Universidade Federal de Pernambuco
Centro de Tecnologia e Geociências
Departamento de Engenharia Mecânica

Pós-graduação em Engenharia Mecânica

Representação Computacional dos Dados Para a Simulação de Sistemas Multi-físicas pelo Método do Elemento Finito

Manassés do Nascimento Monteiro

Dissertação de Mestrado

Recife
2012

Universidade Federal de Pernambuco
Centro de Tecnologia e Geociências
Departamento de Engenharia Mecânica

Manassés do Nascimento Monteiro

Representação Computacional dos Dados Para a Simulação de Sistemas Multi-físicas pelo Método do Elemento Finito

Trabalho apresentado ao Programa de Pós-graduação em Engenharia Mecânica do Departamento de Engenharia Mecânica da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Engenharia Mecânica.

Orientador: *Prof. Dr. Félix Christian Guimarães Santos*

Recife
2012

Catalogação na fonte
Bibliotecário Marcos Aurélio Soares da Silva, CRB-4 / 1175

M775r Monteiro, Manassés do Nascimento.
Representação computacional dos dados para a simulação de sistemas multi-físicas pelo método do elemento finito / Manassés do Nascimento Monteiro. - Recife: O Autor, 2012.
xiv, 103 folhas, il., gráfs., tabs.

Orientador: Profº Drº Félix Christian Guimarães Santos.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CTG. Programa de Pós-Graduação em Engenharia Mecânica, 2012.
Inclui Referências e Apêndices.

1. Engenharia Mecânica. 2.Elemento Finito. 3.Estrutura de Dados. 4.Simulação. 5.Multi-Físicas. I. Santos, Félix Christian G. (Orientador). II. Título.

621 CDD (22. ed.) UFPE
BCTG/2012- 296

“REPRESENTAÇÃO COMPUTACIONAL DOS DADOS PARA A SIMULAÇÃO DE SISTEMAS MULTI-FÍSICAS PELO MÉTODO DO ELEMENTO FINITO”

MANASSÉS DO NASCIMENTO MONTEIRO

ESTA DISSERTAÇÃO FOI JULGADA ADEQUADA PARA OBTENÇÃO DO TÍTULO DE MESTRE EM ENGENHARIA MECÂNICA

ÁREA DE CONCENTRAÇÃO: MECÂNICA COMPUTACIONAL E PROJETO MECÂNICO
APROVADA EM SUA FORMA FINAL PELO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA/CTG/EEP/UFPE

_____
Prof. Dr. FÉLIX CHRISTIAN GUIMARÃES SANTOS
ORIENTADOR/PRESIDENTE

_____
Prof. Dr. JORGE RECARTE HENRÍQUEZ GUERRERO
COORDENADOR DO PROGRAMA

BANCA EXAMINADORA:

_____
Prof. Dr. FÉLIX CHRISTIAN GUIMARÃES SANTOS (UFPE)

_____
Prof. Dr. JOSÉ MARIA ANDRADE BARBOSA (UFPE)

_____
Prof. Dr. RICARDO MASSA FERREIRA LIMA (UFPE)

Dedico esta dissertação de mestrado aos meus pais, Marcos Antonio Monteiro e Glenilce do Nascimento Monteiro que me deram todo o apoio e suporte para eu alcançar este feito. Dedico também a minha tia Jaildes que tanto me ajudou e dedico também a minha esposa Elzanir que sempre esteve comigo pacientemente.

Agradecimentos

Primeiramente agradeço ao meu Deus por ter me ajudado nesta caminhada, pois sem Ele nada disso seria possível.

Agradeço aos meus pais, Marcos Antonio Monteiro e Glenilce do Nascimento Monteiro, pela minha educação e por ter me conduzido para o caminho certo.

Ao meu orientador, Professor Félix Christian Guimarães Santos, com quem tive a oportunidade de ingressar na pesquisa desde o quinto período da graduação em Engenharia Mecânica, onde pude evoluir minha capacidade de raciocínio e tive também a possibilidade de aprender algumas linguagens de programação aplicadas a simulação computacional.

Ao Professor José Maria Bezerra da Silva pela ajuda nas implementações relacionadas ao desenvolvimento do MPhyScas.

A todos os Professores de DEMEC (Departamento de Engenharia Mecânica) da UFPE que ajudaram na minha formação de Engenheiro Mecânico.

A minha esposa, Elzanir Leandro Bandeira da Silva Monteiro, uma pessoa que Deus pôs na minha vida para me ajudar. Agradeço pela paciência e compreeensão quando eu tive que estar ausente em algumas ocasiões para me dedicar ao desenvolvimento da minha dissertação.

A toda minha família, aos meus irmãos Saulo, Débora, Moisés, Maeli e Misael que não se encontra mais entre nós, em especial a minha tia Jaildes que teve uma parcela considerável na minha formação.

Ao meu amigo José Maurício da Silva pelos momentos de descontrações e pelo incentivo.

A todos os meus amigos da graduação do Centro de Tecnologia e Geo-Ciências - CTG; aos amigos da Pós-graduação da Universidade Federal de Pernambuco: O professor e escritor Paulo Gallindo, José Mendes, Igor, Anderson; aos amigos do Labcom: Rodrigo, Michel, João e Rafael; aos amigos do Laboratório de Robótica: Felipe Cruz, Guaraci e Brito;

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo apoio finaceiro durante parte do meu mestrado.

E finalmente, agradeço a todos que me ajudaram direta e indiretamente no desenvolvimento deste trabalho e que não foram citados.

”De tudo ficaram três coisas: A certeza de que estamos começando, A certeza de que é preciso continuar e A certeza de que podemos ser interrompidos antes de terminar. Façamos da interrupção um caminho novo, da queda um passo de dança, do medo uma escola, do sonho uma ponte, e assim terá valido apena existir!”
(Fernando Sabino)

Resumo

Simulação de fenômenos naturais acoplados está se tornando uma prática importante na engenharia, visto que os sistemas físicos onde ocorrem vários fenômenos de forma acoplada são frequentes em várias áreas do conhecimento. As simulações destes sistemas têm permanecido como um desafio importante devido ao fato de que as relações de acoplamento entre os fenômenos sempre implicam em troca de dados. Isto significa que alterações simples nos métodos de solução acarretam grandes alterações nos sistemas computacionais, dificultando o aprimoramento destes sistemas.

Com o enorme avanço da Mecânica Computacional, vem-se tornando possível a simulação de eventos naturais mais complexos bem como o uso dessas simulações no desenvolvimento de sistemas de engenharia. Isto é feito através da modelagem computacional: o desenvolvimento de versões discretizadas de teorias mecânicas, as quais são acessíveis a cálculos digitais, juntamente com os processos complexos de manipulação desta representação digital para fornecer uma ideia real de como esses sistemas se comportam.

Este trabalho tem como objetivo a representação computacional dos dados para a simulação de sistemas multi-físicas pelo Método do Elemento Finito (MEF). O MEF é uma forma de se obter uma aproximação numérica de uma teoria matemática que descreve um comportamento físico, ele tem sido frequentemente utilizado na Mecânica Computacional e é considerado uma técnica computacional para solução de equações diferenciais e integrais que surgem em vários campos da engenharia.

Neste trabalho é realizada a descrição do programa MPhyScas (Multi-Physics Multi-Scale Solver Environment), que é um ambiente de desenvolvimento de simuladores baseados no MEF desde a definição da arquitetura até as descrições das camadas: Kernel, Block, Group e Phenomenon. É apresentada a estrutura de dados geométricos (representação da geometria, malha geométrica e do fenômeno, funções de forma, etc). São apresentados também as operações no baixo nível desde a integração numérica até as operações de álgebra linear e resolvedores.

Palavras-chave: Método do elemento finito, Estrutura de dados, Simulação, Multi-físicas

Abstract

Coupled simulation of natural phenomena is becoming an important practice in engineering, since the physical systems where there are so many coupled phenomena are common in many areas of knowledge. The simulations of these systems have remained a major challenge due to the fact that the coupling relations between the phenomena always involve in exchanging data. It means that simple changes in the methods of solution can lead to big changes in computacional systems, hindering the improvement of these systems.

With the tremendous advancement of Computational Mechanics, is becoming possible to simulate natural events more complex and the use of these simulations in the development of engineering systems.It is done by computational modeling, the development of discretized version of mechanical theories, which are accessible by digital calculation, together with complex procedures for manipulation of the digital representation to provide a true picture of how these systems behave.

This work aims at the computational representation of the data for the simulation of multi-physical systems by Finite Element Method (FEM).The FEM is a way to obtain a numerical approximation of a mathematical theory that describes a physical behavior, it has often been used in Computational Mechanics and is considered a computational technique for solving differential and integral equations that arise in various fields of engineering.

This work presents the description of the program MPhyScas (Multi-Physics Multi-Scale Solver Environment), which is a development environment of simulators based on FEM since the definition of architecture to the descriptions of the layers: kernel, Block, Group and Phenomenon. Is shown the structure of geometric data (representation of the geometry, mesh geometric and phenomenon, shape functions, etc.). Are presented also the perations in the lower level since the numerical integration to operations linear algebra and solvers.

Keywords: Finite element method, Data structure, Simulation, Multi-physics

Lista de Figuras

Lista de Tabelas

Sumário

CAPÍTULO 1

Introdução

1.1 O Método do Elemento Finito - MEF

O Método do elemento finito - MEF [1] surgiu em 1956 e é uma técnica para discretização de equações diferenciais parciais que descrevem fenômenos, cujo comportamento é determinado por variáveis no contínuo (espaço e tempo). Este método basea-se na subdivisão de um domínio geométrico complexo utilizando entes mais simples, porém mais tratáveis matematicamente.

Os simuladores baseados no método do elemento finito podem obedecer a uma hierarquia de processos, ou seja, podem ser modularizados em uma arquitetura de camadas (ver Figura 1.1).

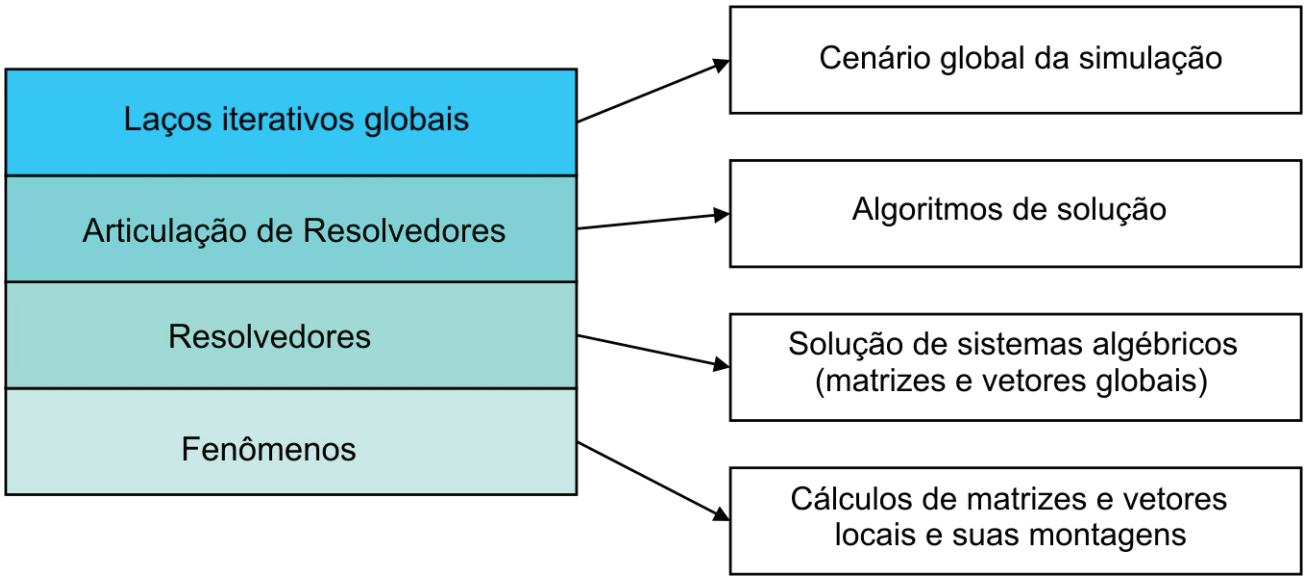


Figura 1.1 Camadas dos simuladores baseados no MEF

Na primeira camada tem-se os laços iterativos globais, os quais envolvem progressão no tempo, adaptação de modelos e discretização. Esta camada corresponde ao cenário global da simulação. Na segunda camada tem-se a articulação entre os resolvedores, que corresponde aos algoritmos de solução. Esta camada articula montagens e soluções de diferentes sistemas algébricos. A terceira camada, resolvedores, é responsável pela montagem e solução de sistemas algébricos. A quarta e última camada, Fenômenos, é responsável pelo cálculo e montagem de matrizes e vetores no nível do elemento finito (matrizes e vetores locais).

