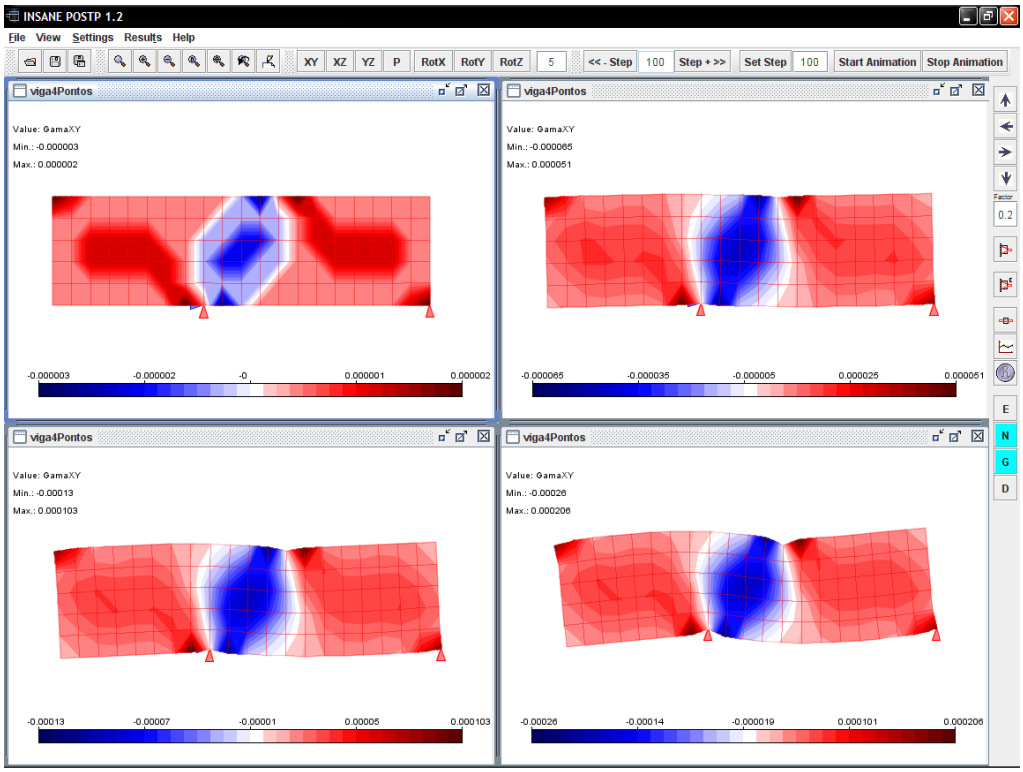


Figura 11.3: Resultados para γ_{xy} ao longo da análise.

A variação das grandezas do processo como um todo pode ser vista através de gráficos. Estes gráficos são formados por valores avaliados nos nós da malha. Desta forma basta escolher as grandezas disponíveis e o nó em questão. Na figura 11.4 verifica-se a variação das deformações γ_{xy} com o incremento de carga ao longo dos passos.

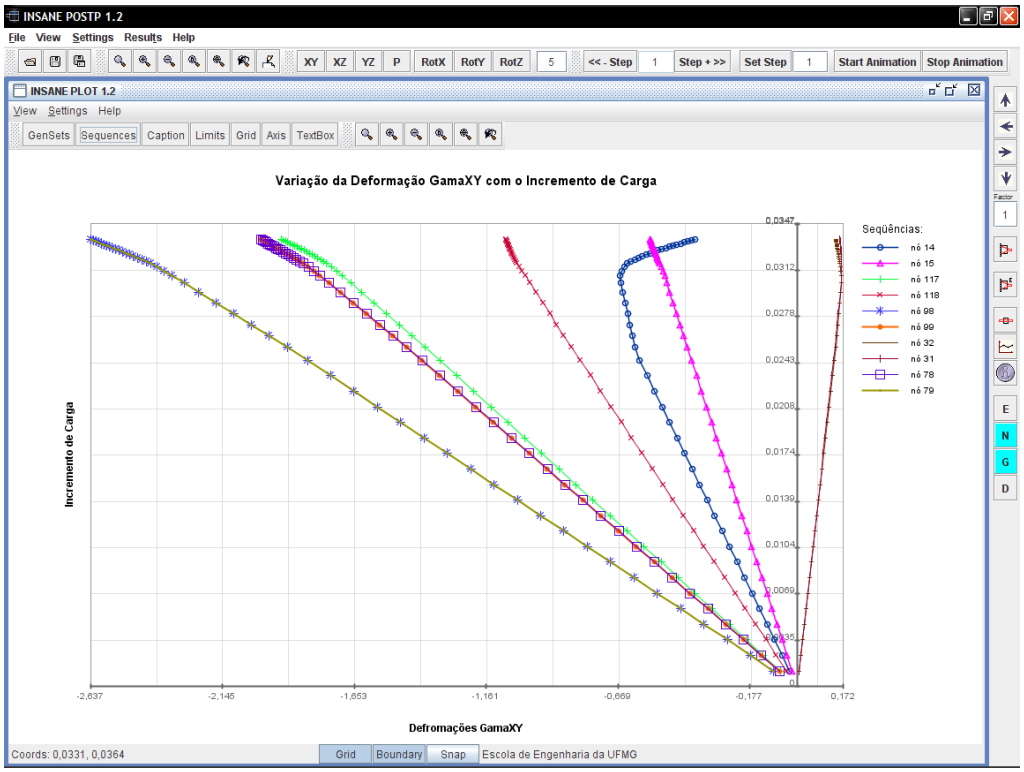


Figura 11.4: Gráfico para a variação das deformações com o incremento de carga.

A variação do campo de deslocamentos pode ser obtida através da composição do vetor correspondente. Esta variação está representada, para o primeiro e último passo da análise, figura 11.5. Valores principais de tensões e deformações podem ser calculadas e exibidos, como ilustra a figura 11.6, que mostra as deformações principais máximas para os primeiro e último passos da análise.

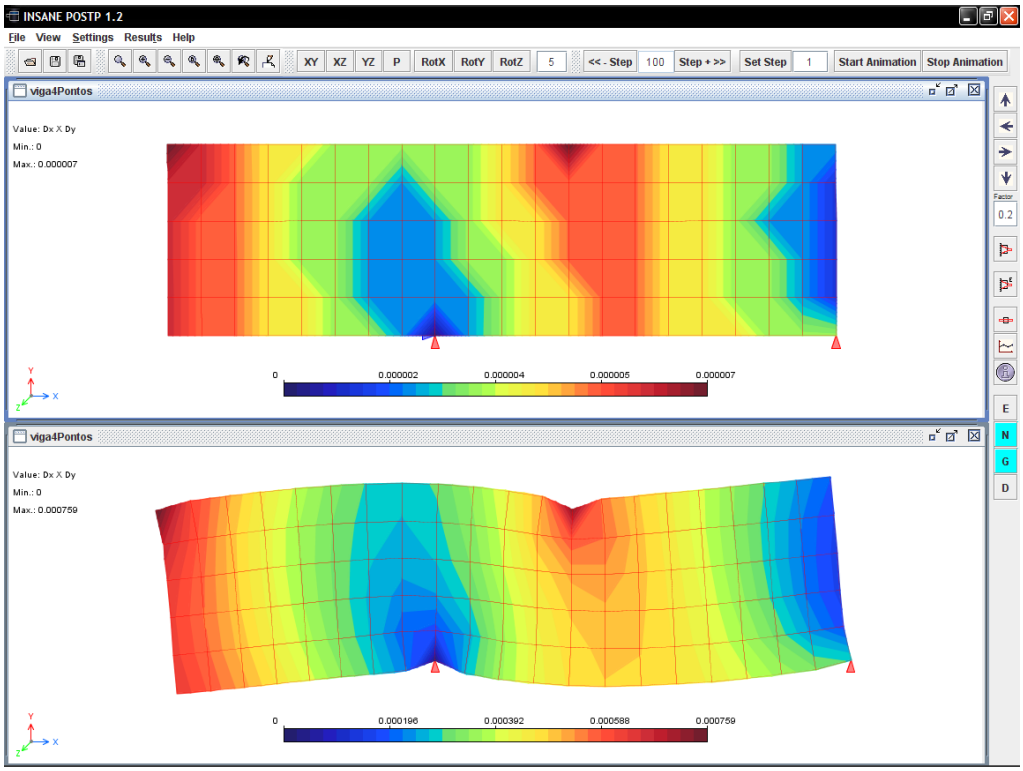


Figura 11.5: Campos de deslocamentos resultantes.

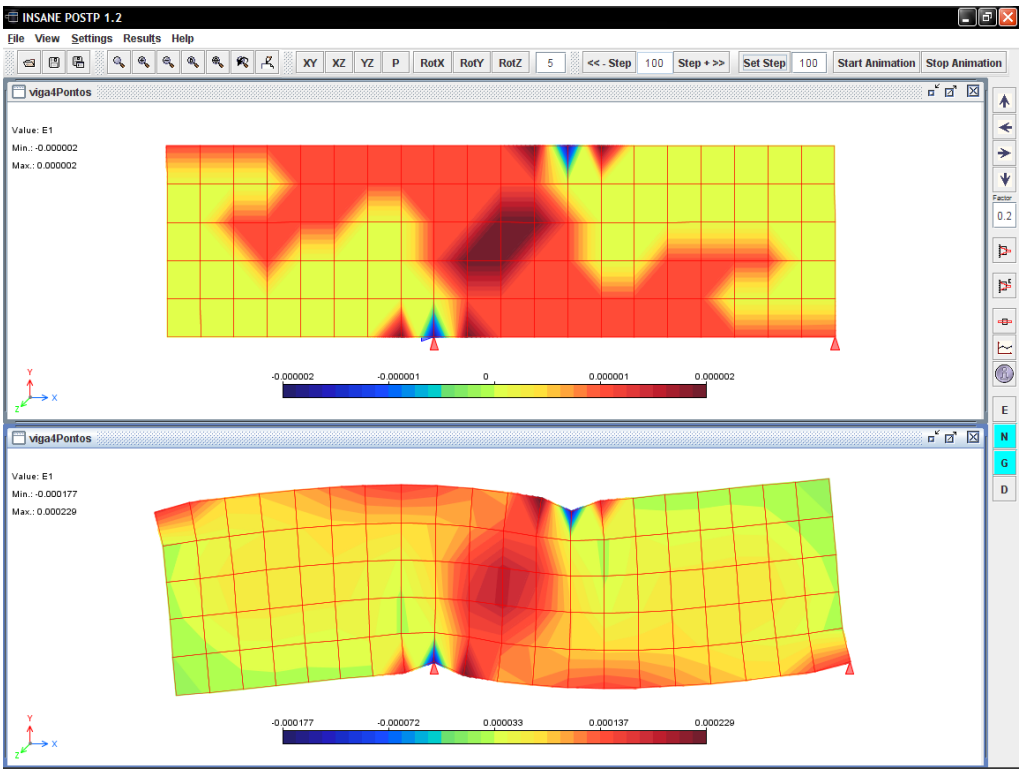


Figura 11.6: Deformações principais máximas.

11.3 Modelos de Flexão de Placas

Nesta seção serão apresentados modelos de flexão de placas. No primeiro exemplo apresenta-se uma placa quadrada com carga concentrada no centro, no segundo exemplo, uma placa em balanço com carga aplicada nas bordas livres e no terceiro exemplo, uma laje cogumelo.

11.3.1 Placa com Carga Concentrada no Centro

Seja uma placa quadrada, simplesmente apoiada, discretizada com elementos quadrilaterais de quatro nós, com carga concentrada no centro, como mostra a figura 11.7.

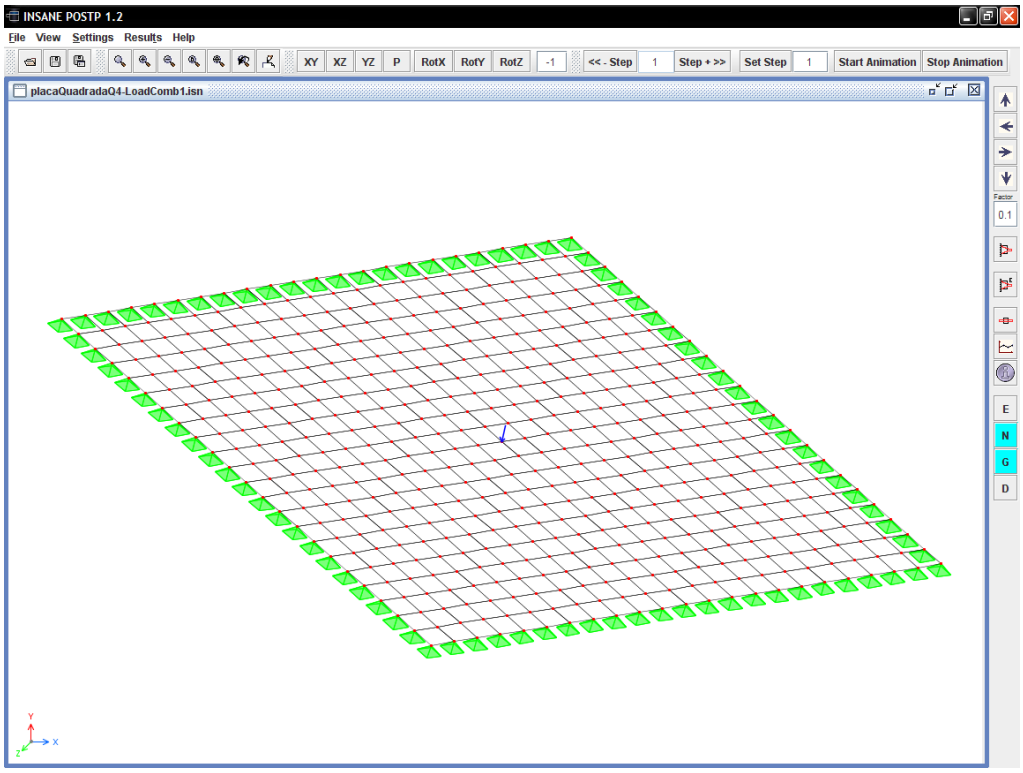


Figura 11.7: Placa quadrada simplesmente apoiada com carga concentrada.

Para este exemplo destaca-se a visualização das isofaixas para os momentos de flexão e as respectivas curvaturas, momento e curvatura de torção, variação da flecha, conforme mostradas nas figuras 11.8, 11.9, 11.10 e 11.11.