1.2 Sistemas Multi-físicas

Os sistemas multi-físicas acoplados são aqueles em que um determinado fenômeno atuando em um certo domínio geométrico, é dependente de outro fenômeno.

Mesmo com a definição das camadas (Figura 1.1), isto não fornece uma visão de como os procedimentos em diferentes camadas interagem, nem como é realizada a troca de dados entre as camadas [2] e, enfim, como pode ser descrita a dependência entre as camadas. Isto torna-se importante nas definições de abstrações, as quais podem padronizar a forma como processos e dados nas camadas se comportam e interagem. Para problemas desacoplados, existem bibliotecas para o método do elemento finito, as quais fornecem suficiente poder de representação computacional (abstração) visto em [3] com a finalidade de facilitar a construção de simuladores em um tempo razoável e com alto grau de reusabilidade. Mas, isto não ocorre quando se trata de fenômenos acoplados como visto em [4], ou seja, sempre que um fenômeno depende de dados de outro fenômeno, as abstrações destas bibliotecas se tormam ineficazes no sentido da reusabilidade e manutenabilidade. Este fato ocorre porque essas bibliotecas não fornecem abstrações para interação entre fenômenos.

Os Problemas multi-físicas acoplados tornam-se mais complexos no nível do elemento finito e no nível da solução. As abstrações utilizadas em sistemas desacoplados não podem ser empregadas com eficiência - no que diz respeito a reusabilidade, adaptabilidade e manutenabilidade, em sistemas acoplados. Estas abstrações não representam de forma adequada a troca de dados e a dependência entre as camadas, que podem ocorrer em problemas acoplados. Isto acontece devido a existência de uma relação muito forte entre as decisões no nível dos algoritmos de solução e os cálculos no nível do elemento finito.

Esta dificuldade pode ser observada e esclarecida analisando os procedimentos no nível mais baixo, isto é, no nível do elemento finito. Estes procedimentos estão relacionados com a produção e montagem das matrizes e vetores locais para cada fenômeno. As matrizes e vetores locais calculadas para cada fenômeno podem estar acopladas com outros fenômenos, assim os cálculos de uma determinada quantia podem necessitar de informações de outros fenômenos. Essas informações são definidas no nível dos algoritmos de solução, fazendo com que alterações no algoritmo de solução venham impor extensa reprogramação em ambas as camadas. Existem outros requisitos que tornam o problema ainda mais difícil de ser solucionado:

- O acoplamento, visto em [5], pode ocorrer em apenas uma parte do domínio geométrico (por exemplo, em alguma parte do contorno);
- Dois fênomenos acoplados em um componente geométrico podem possuir malhas geométricas diferentes, significando que o uso de um estado acoplado por um fenômeno necessita da transferência deste estado de uma malha para outra.

As dificuldades acima apresentadas geram o seguinte problema: A necessidade de adquirir abstrações mais poderosas e versáteis, ou seja, abstrações que podem adequadamente representar e encapsular as informações, as relações e os processos pertencentes ao nível do elemento finito e que sejam do interesse da modelagem numérica de um fenômeno, a fim de descrever e implementar abstrações dos fenômenos no contexto de sistemas multi-físicas acoplados. Ainda, como representar abstratamente os processos e dados inerentes a cada uma das camadas de

cálculo, garantindo que alterações em uma determinada camada impliquem em pouquíssima ou nenhuma alteração nas outras camadas. As soluções para este problema serão apresentadas nos capítulos posteriores.

1.3 O projeto MPhyScas

O MPhyScas é um projeto do Departamento de Engenharia Mecânica (DEMEC) da Universidade Federal de Pernambuco, que tem o propósito de desenvolver simuladores com a capacidade de resolver problemas multi-físicas utilizando o Método do Elemento Finito (MEF). Além da importante contribuição no avanço tecnológico, este projeto vem inserindo alunos das Engenharias na rede de pesquisa em modelagem computacional [6]. O MPhyScas tem o apoio da FINEPE (Financiadora de Estudos e Projetos).

1.4 A proposta

A proposta deste trabalho está descrita abaixo:

- Definir as camadas do MPhyScas adequadamente, incluindo dados e estruturas;
- Definir as estruturas de dados geométricos;
- Definir os processos de integração numérica; e
- Definir uma interface que permita realizar operações de álgebra linear.

1.5 Organização do Trabalho

Esta dissertação divide-se em sete capítulos. Neste primeiro capítulo foi apresentada uma visão do método do elemento finito, sua importância e estruturação do método como organização hierárquica de processos virtuais. E também foi apresentada a descrição do sistema multi-física, sua importância e dificuldades e a necessidade de abstrações mais poderosas e versáteis. Foi apresentada uma síntese sobre o projeto MPhyScas e a proposta deste trabalho. Nas implementações deste trabalho será utilizada a linguagem C++ [7].

Capítulo 2 - **Descrição do MPhyScas** - são apresentadas as camadas do MPhyScas, sua constituição, suas responsabilidades e seus objetos.

Capítulo 3 - **Estrutura de Dados Geométricos** - são apresentadas as estruturas de dados da malha geométrica e do fenômeno, funções de forma, entre outros. E são explicitados também os campos vetoriais do fenômeno.

Capítulo 4 - **Processos de Integração Numérica** - são apresentados todos os processos referentes à integração numérica, sendo os processos divididos em dados intrínsecos e extrínsecos de integração e a integração propriamente dita.

Capítulo 5 - **Campos Vetoriais Discretos, Operações de Álgebra Linear e Resolvedores** - são apresentados os campos vetoriais discretos, as operações de álgebra linear (tipo BLAS)

e os resovedores. E são apresentadas também as formas fracas, que são ferramentas para o cálculo das quantias que estão no baixo nível.

Capítulo 6 - **Resultados e Análises** - são apresentados resultados relacionados à geração de malhas e visualização gráfica. São analisadas as dificuldades e facilidades nas implementações do MPhyScas relacionadas à construção de simuladores para simulação de sistemas multi-físicas, esta análise foi realizada através de um estudo de caso com um aluno que está inserido no programa da Pós-graduação em Engenharia Mecânica na Área da Mecânica Computacional/Projetos.

Capítulo 7 - **Conclusões e Trabalhos Futuros** - são apresentadas as conclusões e considerações finais sobre o trabalho apresentado. Neste capítulo também são apresentadas as propostas para trabalhos futuros.

CAPÍTULO 2

Descrição do MPhyScas

2.1 MPhyScas

O MPhyScas (Multi-Physics Multi-Scale Solver Environment) visto em [8], é um ambiente de desenvolvimento de simuladores baseados no método do elemento finito. O termo multi-física significa um conjunto de interações entre fenômenos, no tempo e no espaço. Estes fenômenos são usualmente de naturezas diferentes (transferência de calor [9], deformação de sólidos [10], campos eletromagnéticos [11], etc.). Um sistema multi-física também é conhecido como um sistema com fenômenos acoplados, ou seja um fenômeno pode depender de outros fenômenos.

2.1.1 Arquitetura do MPhyScas

A arquitetura do MPhyScas proposta em [12], propõe uma representação computacional para as camadas utilizando padrões (Figura 2.1) onde, o **Kernel** representa os laços iterativos globais, O **Block** representa a articulação dos resolvedores, O **Group** representa os resolvedores e os fenômenos são representados pelo **Phenomenon**. Os principais requisitos desta arquitetura são as seguintes: flexibilidade no desenvolvimento de simuladores, possibilidade de extensão do sistema através da integração de componentes, reusabilidade de processos e dados.

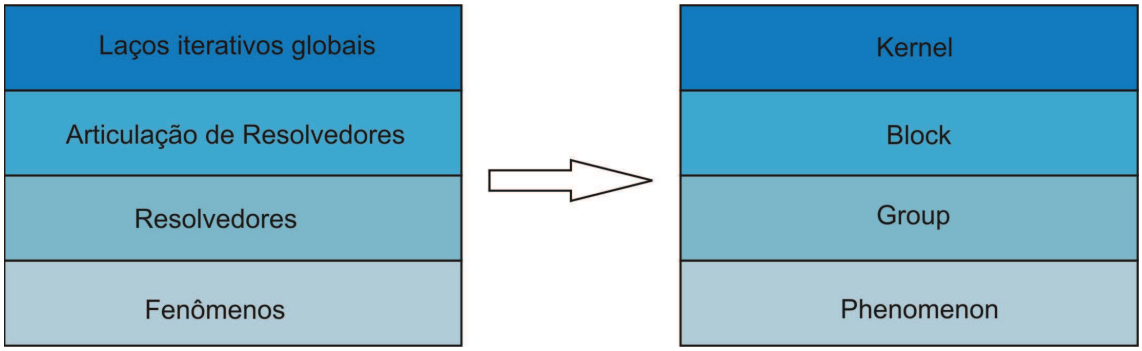


Figura 2.1 Representação computacional das camadas do simulador

A especificação do nosso simulador no MPhyScas é representada por quatro camadas como visto em [13], [14] e [15]:

- **Kernel**: Possui uma constituição simples tendo o seu comportamento definido por um

Algoritmo (*KernelAlgth*). Só há um objeto representando o **Kernel**. O *KernelAlgth* representa o cenário da simulação no seu mais alto nível. Este é o local para a definição e execução de laços adaptativos;

- **Block**: Esta camada é constituida por vários objetos da classe **Block**. Cada objeto **Block** pode possuir vários algoritmos (*BlockAlgth*). Cabe ao *KernelAlgth* a execução destes objetos. Cada objeto *BlockAlgth* representa métodos de soluções locais, como Newton-Raphson para sistemas não lineares, métodos para cálculo de uma nova solução no tempo, etc;

- **Group**: É constituido de vários objetos da classe **Group**. Cada objeto **Group** é capaz de armazenar estados (escalares, vetores e matrizes) e executar operações dos tipos BLAS I, II e III, e resolver sistemas algébricos lineares. Objetos **Group** trabalham sob demandas de seus respectivos objetos **Block**; e

- **Phenomenon**: A última camada é constituida de objetos da classe **Phenomenon**. Cada objeto **Phenomenon** pode realizar operações como calcular quantias dependentes de malha ou não, mas que podem depender de estados de outro objeto **Phenomenon**. Os entes geométricos da simulação estão armazenados nos objetos **Phenomenon**.

2.1.2 SystData

É uma estrutura de dados para armazenar dados de troca de informações entre objetos das quatro camadas. Cada objeto pode expor dados e requer dados. Dados expostos são dados gerados pelo objeto e acessíveis às camadas, as quais necessitam desses dados. Dados requeridos são dados expostos por objetos de camadas superiores ou imediatamente inferior. Portanto, não há troca de informações entre objetos de uma mesma camada, a não ser para dados de acoplamento no nível do Phenomenon. Além disto, uma camada qualquer pode requerer dados de qualquer camada superior, mas somente da camada imediatamente inferior.

2.1.3 Hierarquia no MPhyScas

No MPhyScas a hierarquia entre as camadas se dá através de uma estrutura de dados em forma de árvore binária (Figura 2.2), onde o **Kernel** é o mais alto nível, sendo que o **Kernel** pode conter vários objetos **Block**, e os objetos **Block** poderá conter vários objetos **Group**, mas um determinado objeto **Group** não pode pertencer a mais de um **Block**. E cada **Group** pode conter vários objetos **Phenomenon**, sendo que os objetos **Phenomenon** não pode pertencer a mais de um **Group**.

No exemplo abaixo (Figura 2.3), demonstra que o **Kernel** possui apenas o *KernelAlgth* (AK). Os **Block** (Bi) para i = 0 e 1, possui vários *BlockAlgth* (Aj) para j = 0,...,10. Os **Group** (Gk) para k = 0,...,4, possui vários objetos **GroupTasks** (tarefas do Group) (GTn) para n = 0,...,16. E o **Phenomenon** (Pk) para k = 0,...,7, possui vários objetos **WeakForms** (WFz) para z = 0,...,25.

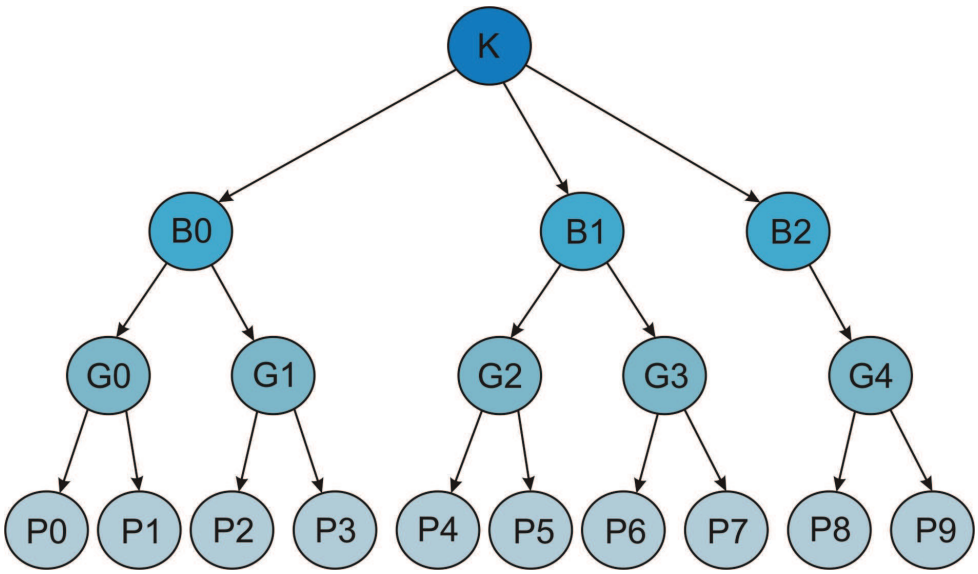


Figura 2.2 Hierarquia das camadas

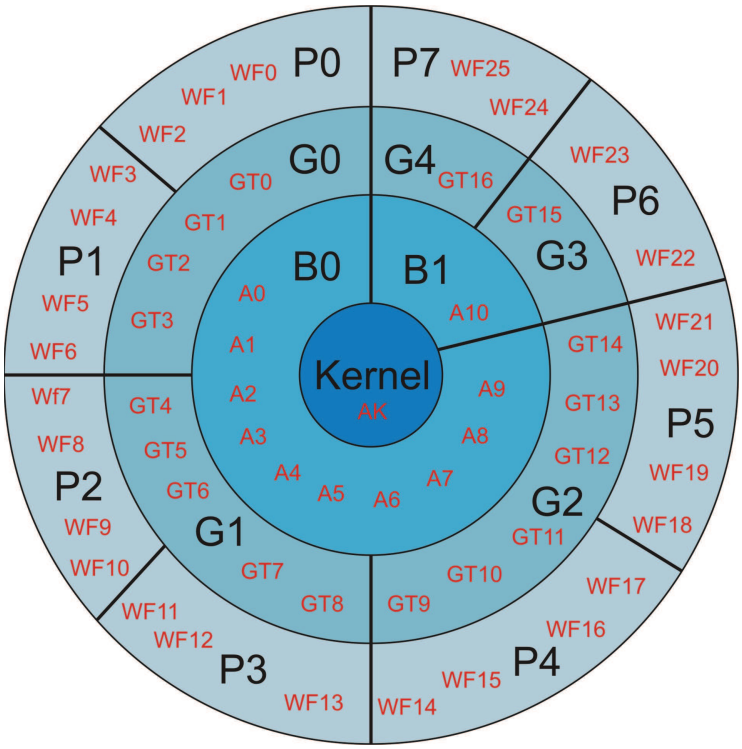


Figura 2.3 Estruturação das camadas

2.2 Kernel

O **Kernel** armazena os dados do sistema, que são relacionados com os parâmetros necessários às suas iterações (por exemplo, variação no tempo (ΔT) exposto pelo **Block** ao **Kernel**). O **Kernel** pode expor dados a todas as camadas, mas só poderá requerer dados do **Block**, que é a camada imediatamente inferior. A classe **Kernel** representa somente a interface.

A definição da classe `Kernel` e seus principais atributos e métodos estão descritos abaixo:

```
class Kernel{
  protected:
    kernelAlgth *krnlAlgthm;
    map<int, Block *> * blkvec;
    SystData *stdt;
    vector<string> *expDefs;
    vector<string> *reqDefs;

  public:
    Kernel();
    virtual ~Kernel();
    virtual int execKernel(int);
};
```

- `kernelAlgth *krnlAlgthm`: É o único Algoritmo do **Kernel**.
- `map<int, Block *> * blkvec`: Mapeia os respectivos códigos do **Block** para o objeto **Block** (a ordem é importante).
- `SystData *stdt`: É um objeto responsável em expor dados acessível a outras camadas e requerer dados das mesmas.
- `vector<string> *expDefs`: É um vetor que contém as definições dos dados expostos pelo **kernel** e acessíveis às camadas inferiores.
- `vector<string> *reqDefs`: É um vetor que contém as definições dos dados requeridos pelo **kernel** às outras camadas.
- `virtual int execKernel(int)`: Este método é responsável em inicializar procedimentos demandados pelo **Kernel**.

A definição da classe **KernelAlgth** e seus principais atributos e métodos estão descritos abaixo:

```
class kernelAlgth{
  protected:
    int code;
    vector<Block *> *blockVec;
```

```
    map<int, vector<int> *> *blkAlgthVec;
    vector<double *> *expDt;
    map<int, vector<double *> *> *reqDtFromBlks;
    vector<double> *algthPars;

  public:
    kernelAlgth();
    virtual ~kernelAlgth();
    virtual int executeAlgth();
};
```

- `vector<Block *> *blockVec`: Vetor que contém referência para o objeto **Block**.
- `map<int, vector<int> *> *blkAlgthVec`: Mapeia o **id** do **Block** para um vetor de inteiros que contém os algoritmos deste **Block**, sendo que a ordem é importante.
- `vector<double *> *expDt`: É um vetor de ponteiros para as áreas de memórias onde estão os dados expostos pelo **Kernel**, que são acessíveis às outras camadas.
- `map<int, vector<double *> *> *reqDtFromBlks`: Mapeia o **id** do **Block** para um vetor de dados que são requeridos ao **Block**. A ordem é importante.
- `Vector<double> *algthPars`: Contém os parâmetros do **Kernel**.
- `virtual int executeAlgth()`: É o método responsável pela execução do *KernelAlgth*.

A classe **KernelAlgth** representa somente a interface, devendo o Algoritmo ser implementado em uma classe que herde esta classe.

2.2.1 Exemplo Elasto-plástico com difusão

Segue abaixo o algoritmo do **Kernel** (ver apêndice A) referente à simulação de um problema elasto-plástico com difusão, ou seja, um problema envolvendo elasticidade [16], plasticidade [17] e difusão [18]. A descrição de cada linha deste algoritmo está descrita no apêndice A.

KERNEL algorithm:

- GetInterval();
- BLK 0 ExecAlgth(0); (Initialize $\mathbf{C_n}, \mathbf{U_n}, \mathbf{X_n}, \boldsymbol{\varepsilon}^{\mathbf{p}}_{\mathbf{n}}, p_n$)
- BLK 0 ExecAlgth(1); (Compute Δt_b)

$\Delta t = \Delta t_b$;

- BLK 0 ExecAlgth(2); (Compute $\dot{\mathbf{C}}_{\mathbf{n}}$)

$t_1 = t_0 = 0.0$;

- BLK 1 ExecAlgth(0); (Record $mesh, \mathbf{C_n}, \mathbf{U_n}, \mathbf{t_1}, \Delta t$)
While $(t_1 < I)$

$t_1 = t_0 + \Delta t$;
- BLK 0 ExecAlgth(3); (Zero:$\mathbf{K}, \mathbf{F}, \mathbf{M}, \mathbf{B}, \mathbf{G}$ assigns the instant k to n:$\mathbf{C_K}, \mathbf{U_K}, \mathbf{X_K}, \boldsymbol{\varepsilon}^{\mathbf{p}}_{\mathbf{K}}, p_K$)
- BLK 0 ExecAlgth(4); (Compute new solution)
- BLK 1 ExecAlgth(0); (Record: $mesh, \mathbf{C_n}, \mathbf{U_n}, t_1, \Delta t$)
- BLK 0 ExecAlgth(1); (Compute new Δt)
$\Delta t = \Delta t_b$;
$t_0 = t_1$;

End

2.3 Block

O **Block** serve ao **Kernel** (expõem dados) e às demais camadas, mas só poderá requerer dados do **Kernel** e do **Group**. Cada **Block** é responsável por uma certa quantidade de objetos **Group** (Figura 2.2), os quais não podem pertencer a outros **Block**. As solicitações de um **Block** para um nível mais baixo devem ser endereçadas aos seus objetos **Group**. O **Block** armazena parâmetros relacionados com suas: iterações, laços e parâmetros relacionados com seus procedimentos.

A definição da classe **Block** e seus principais atributos e métodos estão descritos abaixo:

```
class Block{
  protected:
    map<int, blockAlgth *> *blockAlgs;
    map<int, Group *> *grps;
    SystData *stdt;
    vector<string> *expDefs;
    vector<string> *reqDefs;

  public:
    Block();
    ~Block();
    virtual int executeBlockAlgth(int);
};
```

- `map<int, blockAlgth *> *blockAlgs`: Mapeia o código do algoritmo do **Block** para seu objeto **BlockAlgth**.

- `map<int, Group *> *grps`: Mapeia o código do **Group** para seus objetos **Group**.
- `SystData *stdt`: É um objeto responsável em expor dados acessíveis às outras camadas e requerer dados das mesmas.
- `vector<string> *expDefs`: É um vetor com as definições dos dados expostos por este **Block** que são acessíveis às outras camadas.
- `vector<string> *reqDefs`: É um vetor com as definições dos dados requeridos por este **Block** às outras camadas. A ordem é importante.
- `virtual int executeBlockAlgth(int id)`: É o método responsável pela execução dos *BlockAlgth*.

A definição da classe **BlockAlgth** e seus principais atributos e métodos estão descritos abaixo:

```
class blockAlgth{
  protected:
    int code;
    vector<Group *> *grpVec;
    vector<double> *algthPars;
    vector<double *> *expDefs;
    vector<double *> *reqDtNotGrpDefs;
    map<int, vector<double *> *> *blkGrpDt;
    map<int, vector<int> *> *grpTsks;

  public:
    blockAlgth();
    virtual ~blockAlgth();
    virtual int executeAlgth();
};
```

- `vector<Group *> *grpVec`: É um vetor de ponteiros para objetos **Group** pertencente a este **Block** e que são utilizados em *BlockAlgth*. A ordem é importante.
- `vector<double> *algthPars`: É um vetor que contém os parâmetros do *BlockAlgth*.
- `vector<double *> *expDefs`: É um vetor com ponteiros para as áreas de memória onde estão os dados expostos por este *BlockAlgth*. É um subconjunto dos dados expostos pelo **Block**.
- `vector<double *> *reqDtNotGrpDefs`: É um vetor que contém os dados requeridos pelo *BlockAlgth* e que não vêm de nenhum **Group**.

- `map<int, vector<double *> *> *blkGrpDt:` Mapeia os **id's** dos **Group** para um vetor de ponteiros, que apontam para os dados requeridos por este *BlockAlgth* a este **Group**.
- `map<int, vector<int> *> *grpTsks:` Mapeia os **id's** dos **Group** para um vetor de **id's** dos objetos **GroupTask** deste **Group**.
- `virtual int executeAlgth():` É o método que executa o *BlockAlgth*.

A classe **BlockAlgth** representa somente a interface, devendo o Algoritmo ser implementado em uma classe que herde esta classe.