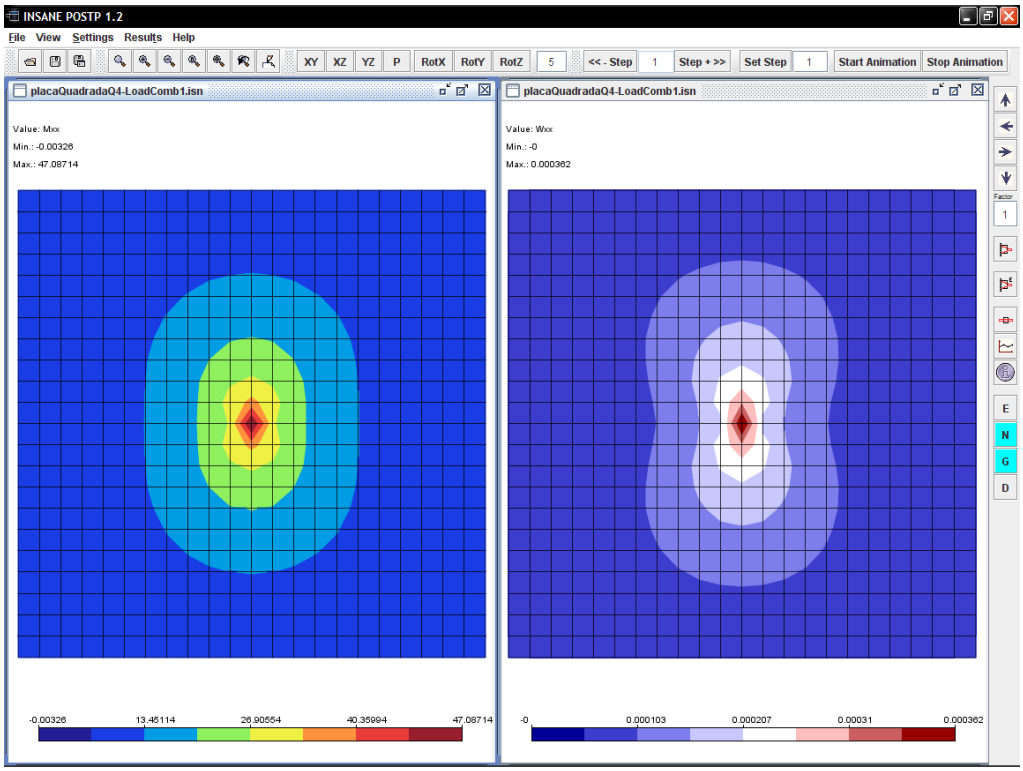


Figura 11.8: Momento e curvatura de flexão na direção x.

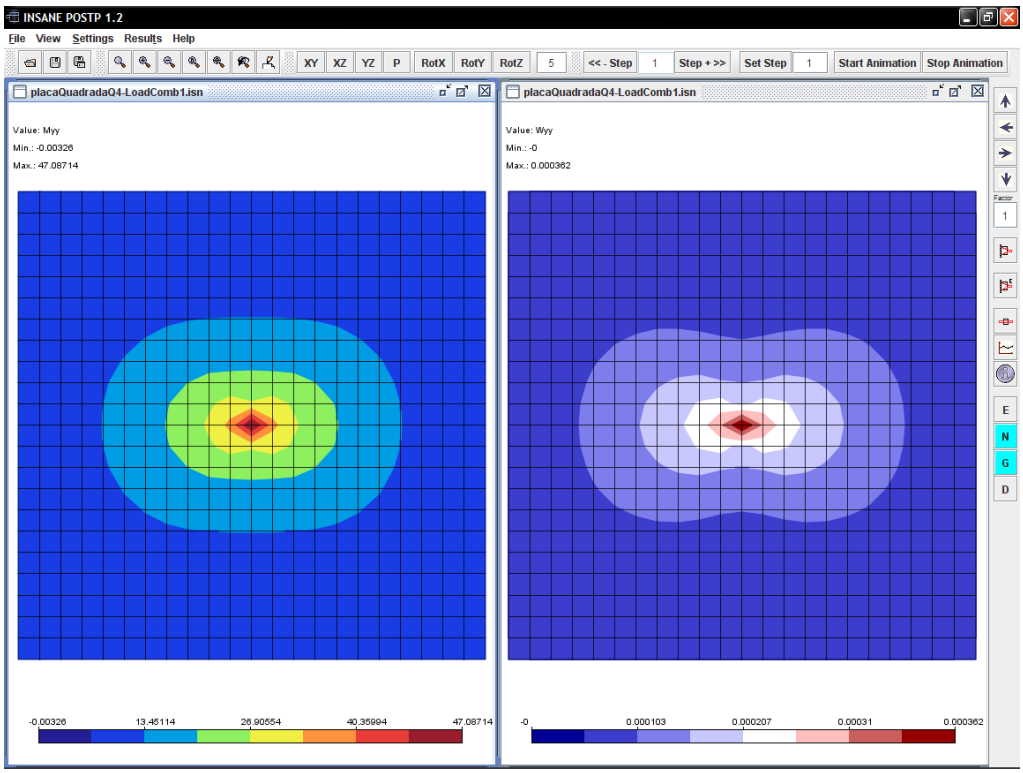


Figura 11.9: Momento e curvatura de flexão na direção y.

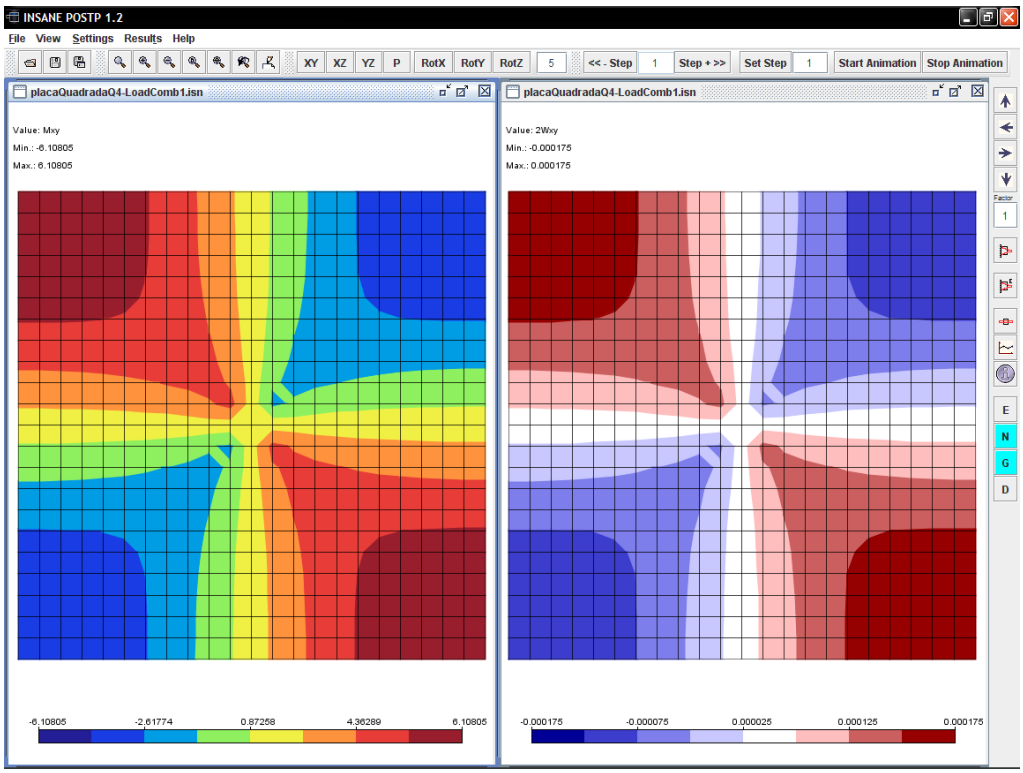


Figura 11.10: Momento e curvatura de torção.

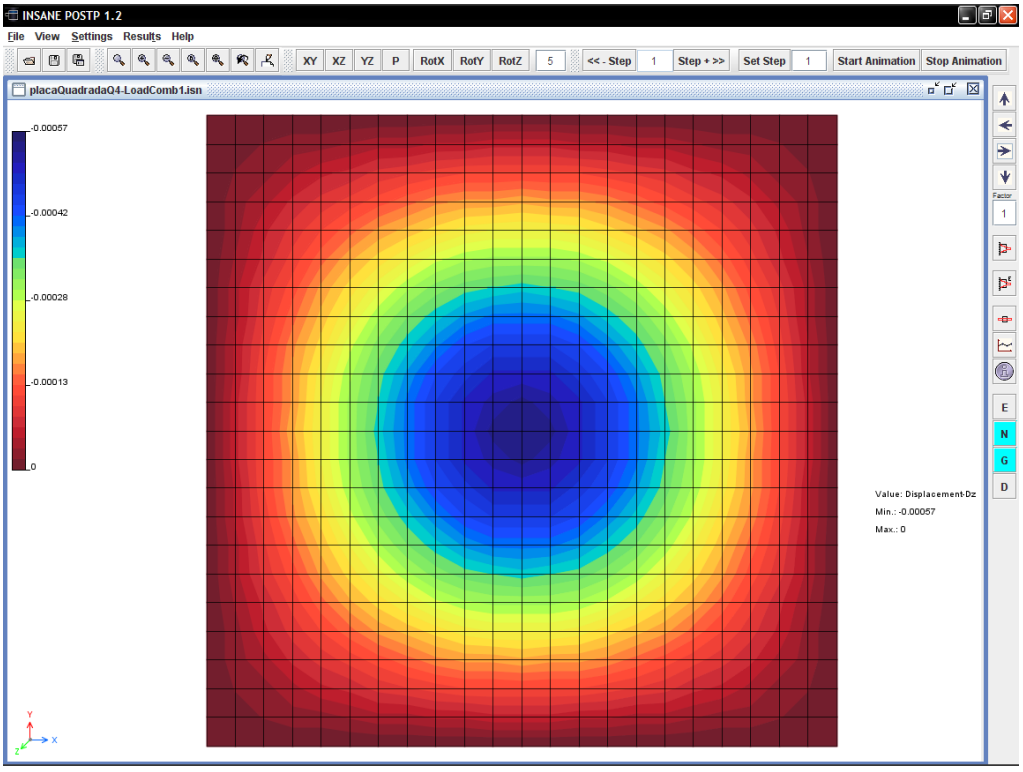


Figura 11.11: Variação das flechas.

11.3.2 Placa em Balanço com Elementos Triangulares

O modelo aqui visualizado é de uma placa com duas bordas engastadas e duas livres carregada que usa elementos finitos triangulares. A figura 11.12 mostra a malha de elementos finitos com as condições de contorno e carregamento.

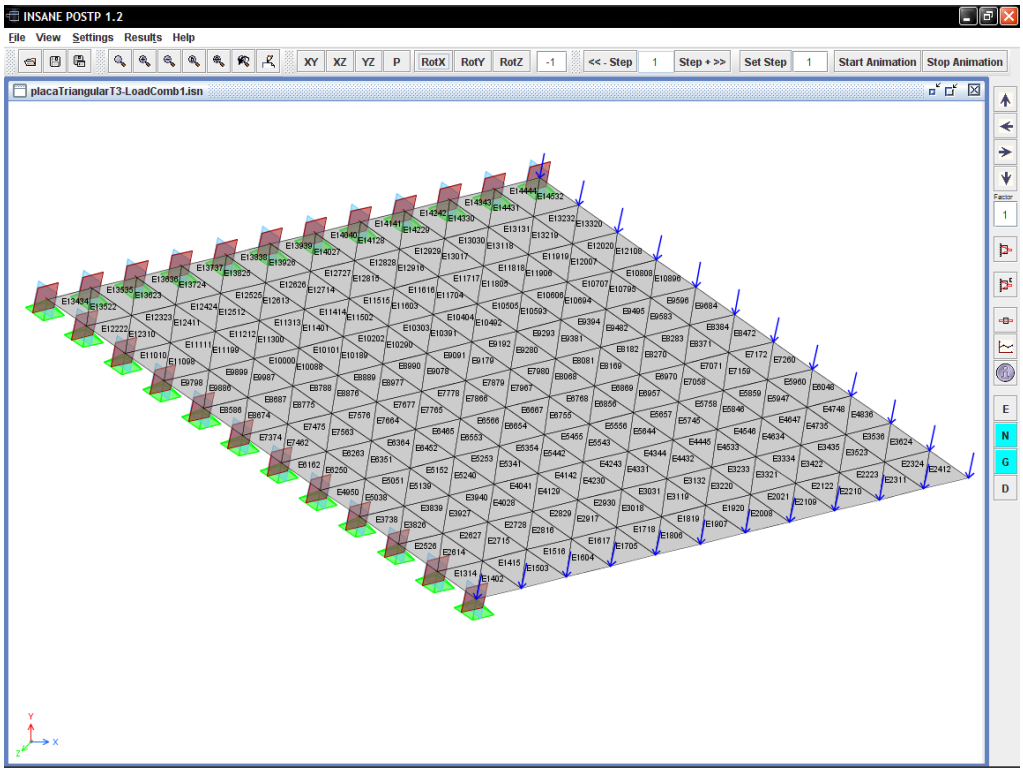


Figura 11.12: Placa em balanço composta por elementos triangulares.

O estado deformado da malha, mostrado na figura 11.13, descreve o efeito do engaste da placa fazendo com as rotações resultantes sejam nulas na respectiva borda e máxima no vértice livre da placa. As rotações resultantes foram obtidas a partir de uma a composição vetorial das rotações nas direções x e y . A figura também mostra as isofaixas para a variação das flechas.