2.3.1 Exemplo Elasto-plástico com Difusão

Seguem abaixo os algoritmos do Block (ver apêndice A) referentes à simulação de um problema elasto-plástico com difusão, ou seja, um problema envolvendo elasticidade, plasticidade e difusão. A descrição de cada linha destes algoritmos estão descritas no apêndice A.

BLOCK Algorithms:

- BLK 0 - ExecAlgth(0) - Initialize
  (Group1 : executeGroupTask(GT0))
  -
  (Group0 : executeGroupTask(GT0))

- BLK 0 - ExecAlgth(1) - Calculate Δt_b
  (Group0 : executeGroupTask(GT1))
  -
  (Group1 : executeGroupTask(GT1))
  -
  $\Delta t_b = \min\{\Delta t_{g0}, \Delta t_{g1}\}$

- BLK 0 - ExecAlgth(2) - Calculate $\dot{\mathbf{C}}_\mathbf{n}$
  (Group1 : executeGroupTask(GT2))
  .
  .
  .

- BLK 0 - ExecAlgth(3) - Zero, Swap

(Group0 : executeGroupTask(GT2))

.

.

.

- BLK 0 - ExecAlgth(4) - Elasticity, Plasticity .....

tol;
$tol_{\Delta p}$;
$nItersMax$;
$nItersMax_{\Delta p}$;
npt;
- Initialize err = 1.0;

While $((err > tol)\&\&(nIters \leq nItersMax))$

- BLK 0 - execAlgth(4.1) - Elasticity .....

(Group0 : executeGroupTask(GT4))

-

(Group0 : executeGroupTask(GT4))

-

(Group0 : executeGroupTask(GT5))

- BLK 0 - ExecAlgth(4.2) - Plasticity .....

(Group0 : executeGroupTask(GT6))

-

(Group0 : executeGroupTask(GT6))

-

(Group0 : executeGroupTask(GT6))

- BLK 0 - ExecAlgth(4.3) - Assembly System

(Group1 - GT7)

- BLK 0 - ExecAlgth(4.4) - Calculate $\mathbf{C_K}$

(Group1 : executeGroupTask(GT8))

-

(Group1 : executeGroupTask(GT8))

-

(Group1 : executeGroupTask(GT8))

- BLK 0 - ExecAlgth(4.5) - Calculate Error
  (Group0 : executeGroupTask(GT7))
  -
  (Group1 : executeGroupTask(GT9))
  -
  $err = \max\{errU_{K+1}, errC_{K+1}\}$

- BLK 0 ExecAlgth(4.6) - Swap ($\mathbf{U_{K+1}}, \mathbf{C_{K+1}}$)
  (Group0 : executeGroupTask(GT8))
  -
  (Group1 : executeGroupTask(GT10))

- BLK 0 ExecAlgth(4.7) - Calculate $\dot{\mathbf{C}}_\mathbf{n}$
  (Group1 : executeGroupTask(GT11))

- BLK 0 - ExecAlgth(4.8) - Swap ($\mathbf{U_n}, \mathbf{C_n}, \mathbf{X_n}, \boldsymbol{\varepsilon}^\mathbf{p}_\mathbf{n}, p_n$)
  (Group1 : executeGroupTask(GT12))
  -
  (Group0 : executeGroupTask(GT9))

Else

Break Simulation

End

- BLK 0 ExecAlgth(5) - Calculate Δt_b
  (Group0 : executeGroupTask(GT10))
  -
  (Group1 : executeGroupTask(GT13))
  -
  $\Delta t_b = \min\{\Delta t_{g0}, \Delta t_{g1}\}$

- BLK 1 - ExecAlgth(0) - Record $mesh, \mathbf{C}, \mathbf{U}, t, \Delta t$

(Group2 : executeGroupTask(GT0))
-
(Group2 : executeGroupTask(GT0))
-
(Group2 : executeGroupTask(GT0))

End

2.4 Group

O **Group** traduz as solicitações do seu **Block** em solicitações para os objetos **Phenomenon** (expõem dados) em um nível mais baixo. Articula as atividades a serem executadas pelos **Phenomenon** em um nível mais baixo. Os objetos **Phenomenon** servem aos objetos **Group**. Cada **Group** é responsável por uma certa quantidade de objetos **Phenomenon**, os quais não podem pertencer a outros **Group**.

A definição da classe **Group** e seus principais atributos e métodos estão descritos abaixo:

```
class Group{
  protected:
    map<int, Phenomenon *> *Phens;
    map<int, int*> *glbSts;
    Phenomenon *gPhen;
    SystData *stdt;
    map<int, GroupTask *> *grpTsks;
    vector<string> *expDefs;
    vector<string> *reqDefs;

  public:
    Group();
    virtual ~Group();
    virtual int computeGroupTask(int);

};
```

- `map<int, Phenomenon *> *Phens`: Mapeia o **id** de um objeto **Phenomenon** para o ponteiro deste objeto. **Phenomenon**.
- `map<int, int*> *glbSts`: Mapeia o **id** de um estado global para o seu tipo.
- `Phenomenon *gPhen`: Definição do objeto **gPhen**, que é o representante do **Group** no nível do **Phenomenon**.

- `SystData *stdt`: É um objeto responsável por expor dados acessível às outras camadas e requerer dados das mesmas.
- `map<int, GroupTask *> *grpTsks`: Mapeia o **id** de um **GroupTask** para ponteiro do seu objeto **GroupTask**.
- `vector<string> *expDefs`: Vetor que contém as definições dos dados expostos pelo **Group** às outras camadas.
- `vector<string> *reqDefs`: Vetor que contém as definições dos dados requeridos pelo **Group** às outras camadas.
- `virtual int computeGroupTask(int id)`: Método responsável em executar os **GroupTasks**.

O **Group** armazena matrizes, vetores e escalares globais, armazena também objetos **GroupTask** (tarefas do Group), os quais encapsulam procedimentos, onde as articulações dos objetos **Phenomenon** do **Group** são necessárias. Os **GroupTask** são programáveis e os seus dados são informações padronizadas.

A definição da classe **GroupTask** e seus principais atributos e métodos estão descritos abaixo:

```
class GroupTask{
  protected:
    string def;
    vector<int> *phenExecCodes;
    vector<string> *execCodeDefs;
    vector<Phenomenon *> *phenVec;

  public:
    GroupTask();
    virtual ~GroupTask();
    int computeGrpTsk();
};
```

- `string def`: Definição de seu objeto **GroupTask**.
- `vector<int> *phenExecCodes`: Vetor de inteiros contendo **ExecCodes(Execution code)** para seu objeto **Phenomenon**.
- `vector<string> *execCodeDefs`: Vetor que contém a definição de `ExecCode` para o respectivo `ExecCode` em `phenExecCodes`. Este vetor possui o mesmo tamanho do `vetor vector<int> *phenExecCodes`.
- `vector<Phenomenon *> *phenVec`: Vetor de ponteiro para o **Phenomenon** que contém os objetos **Phenomenon**. A ordem é importante.

- `int computeGrpTsk(int id)`: Este método executa **GroupTask**.

O **Group** possui o **gPhen** que é o responsável do **Group** no nível do **Phenomenon**. O **gPhen** requer dados do **Block** e expõe ao **Phenomenon** e pode possuir objetos que encapsulam resolvedores lineares responsáveis por resolver sistemas de equações lineares no nível do **Phenomenon**.

A definição da classe **gPhenomenon** e seus principais atributos e métodos estão descritos abaixo. A classe **gPhenomenon** herda a classe **Phenomenon**:

```
class gPhenomenon: public Phenomenon{
  protected:
    map<int, Phenomenon *> *phns;
    vector<string> *expDt;
    vector<string> *notFromPhnReqDt;
    map<int, vector<string> *> *grpPhenReqDt;
    map<int, Solver *> *solvers;

  public:
    gPhenomenon();
    virtual ~gPhenomenon();
    BlasOperator<BLAS> *getBlasOpr();
    virtual int computeExecCode(int cd);

};
```

- `map<int, Phenomenon *> *phns`: Mapeia o **id** de um objeto **Phenomenon** para o ponteiro deste objeto **Phenomenon**.
- `vector<string> *expDt`: Vetor contendo as definições dos dados expostos do **Phenomenon**.
- `vector<string> *notFromPhnReqDt`: Vetor contendo as definições dos dados requeridos pelo **Group** e que não vêm do **Phenomenon**.
- `map<int, vector<string> *> *grpPhenReqDt`: Mapeia o **id** do **Phenomenon** para um vetor que contém as definições dos dados requeridos pelo **Group** a este **Phenomenon**.
- `map<int, Solver *> *solvers`: Mapeia o **id** do **solver** para o objeto **solver**.
- `BlasOperator<BLAS> *getBlasOpr()`: Método responsável em retornar o operador de BLAS I, II e III, que é responsável em realizar operações entre escalares, vetores e matrizes.
- `virtual int computeExecCode(int cd)`: Método responsável em executar `ExecCodes(cd)` contido no **gPhenomenon**.

2.4.1 Exemplo Elasto-plástico com Difusão

Seguem abaixo as tarefas do Group (ver apêndice A) referentes à simulação de um problema elasto-plástico com difusão, ou seja, um problema envolvendo elasticidade, plasticidade e difusão. A descrição de cada linha relacionada às tarefas do Group estão descritas no apêndice A.

Tarefas do GROUP:

- **GROUP 0 - pertencente ao BLK 0**

(GT0 : Phen0 : ComputeExecCode(0))

.
.
.

(GT9 : Phen0 : ComputeExecCode(10))

- **GROUP 1 - pertencente ao BLK 0**

(GT0 : Phen1 : ComputeExecCode(0))

.
.
.

(GT9 : Phen1 : ComputeExecCode(14))

- **GROUP 2 - pertencente ao BLK 1**

(GT0 : Phen2 : ComputeExecCode(0))

-

(GT0 : Phen2 : ComputeExecCode(1))

(GT0 : Phen2 : ComputeExecCode(2))

2.5 Phenomenon

Os **Phenomenon** servem a seus respectivos objetos **Group**. Os objetos **Phenomenon**: armazenam dados relacionados aos parâmetros constitutivos ou outros parâmetros, os quais são específicos dos objetos **Phenomenon**; armazenam também a geometria onde o **Phenomenon** é definido (diferentes objetos **Phenomenon** podem compartilhar uma mesma geometria ou parte da sua geometria); armazenam objetos **WeakForm**, os quais são ferramentas para cálculo de quantias e estão definidos em uma determinada parte da geometria alocada para um Phenomenon.

A definição da classe **Phenomenon** e os principais atributos e métodos estão descritos abaixo:

```
class Phenomenon
{
  protected:
    int code;
    Phenomenon *gPhen;
    SystData *stdt;
    vector<string> *expDt;
    vector<string> *reqDt;
    map<int, VectorField *> *vctrFields;
    map<int, PhenParameter*> *params;
    map<int, WeakForm *> *weakFrms;
    map<int, vector<WeakForm *> *> *execCodeSets;
    map<int, string> *execCodeDefs;
    map<int, GeomMesh *> *gMeshes;
    map<int, PhenMesh *> *pMeshes;
    GeomMeshGenerator *gMeshG;
    map<int, PhenMeshGenerator *> *pMeshG;
    IntegMethd *intgMth;
    geomGraph *gGraph;
    phenGraph *pGraph;

  public:
    Phenomenon();
    virtual ~Phenomenon();
    virtual int computeExecCode(int execCode);

};
```

- `Phenomenon *gPhen`: Definição dos objetos **Phenomenon**.
- `SystData *stdt`: É um objeto responsável em expor dados acessíveis às outras camadas e requerer dados das mesmas.

- `vector<string> *expDt`: É um vetor que contém as definições dos dados expostos pelo **Phenomenon** e que são acessíveis às outras camadas.
- `vector<string> *reqDt`: É um vetor que contém as definições dos dados requeridos pelo **Phenomenon** às outras camadas.
- `map<int, VectorField *> *vctrFields`: Mapeia os **id's** dos estados globais para o objeto **vectorField** no **Phenomenon**.
- `map<int, PhenParameter*> *params`: Mapeia o **id** do parâmetro para o objeto **Phenomenon**.
- `map<int, WeakForm *> *weakFrms`: Mapeia o **id** da **WeakForm** para ponteiro do seu objeto **WeakForm**.
- `map<int, vector<WeakForm *> *> *execCodeSets`: Mapeia um *ExecCode* para um vetor de ponteiros para objetos **WeakForm**.
- `map<int, string> *execCodeDefs`: Mapeia uma *ExecCode* para sua definição.
- `map<int, GeomMesh *> *gMeshes`: Mapeia o **id** do **GeomMesh** para seu objeto **GeomMesh**.
- `map<int, PhenMesh *> *pMeshes`: Mapeia o **id** do gerador de malha **PhenMesh** para seu objeto **PhenMesh**.
- `GeomMeshGenerator *gMeshG`: Ponteiro para o gerador de malhas geométricas.
- `map<int, PhenMeshGenerator *> *pMeshG`: Mapeia o **id** do gerador de malha do **Phenomenon** para o ponteiro do objeto **PhenMeshGenerator**.
- `IntegMethd *intgMth`: Ponteiro para o objeto que define o método de integração numérica.
- `geomGraph *gGraph`: É o ponteiro para o grafo que define a geometria.
- `virtual int computeExecCode(int execCode)`: Método responsável pela execução de um *ExecCode*.

A definição da classe **WeakForm** e seus principais atributos e métodos estão descritos abaixo:

```
class WeakForm{
  protected:
    int Code;
    string def;
    Phenomenon *phen;
```

```
    Phenomenon *gPhen;
    SystData *stdt;

  public:
    WeakForm();
    virtual ~WeakForm();
    virtual int computeWF();

};
```

- `string def:` Definição de **WeakForm**.
- `Phenomenon *phen:` É o ponteiro para o objeto **Phenomenon**, que é o dono das **WeakForm**.
- `Phenomenon *gPhen:` É o ponteiro para o objeto **gPhen**, que é o representante do **Group** no nível do **Phenomenon**.
- `SystData *stdt:` É um objeto responsável em expor e requerer dados.
- `virtual int computeWF():` É o método responsável em executar o objeto **WeakForm**.

2.5.1 Exemplo Elasto-plástico com Difusão

Seguem abaixo os cálculos no nível do **Phenomenon** (ver apêndice A) referente à simulação de um problema elasto-plástico com difusão, ou seja, um problema envolvendo elasticidade, plasticidade e difusão. A descrição de cada linha relacionada aos cálculos no nível do **Phenomenon** estão descritas no apêndice A.

Cálculos no nível do Phenomenon:

- **Phen0 - pertencente ao Group 0**

(ExecCode(0): computeWeakForm(0))

.
.
.

(ExecCode(10): computeWeakForm(0))

- **Phen1 - pertencente ao Group 1**

(ExecCode(0): computeWeakForm(0))

.
.
.

(ExecCode(14): computeWeakForm(1))

- **Phen2 - pertencente ao Group 2**

(ExecCode(0): computeWeakForm(0))

.
.
.

(ExecCode(2): computeWeakForm(1))

CAPÍTULO 3

Estrutura de Dados Geométricos

Neste Capítulo será mostrada toda a estrutura de dados para a malha geométrica e a malha do fenômeno. Serão explicitados também os elementos de referência e as funções de forma e suas peculiaridades.

3.1 Representação Geométrica

Nesta seção será mostrada uma representação geométrica, que possui pontos (estrutura de dimensão zero (OD)), curvas (estrutura de dimensão um (1D)), superfícies (estrutura de dimensão dois (2D)) e volumes (estrutura de dimensão três (3D)).

Considere a geometria (Figura 3.1), onde P_i são pontos para $i = 0,..,np$, C_j são curvas para $j = 0,..,nc$ e S_k são superfícies para $k = 0,..,ns$.

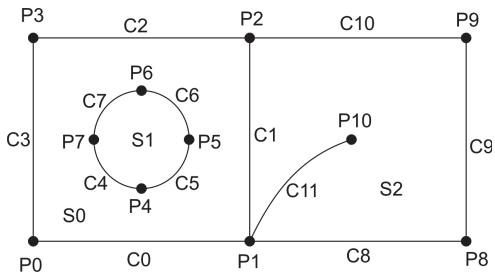


Figura 3.1 Geometria

3.1.1 Orientação dos entes geométricos

Os entes géometricos são orientados como visto em [19] em uma ordem predeterminada pela regra seguinte:

- **Para 0D**: Não possui orientação;
- **Para 1D**: A orientação é a partir da origem (1º filho) para o destino (2º filho), onde o índice do 2º filho é maior em relação ao 1º filho.
- **Para 2D-3D**: A orientação dos entes geométricos de dimensão 2 e imersos no espaço de dimensão 3 é da seguinte forma: se a orientação dos entes da face (0D e 1D) obedecer o sentido do vetor **n** (ver Figura 3.2), então a orientação é considerada positiva. O ente

de maior dimensão tem que ser regular e orientável, ou seja nenhuma derivada tangente vetorial pode ser zero.

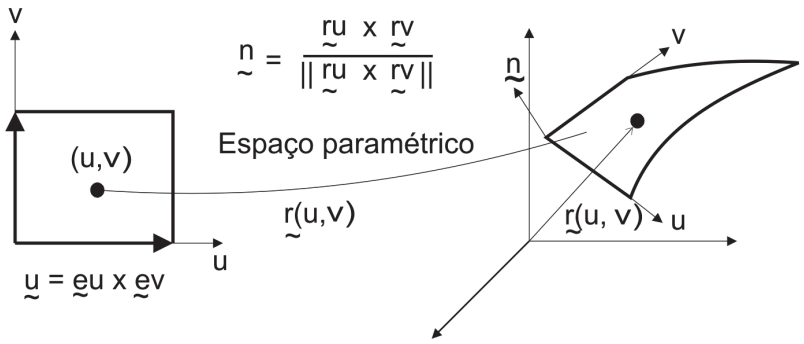


Figura 3.2 Orientação de entes 2D-3D

A Figura 3.3 representa o grafo da geometria - *GeomGraph*, onde o *GeomGraph* aponta para o *root* e o *root* aponta para os entes de maior dimensão da geometria (ver Figura 3.1), sendo que os entes de maior dimensão sempre aponta para seus filhos (uma dimensão menor).

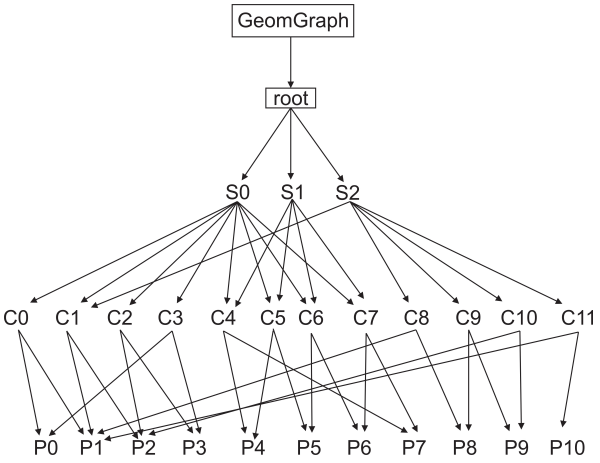


Figura 3.3 Grafo da geometria

Em uma geometria podem existir contornos internos, externos e isolados, onde:

- **Externos**: se percorrido no sentido anti-horário, os pontos materiais se localizam à direita;
- **Internos**: se percorrido no sentido anti-horário, os pontos materiais se localizam à esquerda;
- **Isolado**: não é propriamente um contorno, pois possui pontos materiais da superfície em ambos os lados

Segue abaixo a especificação de cada contorno em relação a dimensão da geometria 0D, 1D, 2D e 3D:

- **Para 0D**: Nenhum contorno (interior vazio);
- **Para 1D**: Contorno externo e isolado;
- **Para 2D**: Contorno externo, interno e isolado;
- **Para 3D**: Contorno externo, interno e isolado.

Nas superfícies da geometria (Figura 3.1) apresentam os seguintes contornos:

- Na superfície $S0$, tem-se:
  - **Contorno interno**: representado pelas curvas $C4$, $C5$, $C6$ e $C7$;
  - **Contorno externo**: representado pelas curvas $C0$, $C1$, $C2$ e $C3$.
- Na superfície $S1$, tem-se:
  - **Contorno externo**: representado pelas curvas $C4$, $C5$, $C6$ e $C7$.

  Na superfície $S2$, tem-se:
  - **Contorno externo**: representado pelas curvas $C1$, $C8$, $C9$ e $C10$;
  - **Contorno isolado**: representado pelas curvas $C11$.

3.1.2 Compartilhamento da geometria

Para haver compartilhamento de malhas entre entes geométricos, a geometria tem que ser fornecida para o fenômeno através de cópias, ou seja, dois ou mais fenômenos podem compartilhar malhas em entes geométricos comuns.

Em geometria onde possui mais que um fenômeno, é estabelecida uma geometria para cada fenômeno (ver Figura 3.4), onde ocorre cópias dos entes comuns aos fenômenos. Há uma quebra na ligação dos entes em relação a geometria geral, onde os entes que são copiados, os seus filhos também serão copiados, mas os seus pais não serão copiados.

Considere a atuação de três fenômenos PhenA, PhenB e PhenC na geometria (Figura 3.4):

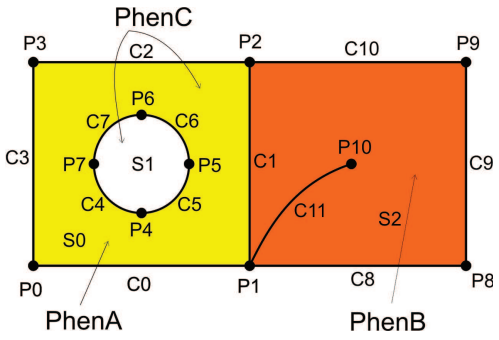


Figura 3.4 Geometria compartilhada

Na Figura 3.5 tem-se o grafo da geometria do PhenA:

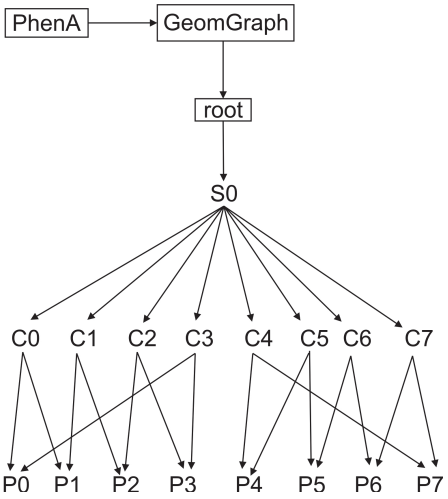


Figura 3.5 Grafo da geometria do PhenA

Na Figura 3.6 tem-se o grafo da geometria do PhenB:

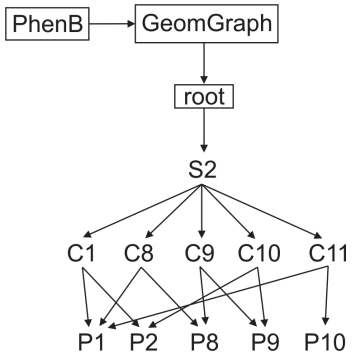


Figura 3.6 Grafo da geometria do PhenB

Na Figura 3.7 tem-se o grafo da geometria do PhenC:

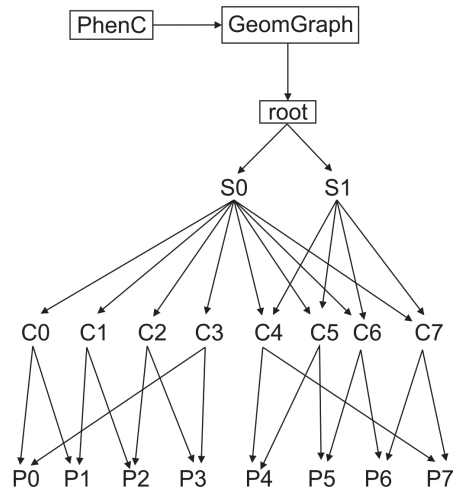


Figura 3.7 Grafo da geometria do PhenC

3.2 Malha Geométrica

3.2.1 Representação topológica da malha geométrica

Uma nova estrutura de dados topológica é proposta para representar a malha do elemento finito. Esta estrutura de dados representa somente entidades explícitas e topologicamente referenciadas. Para entidades topológicas de dimensão três (3D) são são volumes, faces, arestas e vértices; de dimensão dois (2D) são faces, arestas e vértices; dimensão um (1D) são arestas e vértices, onde:

- **Vértices:** Definidos por pontos;
- **Arestas:** Segmentos fechados de reta que une dois vértices;
- **Faces:** Regiões fechadas e definidas por arestas em seu contorno (triângulos, quadriláteros, etc);
- **volumes:** Regiões fechadas e definidas por faces (tetraedro, hexaedro, etc).