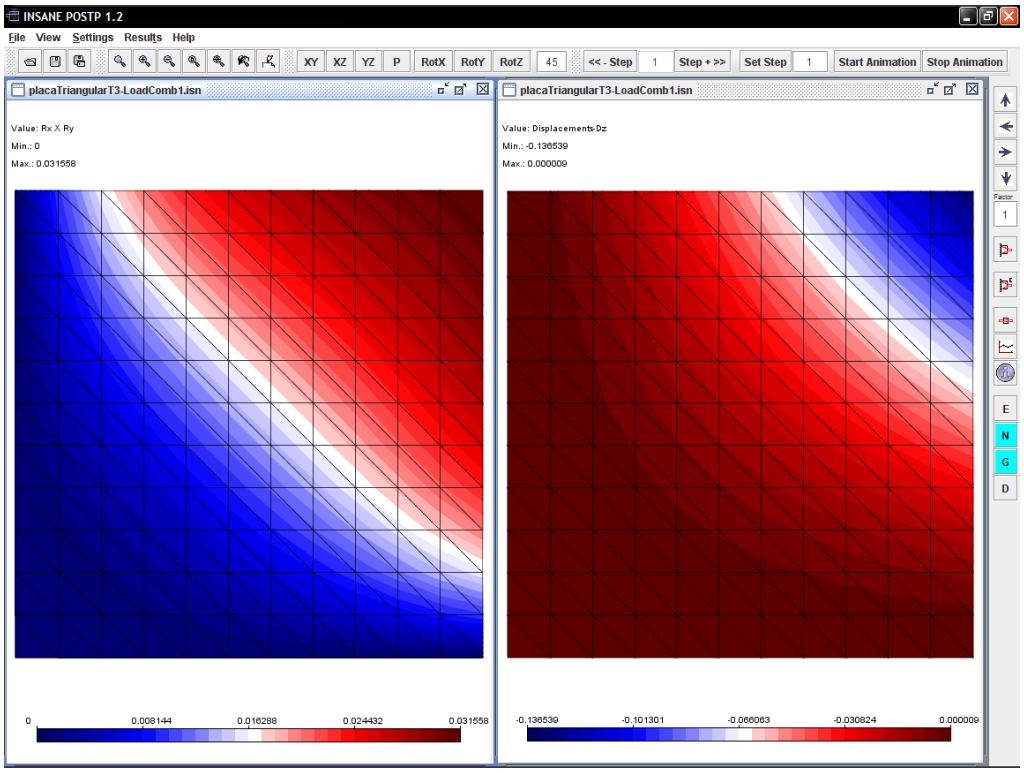


Figura 11.13: Rotações e deslocamentos nodais da placa.

11.3.3 Laje Cogumelo com Carregamento Distribuído

O modelo deste exemplo é uma laje cogumelo submetida a um carregamento distribuído, como mostra a figura 11.14. Na figura 11.15 é mostrada a variação das flechas da laje, exibidas na malha deformada. A figura 11.16 mostra a variação das rotações em relação aos eixos X (Rx) e Y (Ry). As figuras também ilustram os recursos do programa para a visualização, em diversas projeções, das condições de apoio e carga.

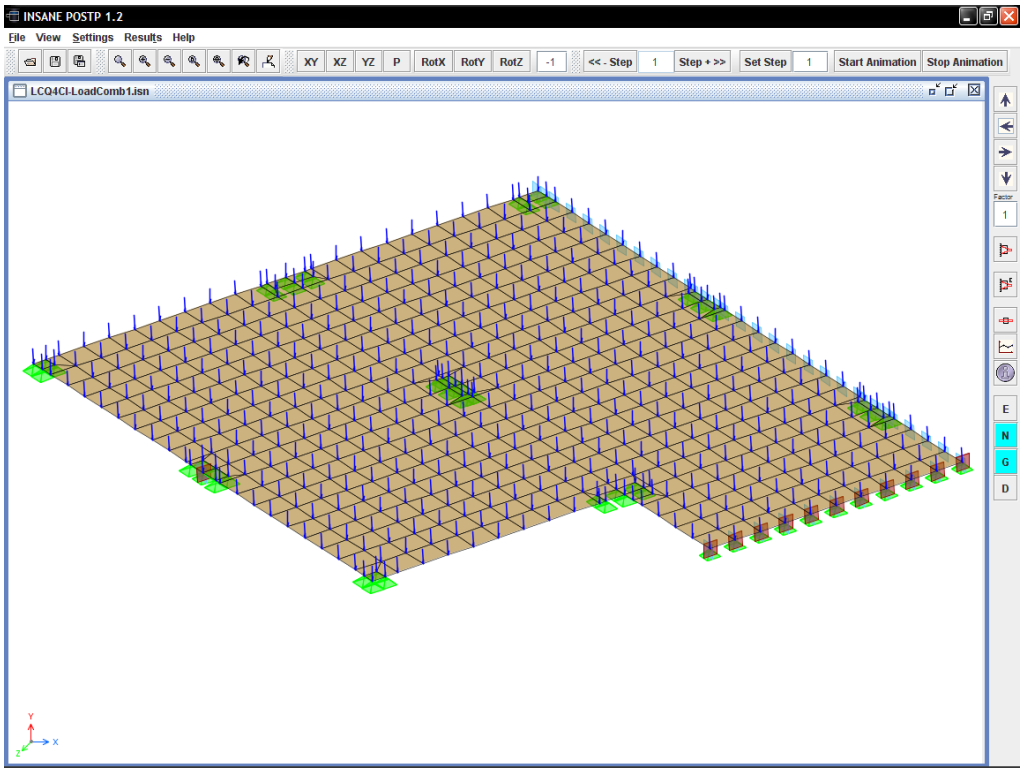


Figura 11.14: Laje cogumelo com carregamento distribuído.

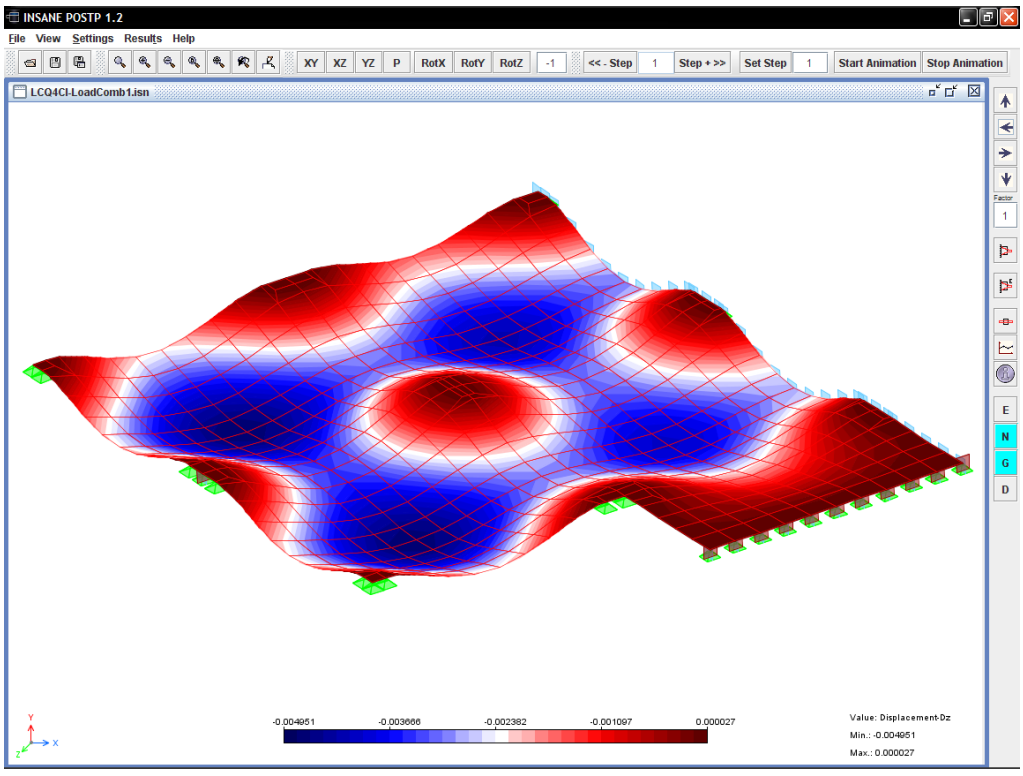


Figura 11.15: Variação das flechas na malha deformada.

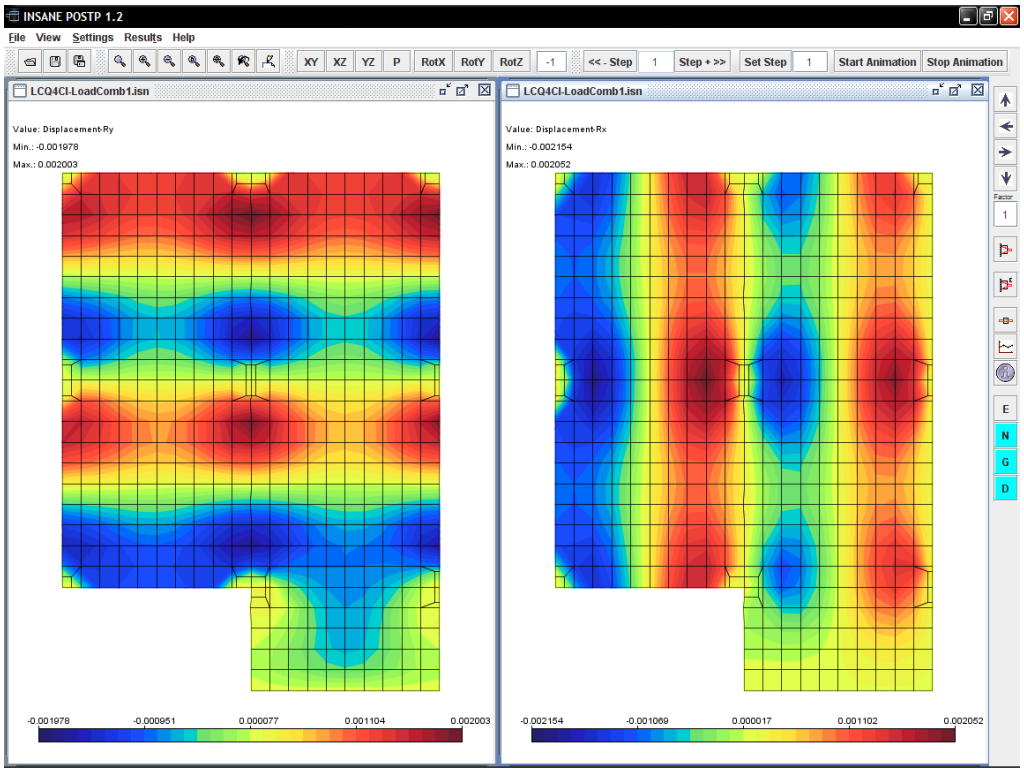


Figura 11.16: Variação das rotações Rx e Ry.

11.4 Modelos de Estado Plano de Tensão

11.4.1 Chapa Furada Submetida à Tração

Este exemplo apresenta a variação das tensões em uma chapa furada, submetida a tração mostrada na figura 11.17.

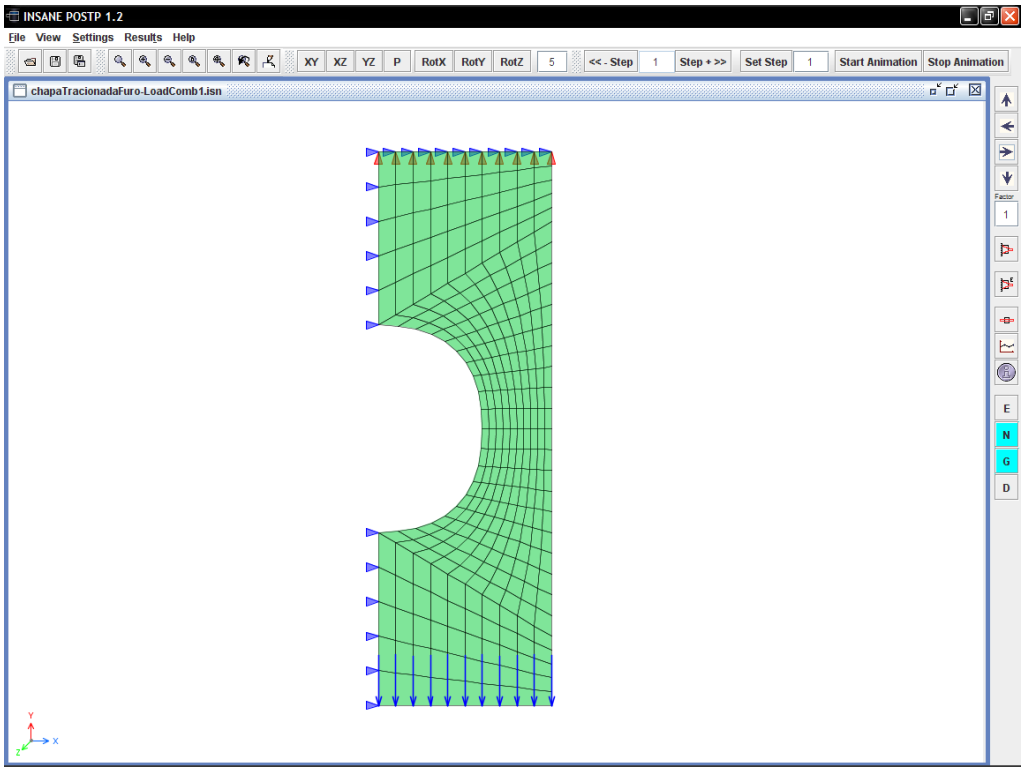


Figura 11.17: Chapa furada submetida à tração.

Em situações como estas as tensões na chapa, ao ser tracionada, se concentram nas regiões próximas ao furo, o que pode ser visto na figura 11.18. Verifica-se uma zona de estricção na chapa no trecho da furação como pode ser visto na figura 11.19, que mostra a variação dos deslocamentos nas direções x e y.

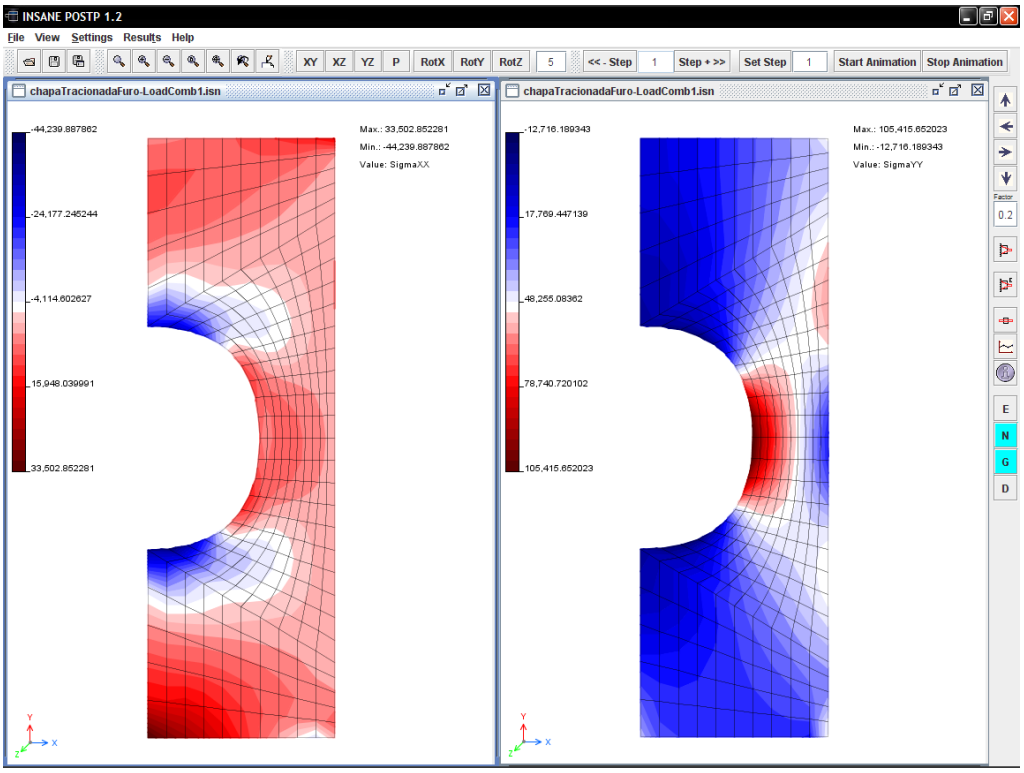


Figura 11.18: Concentração de tensões na chapa.