Para uma representação topológica ser considerada suficiente, ela tem que conter informação necessária, a fim de que relações topológicas entre entidades topológicas possam ser recuperadas com ordem de complexidade 1.

Nossa malha geométrica é do tipo afim, onde seus entes: arestas (1D), são formadas por segmento de reta; triângulos (2D), são circundados por arestas, que são segmentos de retas; e tetraedros (3D), são circundados por faces que contém arestas, que são segmentos de retas.

Esta malha está representada da seguinte forma:

- **StrutDim:** É a dimensão da estrutura podendo ser 0D, 1D, 2D ou 3D.
- **SpaceDim:** É a dimensão do espaço no qual a malha geométrica está imersa.
- **Coords:** São as coordenadas dos vértices da malha geométrica.
- **GeomEntity:** Entes geométricos aos quais se refere a malha, ou seja, para os quais a malha foi gerada.
- **Relação topológica:** É a relação entre entes da malha geométrica.

3.2.2 Malha geométrica de dimensão 1

- **Malha:** A malha geométrica de dimensão um (1D) (Figura 3.8) é composta de vértices e arestas, sendo que os vértices e as arestas são identificados pelos seus **id's** (números inteiros). Estes números iniciam em zero e não possuem lacunas.

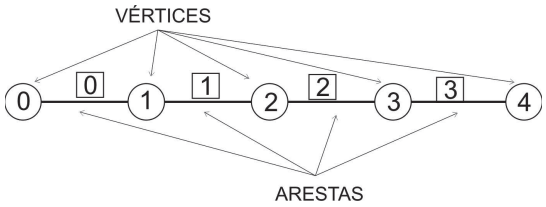


Figura 3.8 Malha - 1D.

- **Elemento:** Esta malha possui elementos de dimensão 1D, que são arestas (Ek) (Figura 3.9). E estes elementos referem-se aos vértices seguindo uma ordem pré-estabelecida (ver seção 3.4):

Figura 3.9 Elemento - 1D.

3.2.3 Malha geométrica de dimensão 2

- **Malha 2D:** A malha geométrica de dimensão dois (2D) [20] (Figura 3.10) é composta de vértices, arestas e faces, sendo que os vértices, as arestas e as faces são identificados pelos seus **id's** (números inteiros). Esta numeração inicia em zero e não possui lacunas.

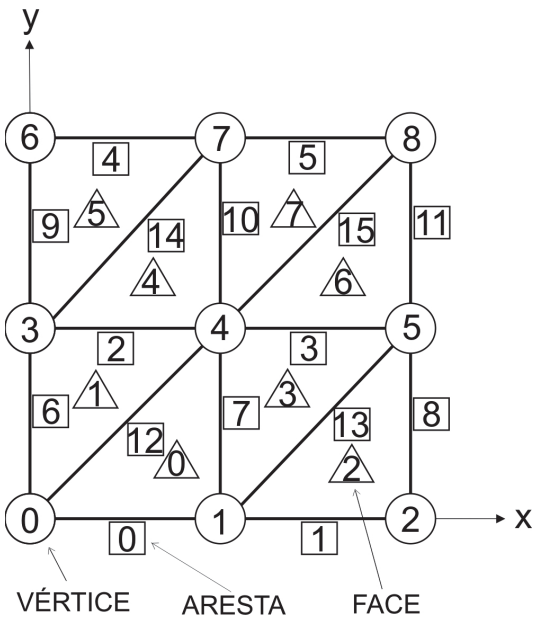


Figura 3.10 Malha - 2D.

- **Elemento:** Esta malha possui elementos de dimensão 2D, que são faces (Fk) (Figura 3.11). E estes elementos referem-se aos vértices e arestas seguindo uma ordem pré-estabelecida (ver seção 3.4):

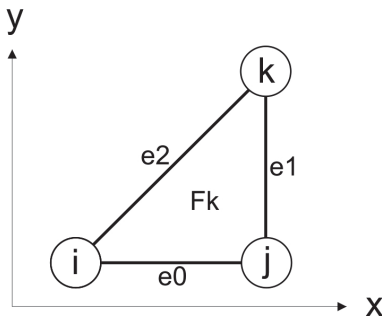


Figura 3.11 Elemento - 2D.

3.2.4 Malha geométrica de dimensão 3

- **Malha:** A malha geométrica de dimensão três (3D) [21] (Figura 3.12) é composta de vértices, arestas, faces e volumes (tetraedro ou hexaedro), sendo que os vértices, as arestas, as faces e os volumes são identificados pelos seus **id's** (números inteiros). Estes números iniciam em zero e não possuem lacunas.

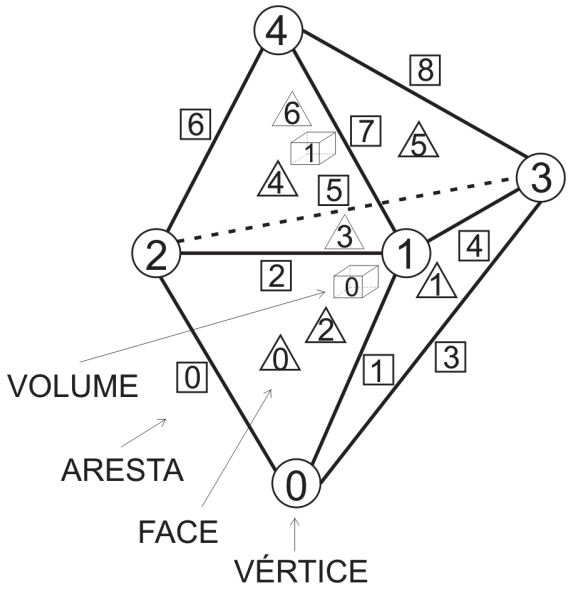


Figura 3.12 Malha - 3D.

- **Elemento:** Esta malha possui elementos de dimensão 3D, que são volumes (Vk) (Figura 3.13). E estes elementos referem-se aos vértices, arestas e faces seguindo uma ordem pré-estabelecida (ver seção 3.4):

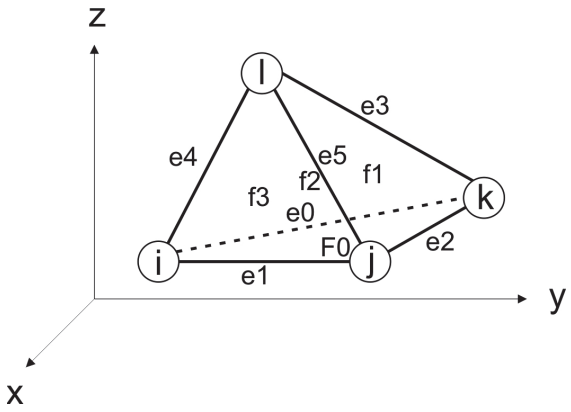


Figura 3.13 Elemento - 3D (Tetraedro).

3.2.5 Relações topológicas da malha geométrica

São relações de vizinhança entre entes da malha (vértices, arestas, faces e volumes). Existem dois tipos de relações: primárias e secundárias. As primárias são aquelas que são estabelecidas no código do gerador de malhas, e as secundárias são implementadas através do encadeamento das relações primárias.

1. **Relações primárias**:

   (a) **Vértice - Arestas**: É a relação de um vértice com as arestas que contêm este vértice;

   (b) **Aresta - Vértices**: É a relação de uma aresta com os vértices pertencentes a esta aresta;

   (c) **Aresta - Faces**: É a relação de uma aresta com as faces que contêm esta aresta;

   (d) **Face - Arestas**: É a relação de uma face com as arestas pertencentes a esta face;

   (e) **Face - Volumes**: É a relação de uma face com os volumes que contêm esta face;

   (f) **Volume - Faces**: É a relação de um volume com as faces pertencentes a este volume.

2. **Relações secundárias**:

   (a) **Vértice - Vértices**: É a relação de um vértice com os vértices pertencentes às arestas que contém este vértice;

   (b) **Aresta - Arestas**: É a relação de uma aresta com as arestas pertencentes às faces que contêm esta aresta;

   (c) **Vértice - Faces**: É a relação de um vértice com as faces que contêm este vértice;

   (d) **Face - Vértices**: É a relação de uma face com os vértices pertencentes a esta face;

   (e) **Face - Faces**: É a relação de uma face com as faces que contêm uma das arestas pertencentes a esta face, ou seja, as faces vizinhas;

   (f) **Vértice - volumes**: É a relação de um vértice com os volumes que contêm este vértice;

   (g) **Volume - Vértices**: É a relação de um volume com os vértices pertencentes a este volume;

   (h) **Aresta - Volumes**: É a relação de uma aresta com os volumes que contêm esta aresta;

   (i) **Volume - Arestas**: É a relação de um volume com as arestas pertencentes a este volume;

   (j) **Volume - Volumes**: É a relação de um volume com os volumes que contêm uma das faces pertencente a este volume, ou seja, volumes vizinhos.

3.3 Elemento de Referência:

O elemento de referência é um elemento fixo e possui os mesmos entes de um elemento real, porém está definido em um espaço de coordenadas especiais (ξ, η, ζ). Eles são utilizados na definição das funções de forma (ver seção 3.5) que serão utilizadas no cálculo dos campos vetoriais (ver seção 3.8) e nos processos de integração numérica.

Seguem abaixo exemplos de elemento de referência:

- Para "1D"(Figura 3.14):

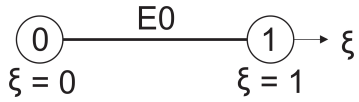


Figura 3.14 Elemento de referência - 1D.

- Para "2D"(Figura 3.15):

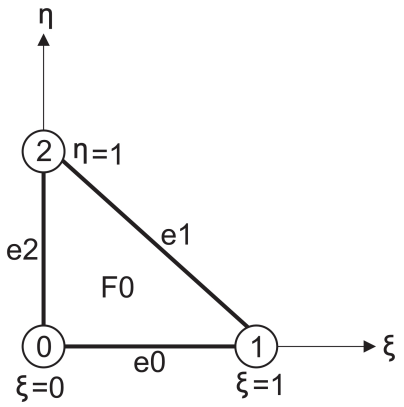


Figura 3.15 Elemento de referência - 2D.

- Para "3D"(Figura 3.16):

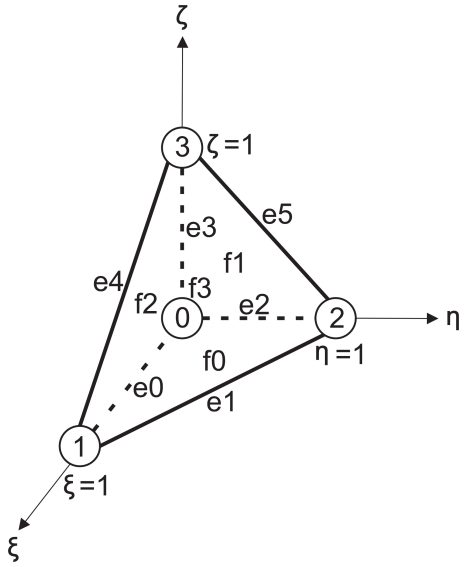


Figura 3.16 Elemento de referência - 3D.

3.3.1 Mapeamento do elemento real no elemento de referência

A geometria do elemento de referência é mapeada a partir da geometria do elemento real usando expressões de transformação geométrica, a fim de obter as seguintes propriedades:

- O mapeamento deve ser uma função bijetora, ou seja, para qualquer ponto do elemento de referência, há um e somente um ponto do elemento real, e vice-versa;
- Os vértices do elemento de referência correspondem aos vértices do elemento real;
- Qualquer parte do contorno do elemento de referência, definido pelos vértices deste contorno, correspondem a uma parte do contorno do elemento real definido pelos vértices correspondentes.