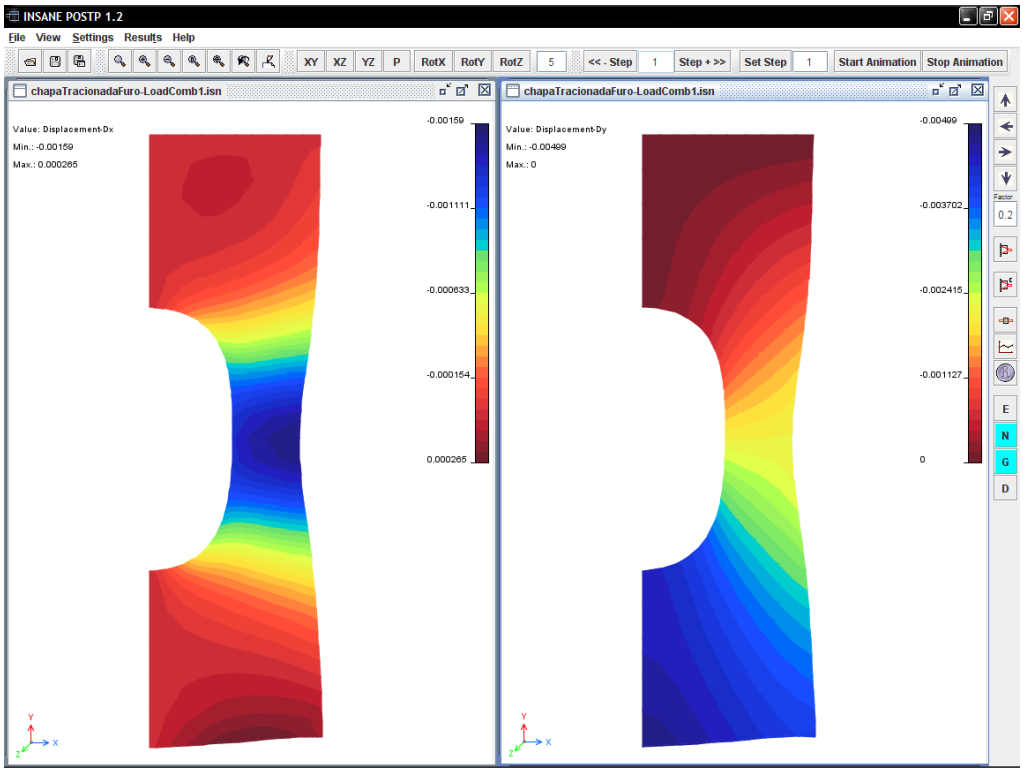


Figura 11.19: Zona de estricção na chapa.

11.4.2 Cantoneira com Chapa Fina

Este exemplo apresenta uma cantoneira submetida a um carregamento vertical e presa em uma extremidade (figura 11.20).

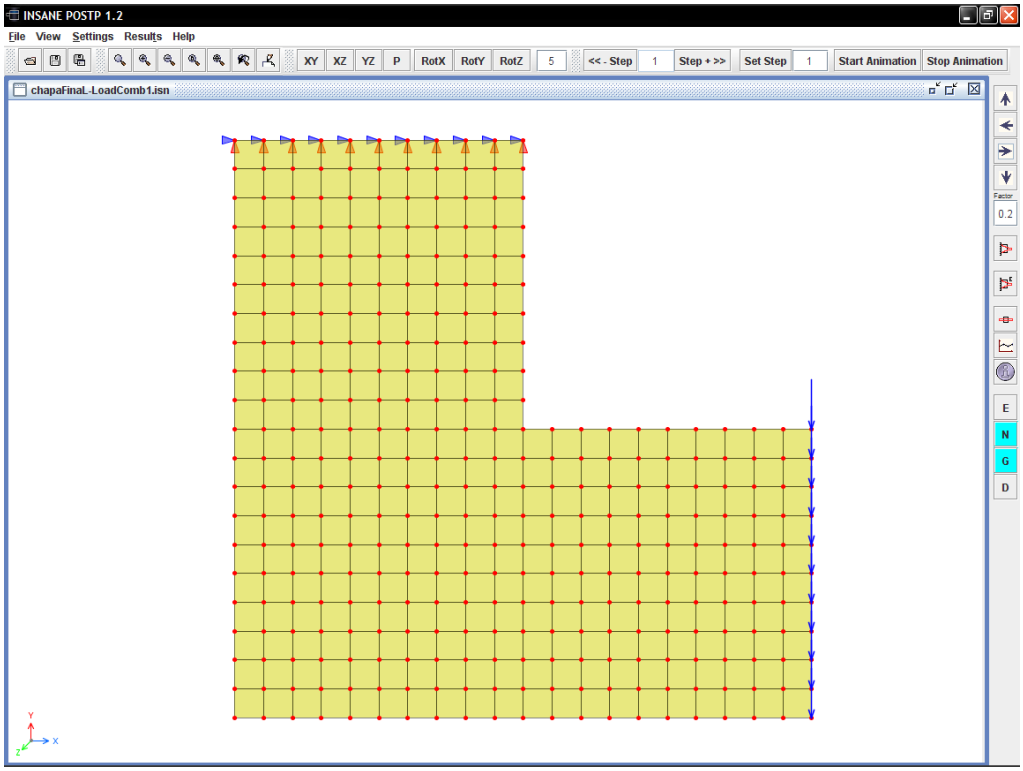


Figura 11.20: Modelo discreto de uma cantoneira.

O pós-processador exibe os detalhes da malha, como os identificadores dos nós e elementos (figura 11.21) e coordenadas nodais. As informações completas da discretização podem ser obtidas no arquivo XML do modelo ou através do dialogo **Information**, que exibe as informações nodais, de elementos ou dos pontos de integração. A figura 11.22 mostra o diálogo com informações dos elementos selecionados na malha.

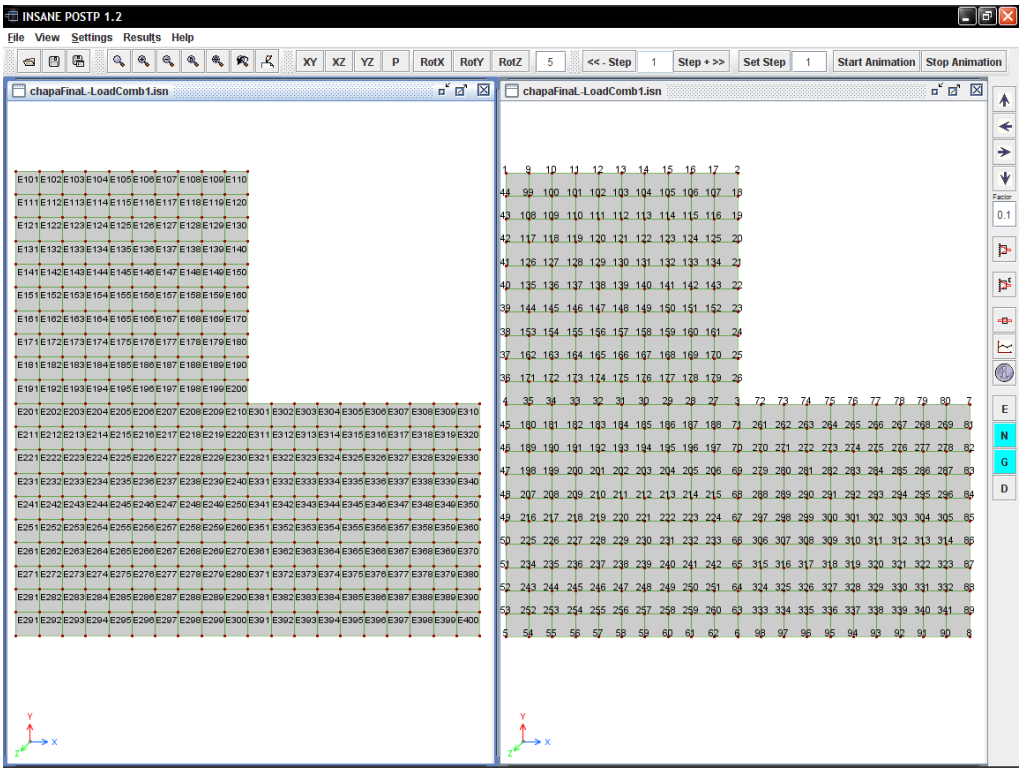


Figura 11.21: Detalhes da Malha.

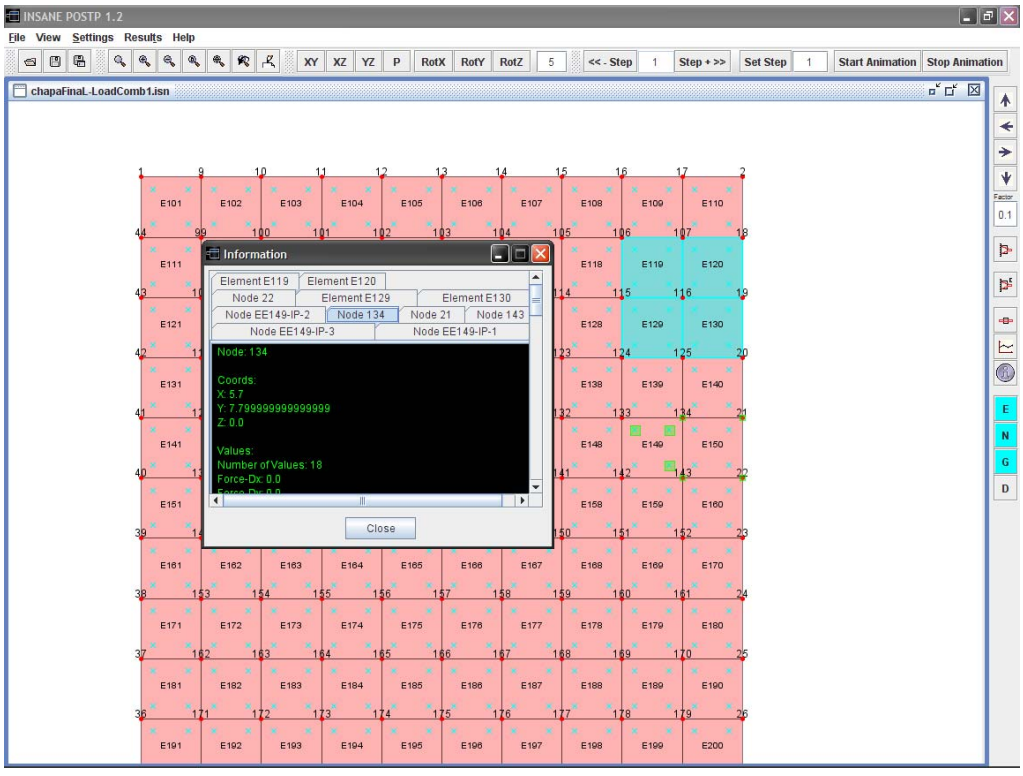


Figura 11.22: Diálogo para as informações do modelo.

Neste exemplo, o esforço crítico advém das tensões cisalhantes. Desta forma através dos resultados, pode-se fazer uma composição das tensões, em um tensor, para se obter o cisalhamento máximo, como ilustra a figura 11.23, que mostra também a variação das tensões τ_{xy}.

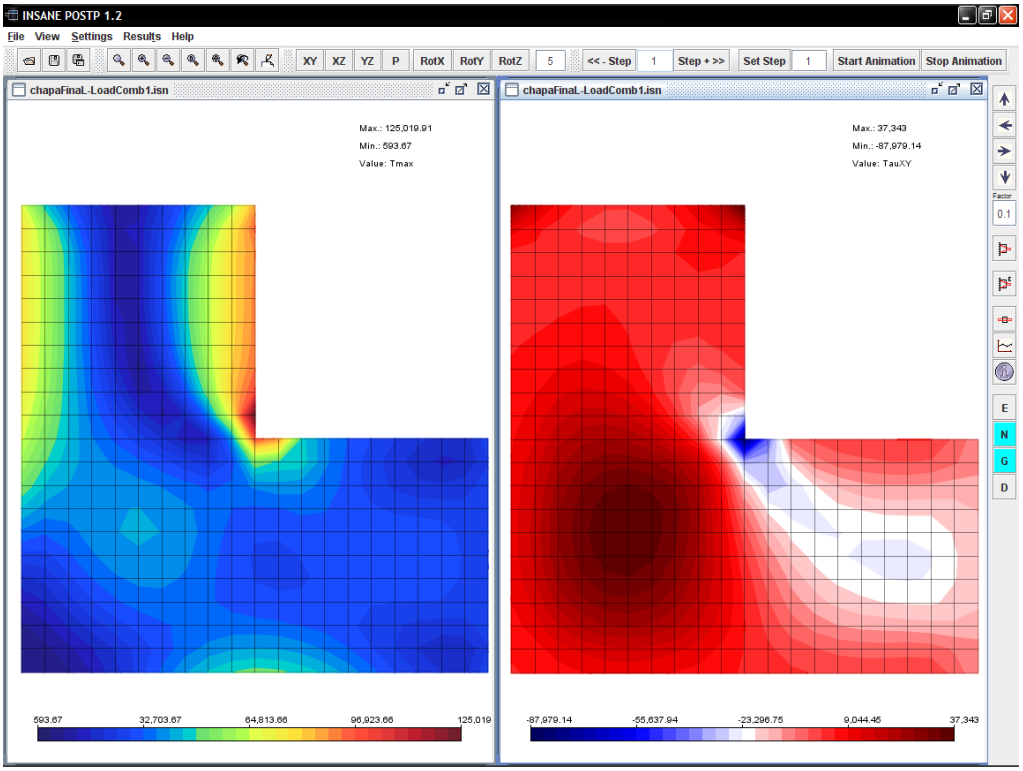


Figura 11.23: Tensões de cisalhamento máxima e τ_{xy}.