Como nossa malha geométrica é afim, o mapeamento do elemento real no elemento de referência fica fácil, pois serão mapeados tetraedros em tetraedros, triângulos em triângulos e segmentos de retas em segmentos de retas.

As equações do mapeamento são:

$$\hat{\mathbf{x}}(\xi) = \mathbf{B} \cdot \xi + \mathbf{x_i}$$

$$\hat{\xi}(x) = \mathbf{B}^{-1} \cdot (\mathbf{x} - \mathbf{x_i})$$

$$\nabla_x(.) = \mathbf{B}^{-T} \cdot \nabla_\xi$$

onde $\mathbf{B} = \frac{\partial \hat{\mathbf{x}}}{\partial \xi}$

Seguem abaixo exemplos de mapeamento do elemento real no elemento de referência para 1D, 2D e 3D (Figuras 3.17, 3.18 e 3.19):

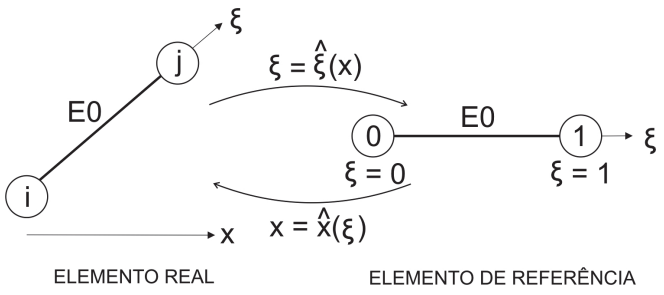


Figura 3.17 Mapeamento do elemento real no elemento de referência - 1D.

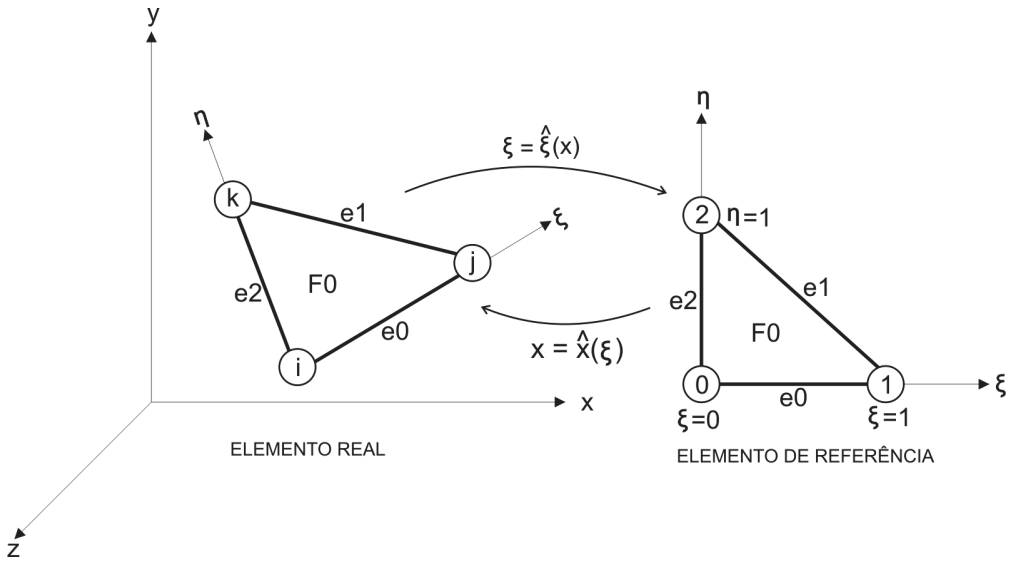


Figura 3.18 Mapeamento do elemento real no elemento de referência - 2D.

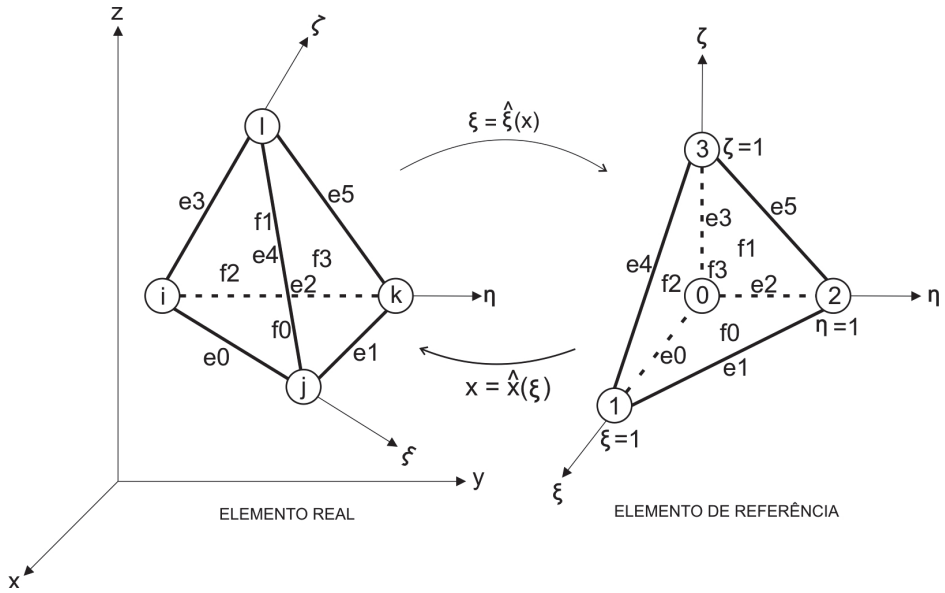


Figura 3.19 Mapeamento do elemento real no elemento de referência - 3D.

3.4 Orientação dos entes da malha

Nesta seção iremos utilizar a notação vista em [22], onde:

- **Ek <i, j>**: Ek é o índice da aresta; i e j são os índices dos vértices pertencentes à aresta Ek nesta ordem;
- **Fk <e0, e1, e2>**: Fk é o índice da face; e0, e1 e e2 são os índices das arestas pertencentes à face Fk nesta ordem;
- **Fk <i, j, k>**: Fk é o índice da face; i, j e k são os índices dos vértices pertencentes à face Fk nesta ordem;
- **Vk <f0, f1, f2, f3>**: Vk é o índice do volume; f0, f1, f2 e f3 são os índices das faces pertencentes ao tetraedro Vk nesta ordem;
- **Vk <e0, e1, e2, e3, e4, e5>**: Vk é o índice do volume; e0, e1, e2, e3, e4 e e5 são os índices das arestas pertencentes ao tetraedro Vk nesta ordem;
- **Vk <i, j, k, l>**: Vk é o índice do volume; i, j, k e l são os índices dos vértices pertencentes ao tetraedro Vk nesta respectiva ordem.

É necessário definir uma orientação global (a mesma do elemento de referência) para as arestas e faces dos elementos da malha, a fim de garantir a unicidade das funções básicas (polinomiais). Existem dois tipos de orientação: orientação $o = 1$ (um) se for orientação global e orientação $o = 0$ (zero) se a orientação for oposta a global.

3.4.1 Orientação dos entes da malha 1D

Cada aresta dos elementos da malha possui uma orientação global, ou seja, as arestas são orientadas de acordo com a numeração de seus vértices sendo da menor numeração (índice) para a maior numeração (índice) (ver Figura 3.20):

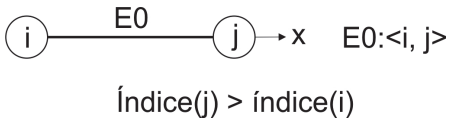


Figura 3.20 Orientação da aresta

3.4.2 Orientação dos entes da malha 2D

Cada face é orientada em relação as suas arestas e seus vértices (Figura 3.21), e possui uma orientação global. Se a orientação for pelas arestas, começa-se pelo menor índice e os demais seguem no sentido anti-horário, ou seja, obedecendo o sentido da regra da mão direita onde o polegar representa o vetor nornal, e similarmente para os vértices.

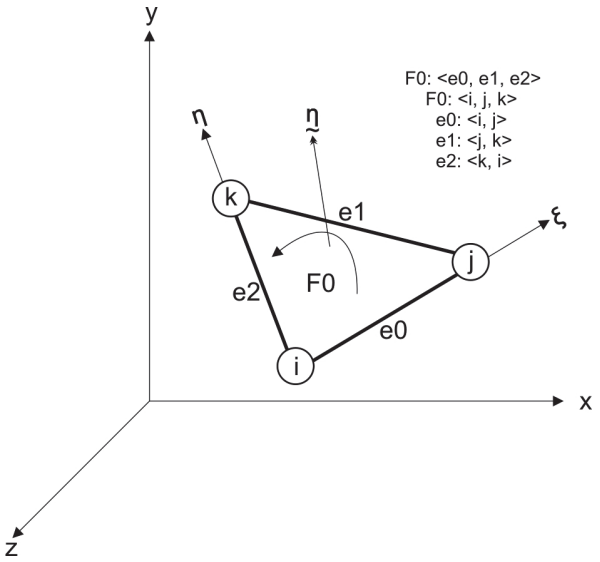


Figura 3.21 Orientação da face

Para uma face triangular existem seis possibilidades de orientações, onde apenas 3 destas obedecem a orientação global e 3 possuem orientações opostas a orientação global (Figura 3.22).

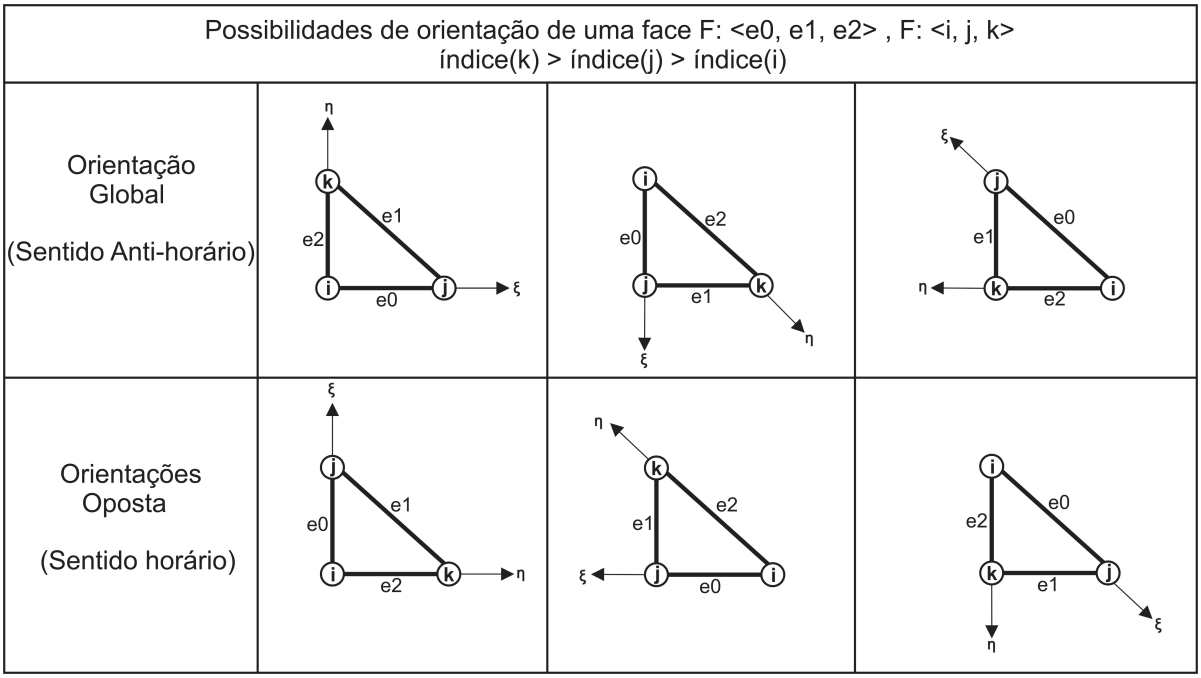
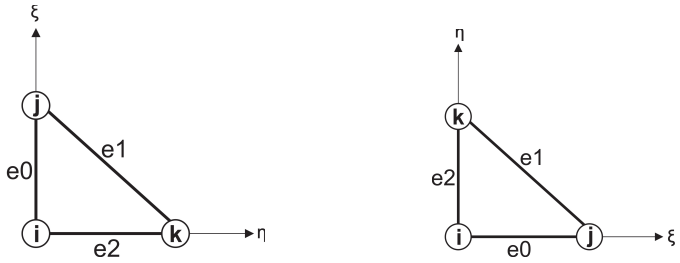


Figura 3.22 Orientação da face

Considere a Figura 3.23(a), onde tem-se a orientação oposta a orientação global predefinida. Para obter a orientação global existe uma rotina no MPhysCas que realiza uma permutação dos vértices, ou seja, uma troca do vértices com intuito de obter a orientação desejada. Na Figura 3.23(b) tem-se um exemplo em que a orientação global é obtida através da permutação dos vértices (j, k).