11.4.3 Viga Alta com Vários Modelos

Neste exemplo, visualiza-se uma viga alta submetida a três carregamentos diferentes. Analisa-se as cargas atuando separadamente na viga e depois em conjunto, como mostrado na figura 11.24.

Na figura 11.25 são vistas as deformadas respectivas a cada caso de carga, e na figura 11.26 tem-se resultados para tensões (no caso σ_{xx}).

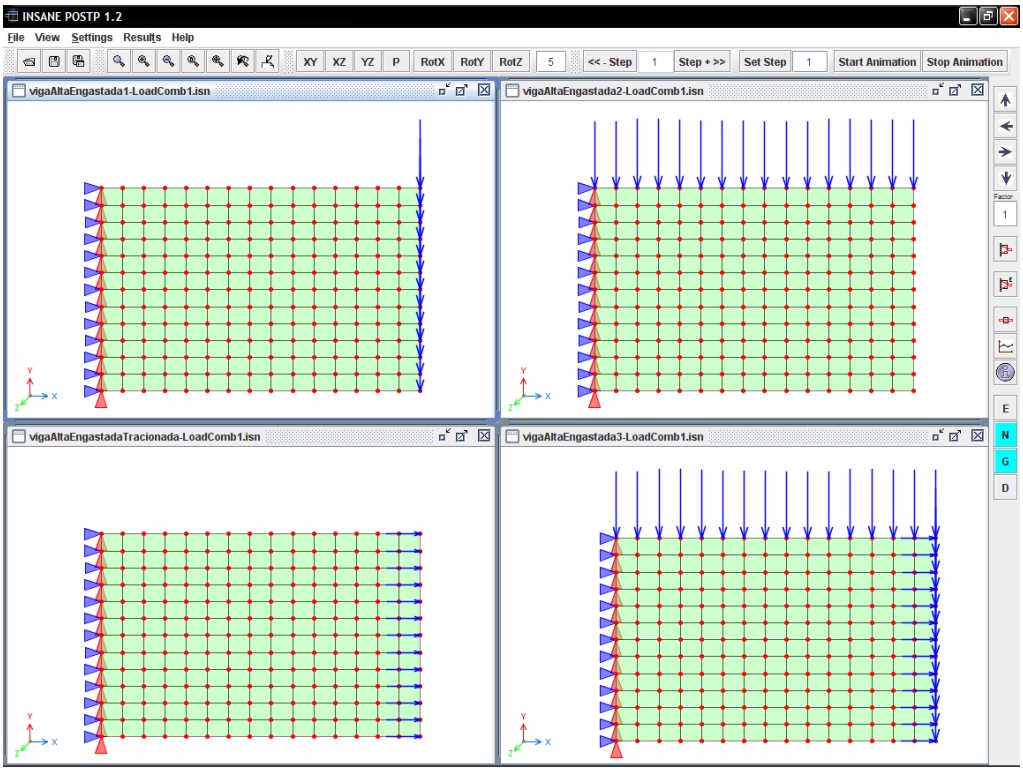


Figura 11.24: Combinação das cargas do modelo.

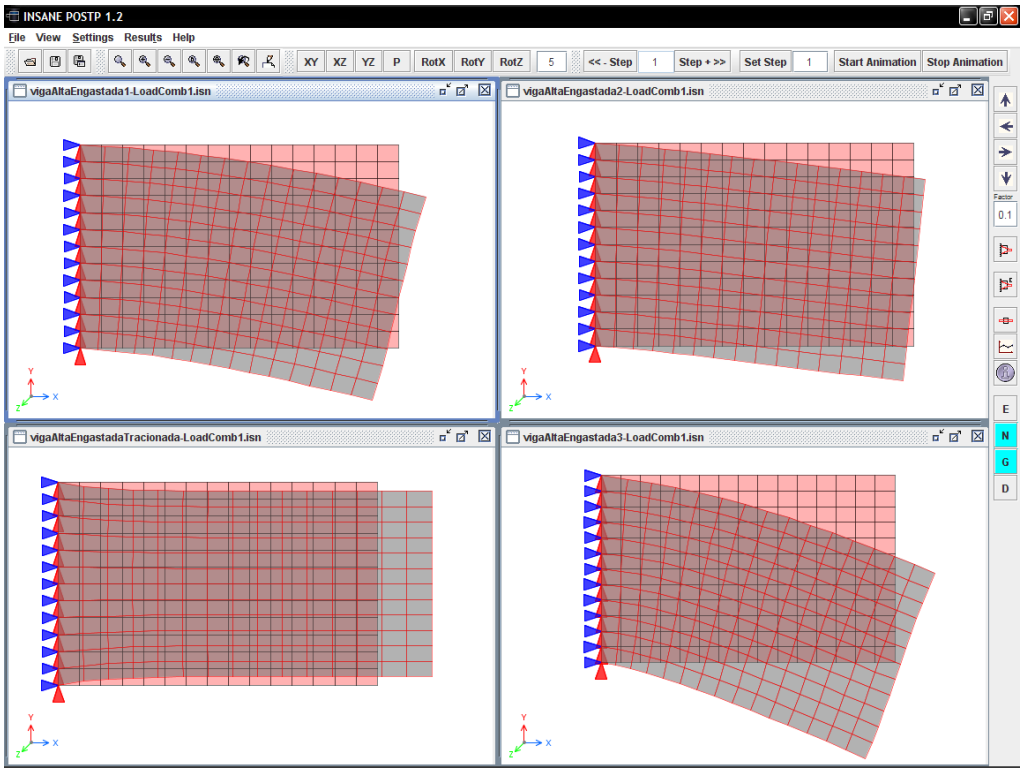


Figura 11.25: Deformadas do Modelo devido aos quatro casos de carga.

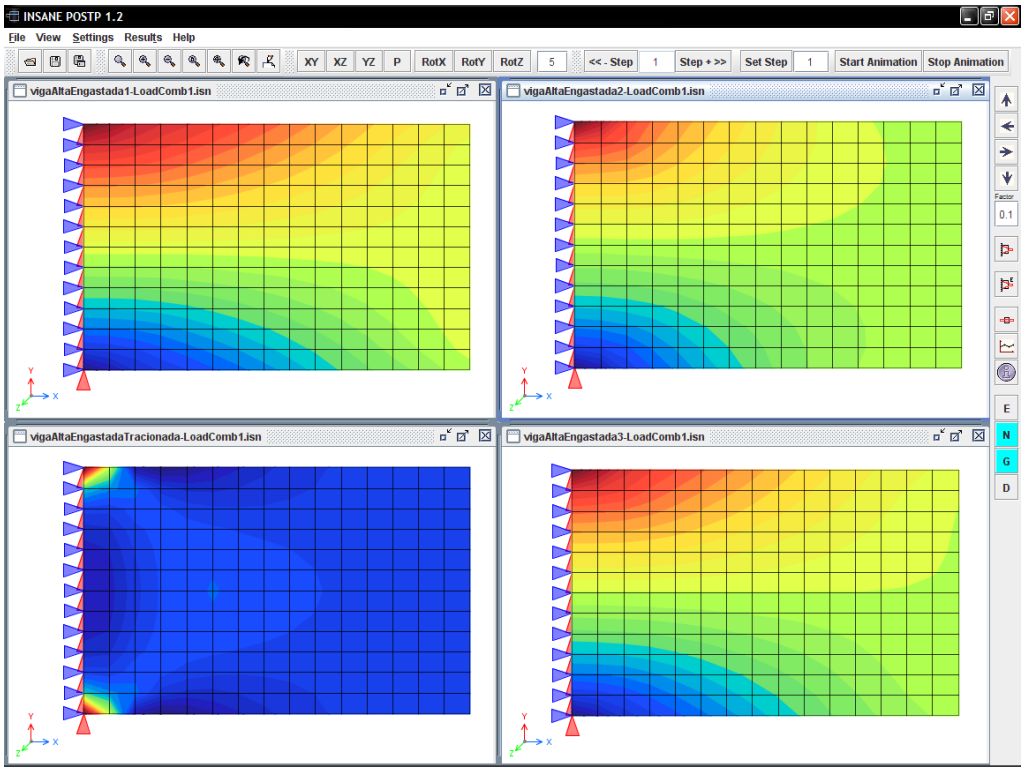


Figura 11.26: Resultados de Tensões.

11.5 Modelos de Estado Plano de Deformação

11.5.1 Maciço de Concreto

A interferência da malha do modelo na representação dos resultados pode ser vista neste exemplo. Um maciço de concreto com uma carga pontual, foi modelado com o mesmo número de elementos, porém de tipos diferentes (figura 11.27): um modelo com elementos quadrilaterais de quatro nós e o outro com elementos quadrilaterais de oito nós.

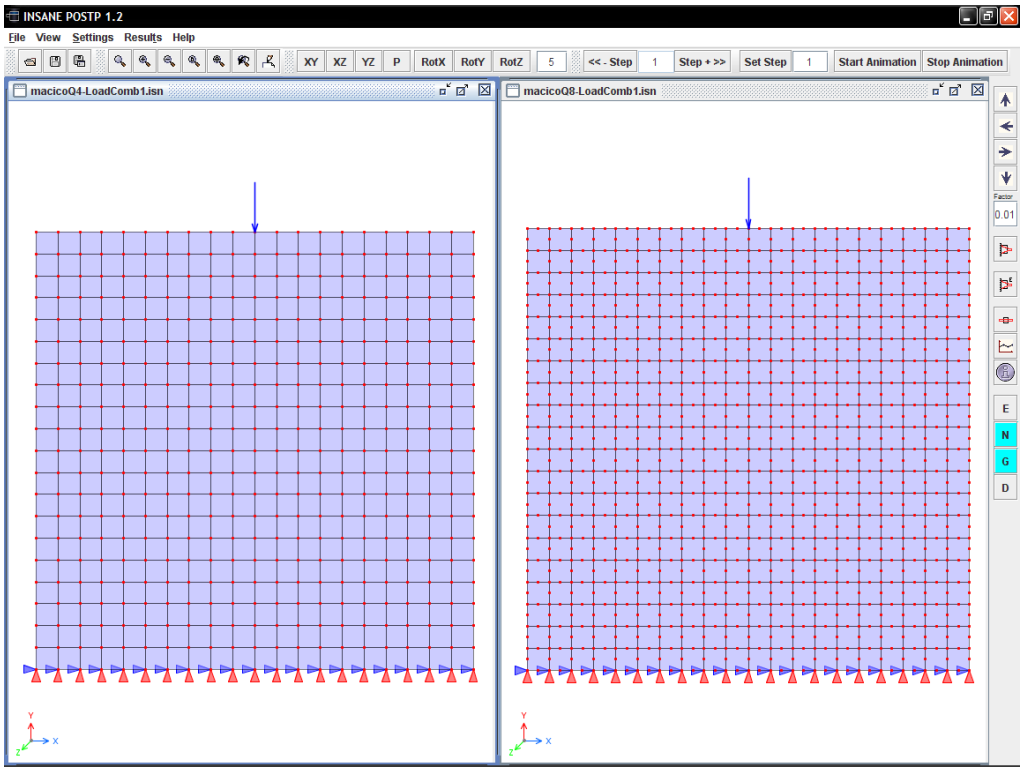


Figura 11.27: Modelo do maciço de concreto.

Avaliando os resultados, pode-se ver claramente, nas faixas de valores para as tensões σ_{xx} (figura 11.28) e σ_{yy} (figura11.29), que o comportamento das tensões são semelhantes mas que a representação na malha de elementos de oito nós tem-se um maior detalhamento, uma vez que, do ponto de vista do pós-processamento, dispõe-se de uma maior quantidade de valores para o traçado das isofaixas.

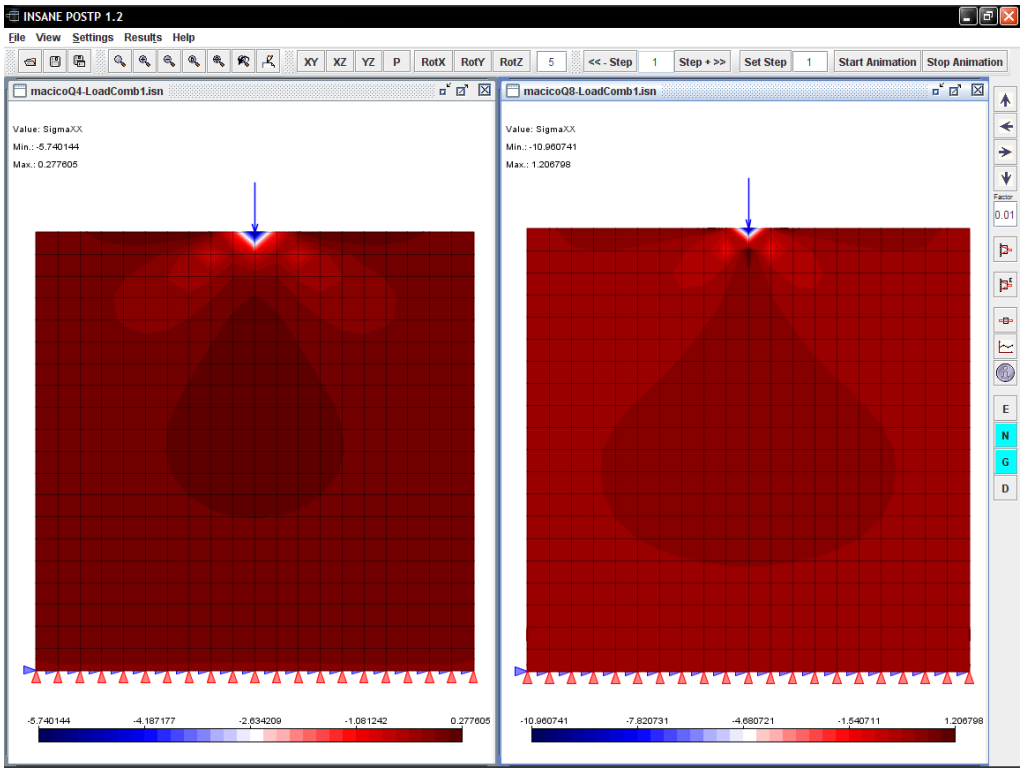


Figura 11.28: Tensões σ_{xx}.

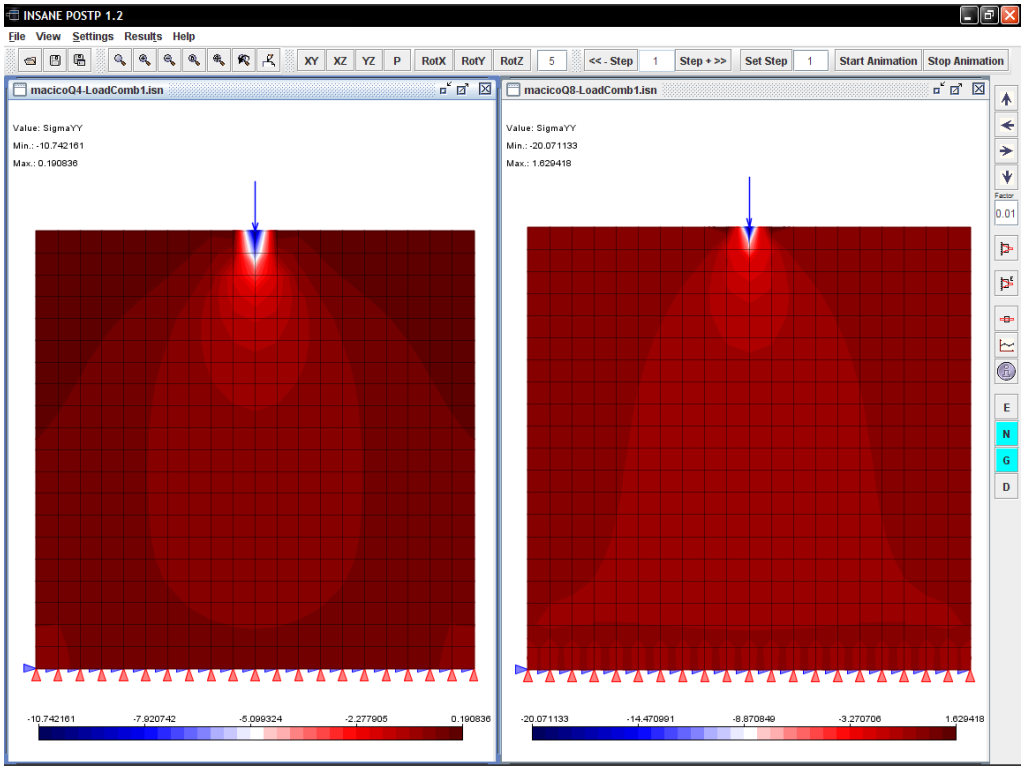


Figura 11.29: Tensões σ_{yy}.

11.5.2 Muro de Arrimo

Neste exemplo, a seção transversal de um muro de arrimo foi modelada com elementos finitos de estado plano de deformação e analisada para 25 passos de carga. A figura 11.31 mostra as isofaixas para a variação da deformação ε_{xx} em diferentes passos da análise. Pode-se ver também o deslocamento do modelo ao longo do processamento.

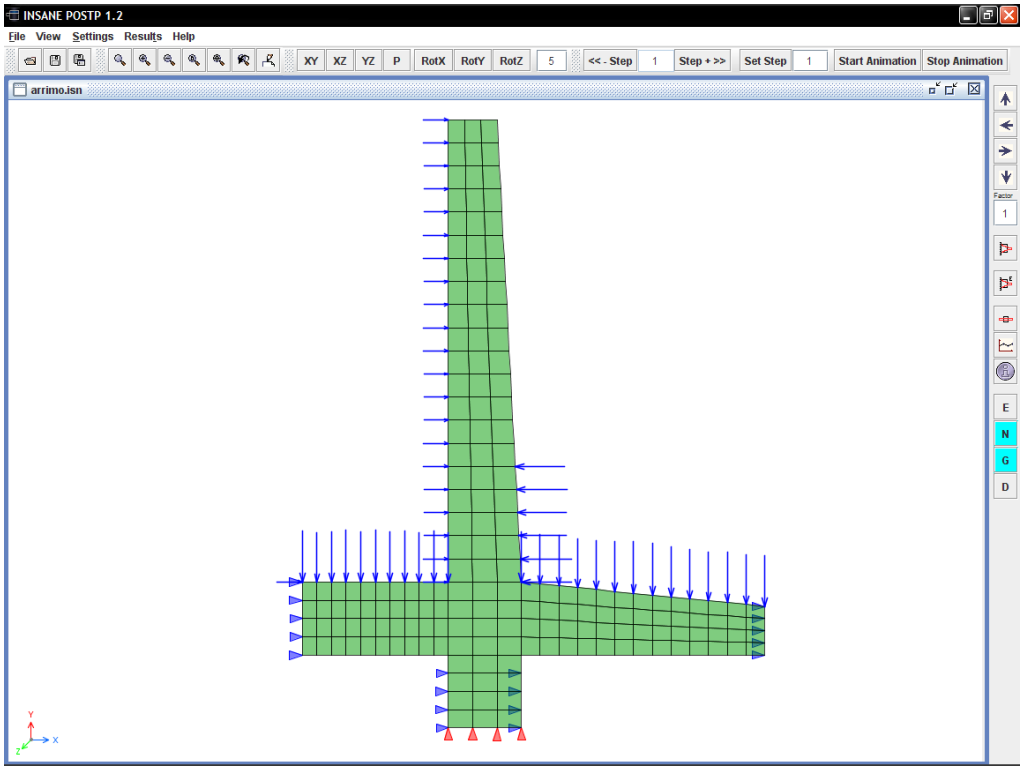


Figura 11.30: Modelo discreto do muro de arrimo.

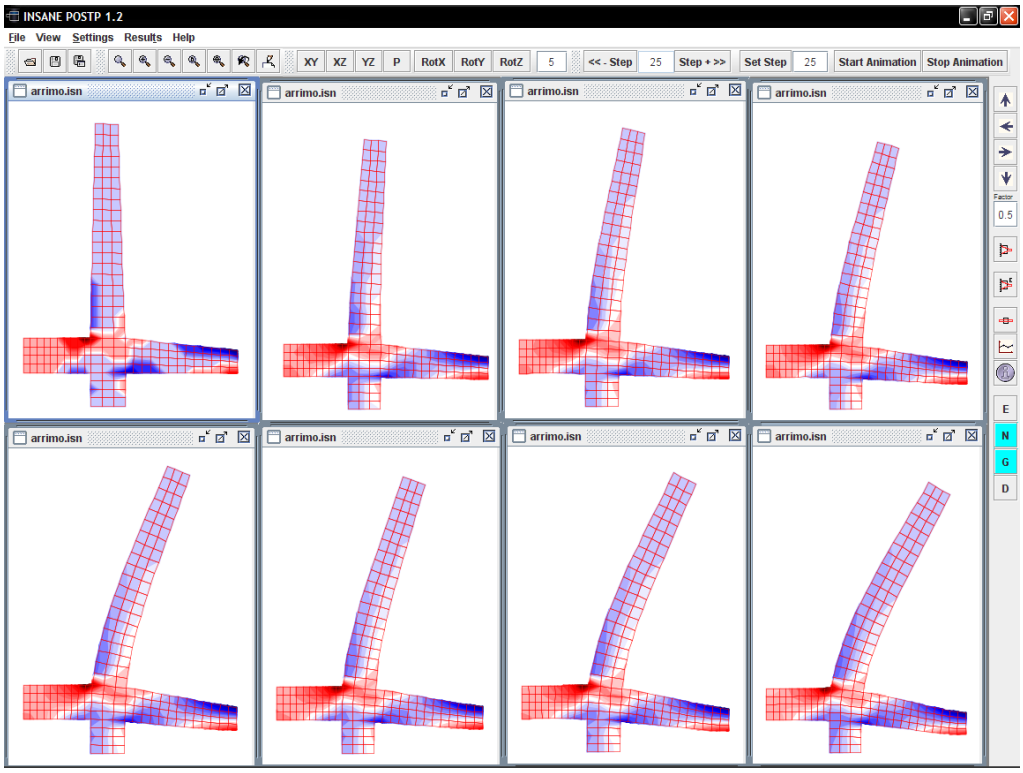


Figura 11.31: Variação da deformações ε_{xx}.

Capítulo 12

CONSIDERAÇÕES FINAIS

Consoante com os objetivos gerais do projeto **INSANE**, este trabalho contribui sobremaneira para a difusão da filosofia de software livre, uma vez que resultou em mais um aplicativo de código aberto que pode ser usado para qualquer propósito, modificando-o para adaptá-lo a necessidades particulares, ter cópias modificadas distribuídas de forma gratuita, beneficiando a comunidade com as melhorias.

Integrado ao sistema **INSANE**, a realização deste trabalho se baseou em um aplicativo computacional segmentado, que viabilizou expansões e mudanças, sem a necessidade de reescrever o código. Para tanto, o uso de padrões de projeto de software, linguagem de marcação XML e o paradigma de programação orientado a objetos foram fundamentais.

12.1 Contribuições deste Trabalho

O programa desenvolvido disponibilizou no **INSANE** um aplicativo gráfico interativo para o pós-processamento de modelos bidimensionais, capaz de representar resultados de análises lineares e não-lineares.

O desenvolvimento do pós-processador ampliou o uso do padrão MVC, conjugando-o à múltiplas vistas e possibilitando a interação entre as mesmas. Esta ampliação introduziu um novo formato para a interface com o usuário do **INSANE** baseando-se em um **Desktop** com múltiplas janelas, cada uma associada a um par vista-controlador.

O uso do padrão *Observer* associado à múltiplas "*threads*" de processamento explorou recursos de sincronismo entre o núcleo numérico e o pós-processamento. Tal padrão, integrado ao MVC, associou o sincronismo às diversas vistas e, fazendo uso da independência entre as aplicações, resultados diferentes podem agora ser visualizados simultaneamente.

A independência do pós-processador foi possível devido à implementação da estrutura de dados de semi-arestas que permitiu tratar modelos de elementos finitos de forma genérica, baseando-se em informações geométricas. A implementação da estrutura de semi-arestas também pode ser aplicada a modelos unidimensionais e tridimensionais.

As implementações de algoritmos de geometria computacional, triangulação de Delaunay, transformações geométricas e projeções, entre outras, podem ser utilizadas nos demais módulos do programa.

12.2 Sugestão para Trabalhos Futuros

A seguir são apresentadas algumas sugestões para trabalhos futuros.

1. Implementação de novos recursos para a representação de resultados, como por exemplo o desenho de isocurvas.

2. A expansão do pós-processador para a representação de modelos tridimensionais.

3. Integração ao pós-processador de modelos unidimensionais.

4. Implementação de um pré-processador que faça uso da triangulação de Delaunay para refinamento da malha.

5. Aprimoramento do aplicativo gráfico possibilitando a criação de gráficos para a variação de grandezas não nodais.

6. Aprimoramento dos algoritmos de geração e manipulação das imagens de modo a se obter melhor desempenho.

Apêndice A

Núcleo Numérico

O núcleo numérico do INSANE é composto por um conjunto de classes e interfaces que visam tratar de forma genérica modelos discretos, independente da natureza do problema. As interfaces **Model**, **Solution**, **Assembler** e **Persistence** são as mais importantes do conjunto (figura A.1).

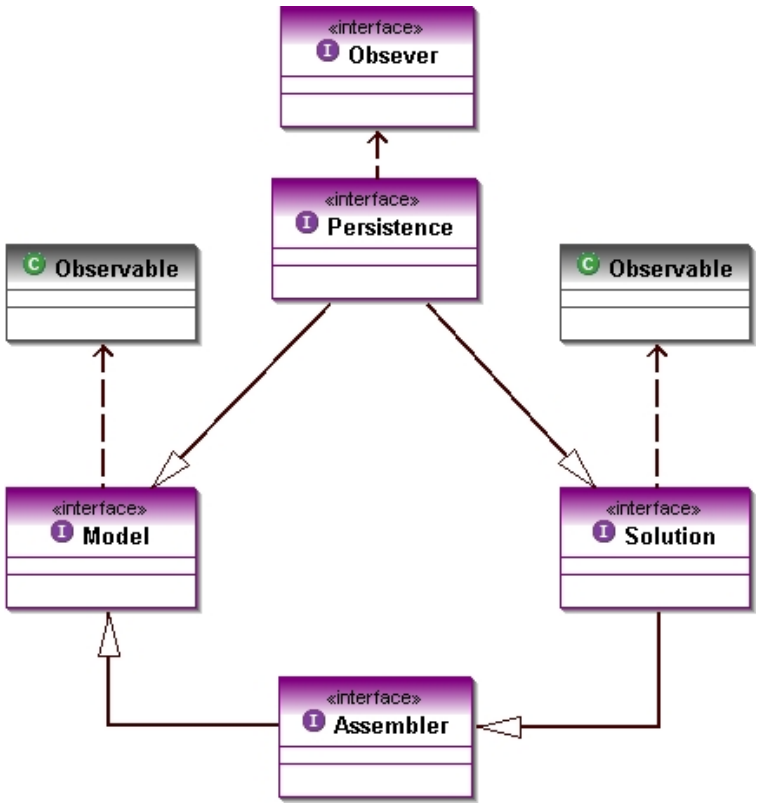


Figura A.1: Organização do núcleo numérico

A partir desta organização, é possível compor um modelo por meio de dados

persistidos para determinado problema, montar as equações do modelo, solucionar o modelo e armazenar os resultados para posterior análise.

A interface **Persistence**, padroniza a composição dos modelos através de arquivos texto ou binários. A interface **Model** representa os modelos a serem solucionados, e as implementações do mesmo possuem os dados que compõem o problema. A interface **Assembler** pode ser implementada para vários tipos de problemas, uma vez que tem como finalidade montar o sistema de equações do problema, buscando no modelo os dados necessários. Já a interface **Solution** define os métodos para solucionar as equações dos possíveis modelos.

As classes que implementam a interface **Model** representam os modelos a serem resolvidos. Como ilustra a figura A.2, pode-se ter, por exemplo, um modelo de elementos finitos ou qualquer outro tipo de modelo.

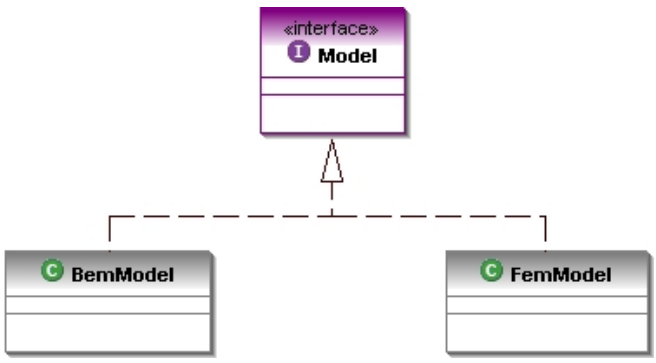


Figura A.2: Interface **Model**

Na figura A.3, tem-se a interface **Solution**, com classes que implementam possíveis soluções. No caso de uma solução para a obtenção da trajetória equilíbrio (*EquilibriumPath* como mostrado) deve-se ainda definir a estratégia de iteração.

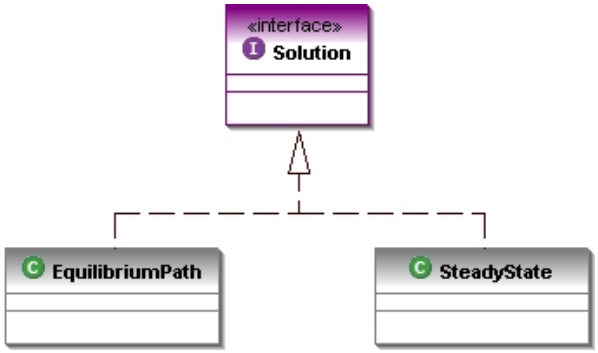


Figura A.3: Interface **Solution**

A interface **Shape** define as classes das funções de forma dos elementos usados na modelagem, como mostra a figura A.4. Estes elementos são definidos pela classe abstrata **Element** cuja hierarquia (figura A.5) apresenta diferentes tipos de elementos. Estas duas classes são de extrema importância para o pós-processador, visto que as mesmas definem a geometria dos elementos, o número de nós e as funções de aproximação. Esta última característica pode ser usadas em alguns métodos de pós-processamento.

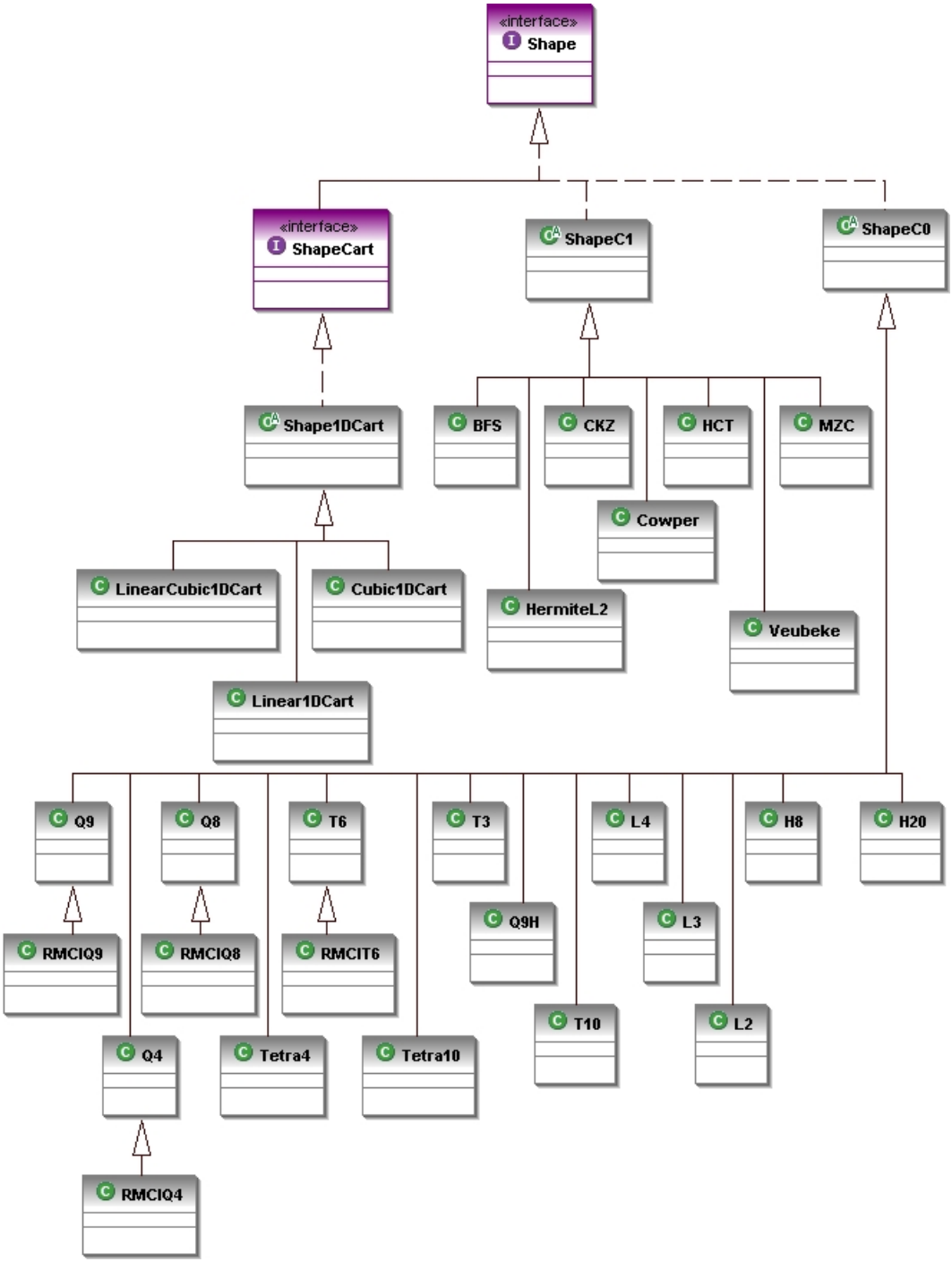


Figura A.4: Interface **Shape**

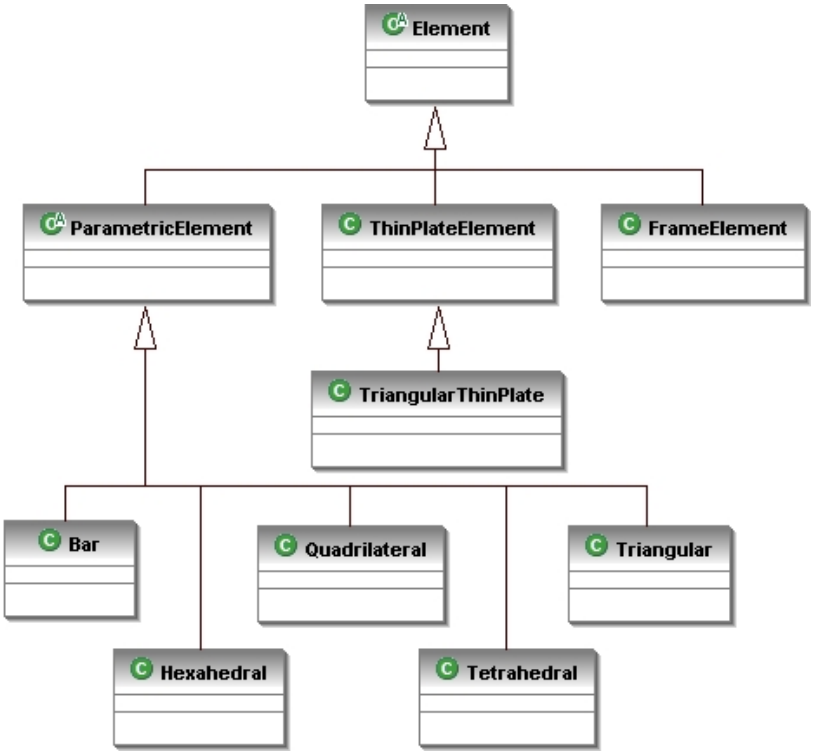


Figura A.5: Classe **Element**

As classes implementadas se relacionam durante o processamento dos diversos modelos através de instâncias. Estas instâncias são características de cada classe e são usadas na composição do problema, bem como em sua solução.

O modelo de elementos finitos é composto por dados dependentes de outras classes. Na figura A.6, são vistas as classes usadas pela classe **FemModel**. Destaca-se a interface **ProblemDriver** que guarda as informações necessárias para que as classes que a implementam descrevam as características para resolver cada modelo. Destaca-se também as classes **Loading**, que descreve o carregamento de cada modelo e **LoadCombination**, que dá a possibilidade de combinação entre os carregamentos.

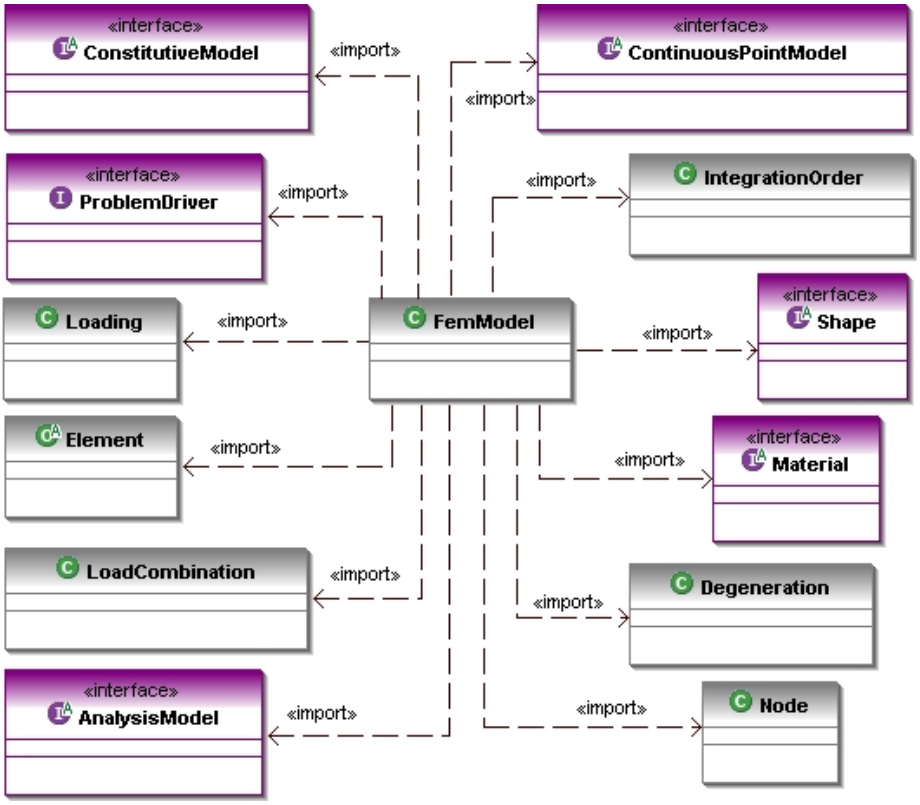


Figura A.6: Dependências de **FemModel**

A classe **FemModel**, como visto na figura A.7, possui instâncias relativas à composição de um modelo de elementos finitos. Desta forma, um modelo possui uma lista de nós, elementos, degenerações, funções de forma diferentes, diferentes ordens de integração, composto por materiais diferentes e, portanto, modelos constitutivos diferentes, bem como por modelos de análise diferentes.

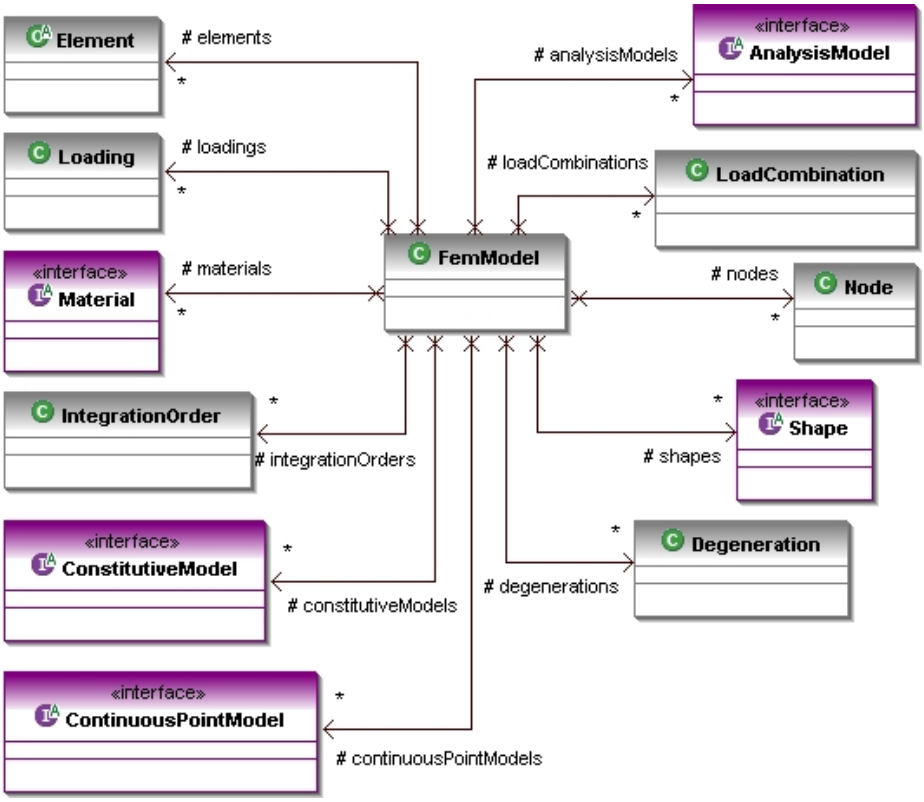


Figura A.7: Instâncias de **FemModel**

Os elementos que compõem o modelo, são representados pela classe **Element**, e suas dependências são mostradas na figura A.8. Verifica-se características típicas de um elemento como funções de forma, ordem de integração, modelo constitutivo, material, sua respectiva degeneração, os nós que compõe o elemento.

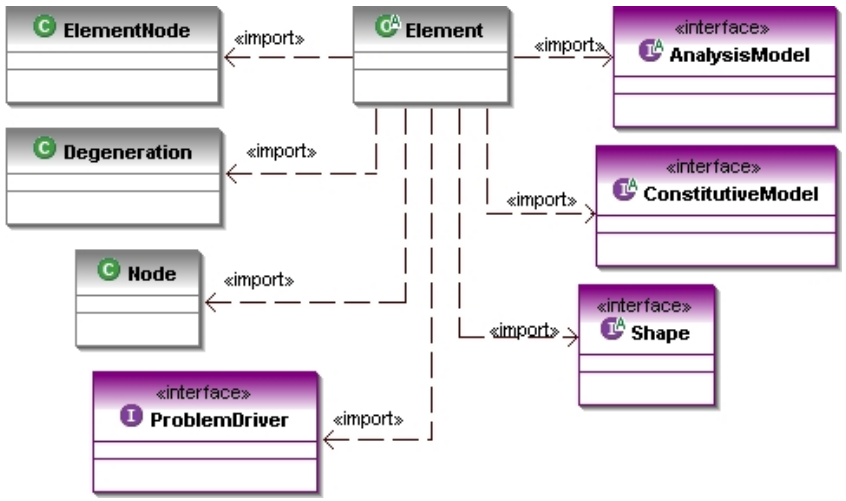


Figura A.8: Dependências de **Element**

Apêndice B

Definição de Dados e Arquivos XML do Pós-Processador

Na seção 10.9 foram mostrados exemplos carregados a partir de arquivos XML independentes do processamento. Para tanto foram preenchidos arquivos na definição de dados (DTD ou *Document Type Definition*) mostrada nas figuras B.1 e B.2.

O arquivo de entrada de dados do exemplo de pórtico plano apresentado na seção 10.9 é mostrado nas figuras B.3 e B.4. O pós-processador possui funcionalidades para modelos bidimensionais, portanto modelos diferentes não podem exibir todos os resultados. Resultados relativos à geometria do modelo (como por exemplo a deformada de uma estrutura) podem ser representadas pois o modelo geométrico baseia-se em uma estrutura de semi-arestas.

O exemplo bidimensional mostrado na seção 10.9 foi obtido pelo preenchimento de um arquivo XML, de modo que seus valores foram colocados de forma aleatória no o intuito de ilustrar um modelo bidimensional qualquer e que as funcionalidades do pós-processador só dependem dos dados de entrada do arquivo preenchido. O arquivo XML do respectivo exemplo pode ser visto nas figuras B.5 e B.6.

```
<!--
* INSANE - Interactive Structural Analysis Environment
* Copyright (C) 2003-2005
* Universidade Federal de Minas Gerais
* Escola de Engenharia
* Departamento de Engenharia de Estruturas
*
* Author's email :    insane@dees.ufmg.br
* Author's website :  http://www.dees.ufmg.br/insane
*
* This program is free software; you can redistribute it and/or
* modify it under the terms of the GNU General Public License
* as published by the Free Software Foundation; either version 2
* of the License, or any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307,
USA.
-->

<!-- The postp contain: a Step -->
<!ELEMENT PostpData (Step)>

<!-- The Step contains: GeoPostpModelList and StepAttributes-->
<!ELEMENT Step (GeoPostpModelList , StepAttributes *)>
<!ATTLIST Step label CDATA #REQUIRED >

<!-- The GeoPostpModelList contains: GeoPostpModel -->
<!ELEMENT GeoPostpModelList  (GeoPostpModel *)>

<!-- StepAttributes contains data -->
<!ELEMENT StepAttributes  (#PCDATA )>

<!-- The GeoPostpModel contains: PointsList, PatchesList, BoundariesList -
<!ELEMENT GeoPostpModel  (PointsList , PatchesList *, BoundariesList *)>
<!ATTLIST GeoPostpModel  label CDATA #REQUIRED >

<!-- The PointsList contains many PointModels -->
<!ELEMENT PointsList  (PointModel )+>

<!-- The PatchesList contains many Patches -->
<!ELEMENT PatchesList  (Patch )*>

<!-- The BoundariesList contains many Boundary -->
<!ELEMENT BoundariesList  (Boundary )*>
```

Figura B.1: Arquivo DTD para "XML" do pós-processador (1^a parte).

```
<!-- The PointModel contains its coord-->
<!-- It may contain: many Key/Values -->
<!ELEMENT PointModel (Coord, Value*)>
<!ATTLIST PointModel label CDATA #REQUIRED>

<!-- coord contains data -->
<!ELEMENT Coord (#PCDATA)>

<!-- values contains data -->
<!ELEMENT Value (#PCDATA)>

<!-- The Patch contains its label-->
<!ELEMENT Patch (PatchPointsLabels)>
<!ATTLIST Patch label CDATA #REQUIRED>

<!-- values contains data -->
<!ELEMENT PatchPointsLabels (#PCDATA)>

<!-- The Patch contains its label-->
<!ELEMENT Boundary (BoundLabel, BoundPointsLabels)>
<!ATTLIST Boundary label CDATA #REQUIRED>

<!-- values contains data -->
<!ELEMENT BoundPointsLabels (#PCDATA)>
```

Figura B.2: Arquivo DTD para "XML" do pós-processador (2^a parte).

```
<?xml version="1.0" encoding="ISO8859-1" ?>
<!DOCTYPE PostpData SYSTEM "postp.dtd">
<PostpData>
    <Step label="single">
        <GeoPostpModelList>
            <GeoPostpModel label="Nodal">
                <PointsList>
                    <PointModel label="1">
                        <Coord>0 0 0</Coord>
                        <Value>RESTRAINTS -Dx true</Value>
                        <Value>RESTRAINTS -Dy true</Value>
                    </PointModel>
                    <PointModel label="2">
                        <Coord>0 3 0</Coord>
                    </PointModel>
                    <PointModel label="3">
                        <Coord>0 6 0</Coord>
                        <Value>Force-Dy -1000</Value>
                    </PointModel>
                    <PointModel label="4">
                        <Coord>5 0 0</Coord>
                        <Value>RESTRAINTS -Dx true</Value>
                        <Value>RESTRAINTS -Dy true</Value>
                    </PointModel>
                    <PointModel label="5">
                        <Coord>5 3 0</Coord>
                    </PointModel>
                    <PointModel label="6">
                        <Coord>5 6 0</Coord>
                        <Value>Force-Dy -1000</Value>
                    </PointModel>
                    <PointModel label="7">
                        <Coord>10 0 0</Coord>
                        <Value>RESTRAINTS -Dx true</Value>
                        <Value>RESTRAINTS -Dy true</Value>
                    </PointModel>
                    <PointModel label="8">
                        <Coord>10 3 0</Coord>
                    </PointModel>
                    <PointModel label="9">
                        <Coord>10 6 0</Coord>
                        <Value>Force-Dy -1000</Value>
                    </PointModel>
                    <PointModel label="10">
                        <Coord>0 0 3</Coord>
                        <Value>RESTRAINTS -Dx true</Value>
                        <Value>RESTRAINTS -Dy true</Value>
                    </PointModel>
                </PointsList>
```

Figura B.3: Arquivo XML genérico para visualização de resultados no pós-processador (1^a parte).

```
                <PatchesList >
                    <Patch label="1">
                        <PatchPointsLabels >1 2</PatchPointsLabels >
                    </Patch>
                    <Patch label="2">
                        <PatchPointsLabels >2 3</PatchPointsLabels >
                    </Patch>
                    <Patch label="3">
                        <PatchPointsLabels >4 5</PatchPointsLabels >
                    </Patch>
                    <Patch label="4">
                        <PatchPointsLabels >5 6</PatchPointsLabels >
                    </Patch>
                    <Patch label="5">
                        <PatchPointsLabels >7 8</PatchPointsLabels >
                    </Patch>
                    <Patch label="6">
                        <PatchPointsLabels >8 9</PatchPointsLabels >
                    </Patch>
                    <Patch label="7">
                        <PatchPointsLabels >2 5</PatchPointsLabels >
                    </Patch>
                    <Patch label="8">
                        <PatchPointsLabels >5 8</PatchPointsLabels >
                    </Patch>
                    <Patch label="9">
                        <PatchPointsLabels >3 6</PatchPointsLabels >
                    </Patch>
                    <Patch label="10">
                        <PatchPointsLabels >6 9</PatchPointsLabels >
                    </Patch>
                </PatchesList >
            </GeoPostpModel >
        </GeoPostpModelList >
    </Step>
</PostpData >
```

Figura B.4: Arquivo XML genérico para visualização de resultados no pós-processador (2^a parte).

```
<?xml version="1.0" encoding="ISO8859-1" ?>
<!DOCTYPE PostpData SYSTEM "postp.dtd">
<PostpData>
    <Step label="single">
        <GeoPostpModelList>
            <GeoPostpModel label="Nodal">
                <PointsList>
                    <PointModel label="1">
                        <Coord>0 0 0</Coord>
                        <Value>RESTRAINTS -Dx true</Value>
                        <Value>RESTRAINTS -Dy true</Value>
                        <Value>Sxx 30</Value>
                        <Value>Syy -7</Value>
                    </PointModel>
                    <PointModel label="2">
                        <Coord>1.0 0 0</Coord>
                        <Value>Sxx 6</Value>
                        <Value>Syy -5</Value>
                    </PointModel>
                    <PointModel label="3">
                        <Coord>1.0 0.5 0</Coord>
                        <Value>Sxx 6</Value>
                        <Value>Syy 5</Value>
                    </PointModel>
                    <PointModel label="4">
                        <Coord>0 0.5 0</Coord>
                        <Value>RESTRAINTS -Dx true</Value>
                        <Value>Sxx 30</Value>
                        <Value>Syy 7</Value>
                    </PointModel>
                    <PointModel label="5">
                        <Coord>2.0 0 0</Coord>
                        <Value>Sxx 1.5</Value>
                        <Value>Syy -3</Value>
                    </PointModel>
                    <PointModel label="6">
                        <Coord>2.0 0.5 0</Coord>
                        <Value>Sxx 1.5</Value>
                        <Value>Syy 3</Value>
                    </PointModel>
                    <PointModel label="7">
                        <Coord>3.0 0 0</Coord>
                        <Value>Sxx 0.5</Value>
                        <Value>Syy -1</Value>
                    </PointModel>
                    <PointModel label="8">
                        <Coord>3.0 0.5 0</Coord>
                        <Value>Sxx 0.5</Value>
                        <Value>Syy 1</Value>
                    </PointModel>
                </PointsList>
```

Figura B.5: Arquivo XML de modelo bidimensional para visualização de resultados no pós-processador (1^a parte).

```
                <PatchesList >
                    <Patch  label="1">
                        <PatchPointsLabels >1  2  3  4</PatchPointsLabels >
                    </Patch>
                    <Patch  label="2">
                        <PatchPointsLabels >2  5  6  3</PatchPointsLabels >
                    </Patch>
                    <Patch  label="3">
                        <PatchPointsLabels >5  7  8  6</PatchPointsLabels >
                    </Patch>
                </PatchesList >
            </GeoPostpModel >
        </GeoPostpModelList >
    </Step>
</PostpData >
```

Figura B.6: Arquivo XML de modelo bidimensional para visualização de resultados no pós-processador (2^a parte).

Referências Bibliográficas

Azevedo, E. e Conci, A., 1999. *Computação Gráfica - Teoria e Prática*. 1ª ed, Elsevier.

Battaiola, A. L., 1998. *Apostila do Curso de Computação Gráfica*. Departamento de Computação - DC - Universidade Federal de São Carlos - UFSCar.

Brugiolo, M. A., 2004. Geração de Malhas Bidimensionais de Elementos Finitos Baseada em Mapeamento Transfinitos. Dissertação de Mestrado, UFMG - Universidade Federal de Minas Gerais, Belo Horizonte.

Buschmann, F., Meunier, R., Rohnert, H., Sommerland, P. & Stal, M., 1995. *Pattern-Oriented Software Architecture: A System of Patterns*. Vol. 1, Wiley.

Christian Icking, Rolf Klein, P. K. L. M., 2001, 'Voroglide', http://www.pi6.fernuni-hagen.de/GeomLab/VoroGlide/index.html.en Acessado em: 15/08/2006.

Deitel, H. M., Deitel P.J., 2003. *JAVA Como Programar*. 4ª ed, Bookman.

Deitel, H. M., Deitel P.J., 2006. *JAVA Como Programar*. 6ª ed, Pearson - Prentice Hall.

Del Savio, A. A., Santi, M. R. e Martha, L. F., 2004, Traçado de curvas offset para auxilio na geração de malhas, *em* 'CILAMCE'.

Du, C., 1996. 'An algorithm for automatic delaunay triangulation of arbitrary planar domains'. *Advances in Engineering Software*, vol. 27, pp. 21–26.

Figueiredo, L. H. e Carvalho, P. C. P., 1991. IMPA, Rio de Janeiro.

Fuina, J. S., 2004. Métodos de controle de deformações para análise não-linear de estruturas. Dissertação de Mestrado, Universidade Federal de Minas Gerais, Belo Horizonte.

Gamma, E., Helm, R., Jonhson, R. e Vlissides, J., 1995, Design patterns - element of reusable of object oriented software, Addi.

Gattass, M., W., C. F. e Fonseca, G., 1991. *Computação Gráfica Aplicada ao Método dos Elementos Finitos*. Vol. Notas de mini-curso da SBMAC, XIV CNMAC.

Jandl, P., 2003. *Mais JAVA*. Futura.

Martha, L. F., Carvalho, M. T. M. e Seixas, R. B., 1996, Volume contouring of generic unstrucutred meshes, IX SIBIGRAPI.

Mäntylä, M., 1987. *An Introduction to Solid Modeling*. Computer Science Press, Helsinki University Of Technology.

Nikishikov, G. P., 2003. 'Generating contours on fem/bem higher-order surfaces using java 3d textures'. *Advances in Engineering Software*, vol. 34, pp. 469–47.

Oñate, E., 1995. *Calculo de Estructuras por el Metodo de Elementos Finitos - Análisis estático lineal*. Centro Internacional de Métodos Numéricos en Ingeniería.

Ramalho, J. A., 2002. *XML - Teoria e Prática*. 1ª ed, Berkeley.

Rowe, G. W., 2001. *Computer Graphics With Java*. 4ª ed, Palgrave.

Salem, M. M., 1988. Um sistema gráfico interativo para representação de resultados e dimensionamento de lajes de concreto armado. Dissertação de Mestrado, PUC-Rio.

Santos, R. J., 1999. *Geometria Analítica e Algebra Linear*. UFMG, Belo Horizonte.

Sedgewick, R., 1988. *Algorithms*. Addison-Wesley Publishing Company, Princeton University.