(a) Orientação Oposta (b) Orientação Global

Figura 3.23 Orientações de uma face

3.4.3 Orientação dos entes da malha 3D

Os tetraedros são orientados em relação as suas faces, arestas e vértices (Figura 3.24), a orientação começa pelo menor índice, tanto para os vértice, arestas e faces. A orientação do tetraedro em relação as suas arestas e seus vértices obedece ao sentido da regra da mão direita, sendo o polegar representando o vetor normal apontado para o interior do tetraedro, ou tomando-se como o referencial o observador no centro do tetraedro, a orientação é sempre no sentido anti-horário. Já a orientação em relação a face começa-se a partir no menor índice seguido dos índices das faces pertencentes às arestas da face inicial já orientada.

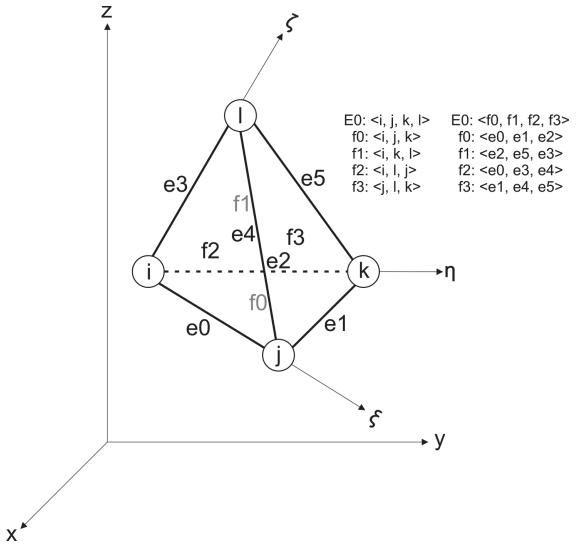


Figura 3.24 Orientação do tetraedro

3.5 Funções de forma

Nesta seção descreveremos as funções de forma que serão utilizadas mais adiante para definir campos vetorias (por exemplo temperatura, deslocamento, etc). Estas funções estão definidas nos elementos de referência e utilizam as relações do elemento de referência com o elemento real para que os campos vetoriais possam ser definidos no espaço real.

As funções de forma utilizadas no cálculo das quantias (dos fenômenos) são relacionadas bi-univocamente com os nós do elemento. Estes nós podem ser relacionados aos vértices, às arestas, às faces e aos volumes do elemento. Sendo assim, as funções de forma também serão ordenadas com os seus subconjuntos dos vértices, das arestas, das faces e do volume.

As funções de forma utilizadas nas simulaçãoes do MPhyScas são do tipo hierárquicas, ou seja, são aquelas em que as funções de interpolação de grau variável nos elementos se faz conservando inalteradas as funções de interpolação anteriores (de menor grau).

Será utilizada a seguinte notação para as funções de forma: ψ_{ab}, onde $a = c$ para vértices, $a = r$ para arestas, $a = f$ para faces e $a = v$ para volumes, $b = 0, ..., (g-1)$, g é o número de funções relacionadas àquele ente do elemento.

3.5.1 Funções de forma para elementos da malha 1D

- **Funções de forma de vértice**: As funções de forma (dos vértices) vista em [23] correspondem as funções polinomiais de grau 1. Estas funções possuem valor 1 em um vértice e zero no outro vértice. Elas são definidas como:

$$\psi_{c0}(\xi) = 1 - \xi$$
$$\psi_{c1}(\xi) = \xi$$

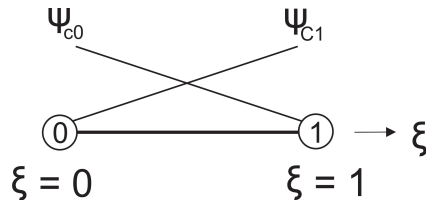


Figura 3.25 Funções de forma dos vértices de um elemento da malha 1D

O número de funções de forma hierárquicas dos vértices para elementos da malha 1D é:

$$Nfv = 2$$

- **Funções de forma de aresta**: As funções de forma das arestas vistas em [23], como o nome já diz, estão associadas a uma aresta. Estas funções são iguais a zero em todos os vértices. As funções das arestas são definidas como:

$$\psi_{r0(n)}(\xi) = \psi_{c0}(\xi)\psi_{c1}(\xi)f_n(\xi);$$

Onde $f_n(\xi)$ é um conjunto de funções escolhidas pelo usuário (por exemplo, funções polinomiais de lagrange, legendre, entre outras. (ver subseção 3.5.4)), que irão formar uma ordem polinômial de aproximação do elemento finito, sendo $\xi \in [\ 0,\ 1\ ]$ e $n = 0, ..., p_a - 2$, onde p_a é a ordem polinomial relacionada à aresta.

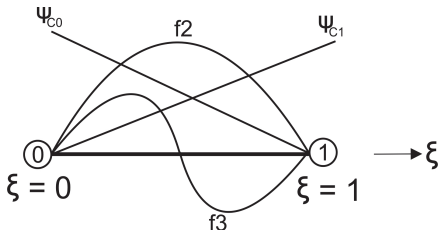


Figura 3.26 Funções de forma da aresta de um elemento da malha 1D

3.5.2 Funções de forma para elementos da malha 2D

- **Funções de forma de vértice**: Para o triângulo as funções de forma (dos vértices) vista em [23] corresponde as funções polinomiais de grau 1. Estas funções possuem valor 1 em um vértice e zero em todos os outros vértices. Elas são polinômios de grau 1 sobre todas as arestas e diferentes de zero somente sobre as arestas e faces ligadas ao vértice que esta função é zero. Estas funções formam uma base para o espaço dos polinômios de grau ≥ 1 no triângulo. Elas são definidas como:

$$\psi_{c0}(\xi, \eta) = 1 - \xi - \eta;$$
$$\psi_{c1}(\xi, \eta) = \xi;$$
$$\psi_{c2}(\xi, \eta) = \eta.$$

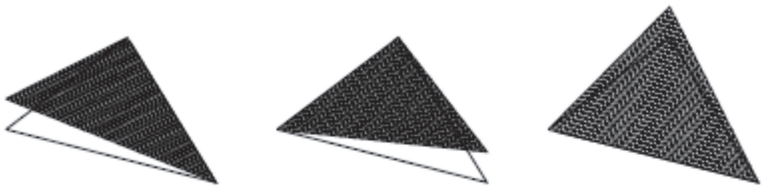


Figura 3.27 Funções de forma dos vértices pertencente a face triângulo

O número de funções de forma hierárquicas dos vértices de um triângulo é:

$$Nfv = 3$$

- **Funções de forma de aresta**: As funções de forma das arestas do triângulo vistas em [23], como o nome já diz, estão associadas a uma aresta. Estas funções são iguais a zero em todos os vértices da face a qual a determinada aresta está associada. As funções das arestas são definidas como:

$$\psi_{r0(n)}(\xi,\eta) = \psi_{c0}\psi_{c1}f_n(\psi_{c1}-\psi_{c0});$$
$$\psi_{r1(n)}(\xi,\eta) = \psi_{c1}\psi_{c2}f_n(\psi_{c2}-\psi_{c1});$$
$$\psi_{r2(n)}(\xi,\eta) = \psi_{c2}\psi_{c0}f_n(\psi_{c0}-\psi_{c2}).$$

Onde f_n é um conjunto de funções escolhidas pelo usuário (por exemplo, funções polinomiais de lagrange, legendre, entre outras. (ver subseção 3.5.4)), sendo $n = 0,...,p_a-2$, onde p_a é a ordem polinomial relacionada à aresta.

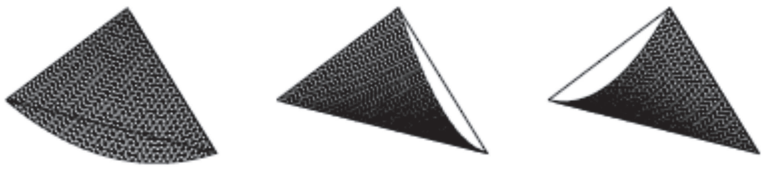


Figura 3.28 Funções de forma das arestas da face triângulo

O número de funções de forma hierárquicas das arestas de um triângulo é:

$$Nfa = 3*(p_a-1)$$

- **Funções de forma de face**: As funções de forma das faces do triângulo vistas em [23], como o nome já diz estão associadas a uma face. Elas são iguais a zero em todas as arestas e vértices e são diferentes de zero somente na face associada. As funções das faces são definidas como:

$$\psi_{f0(n1,n2)}(\xi,\eta) = \psi_{c0}\psi_{c1}\psi_{c2}f_{n1}(\psi_{c1}-\psi_{c0}-\psi_{c2})f_{n2}(\psi_{c2}-\psi_{c0}-\psi_{c1});$$

Para $0 \leq n1+n2 \leq p_f-3$. Onde p_f é a ordem polinomial relacionada à face, sendo $p_f \geq 3$.
Onde f_{n1} e f_{n2} são conjuntos de funções escolhidas pelo usuário (por exemplo, funções polinomiais de lagrange, legendre, entre outras. (ver subseção 3.5.4)).

O número de funções de forma hierárquicas da face triângulo é:

$$Nff = \frac{(p_f-2)*(p_f-1)}{2}$$

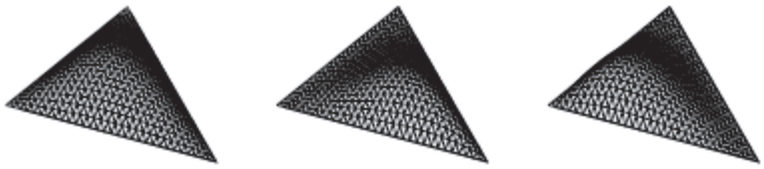


Figura 3.29 Funções de forma da face triângulo

3.5.3 Funções de forma para elementos da malha 3D

- **Funções de forma de vértice**: Para o tetraedro as funções de forma (dos vértices) vista em [23] corresponde as funções polinomiais de grau 1. Estas funções possuem valor 1 em um vértice e zero em todos os outros vértices. Elas são polinômios de grau 1 sobre todas as arestas e diferentes de zero somente sobre as arestas e faces ligadas ao vértice onde esta função é zero. Estas funções formam uma base para o espaço dos polinômios de grau ≥ 1 no tetraedro. Elas são definidas como:

$$\psi_{c0}(\xi,\eta,\zeta) = 1-\xi-\eta-\zeta;$$
$$\psi_{c1}(\xi,\eta,\zeta) = \xi;$$
$$\psi_{c2}(\xi,\eta,\zeta) = \eta;$$
$$\psi_{c3}(\xi,\eta,\zeta) = \zeta.$$

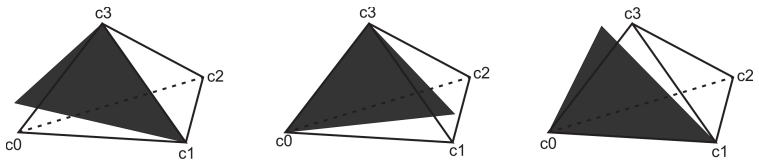


Figura 3.30 Funções de forma dos vértices pertencente a face Fk de um tetraedro

O número de funções de forma hierárquicas dos vértices de um tetraedro é:

$$Nfv = 4$$

- **Funções de forma de aresta**: As funções de forma das arestas do tetraedro vistas em [23], como o nome já diz, estão associadas a uma aresta. Estas funções são zero em todos os vértices da face a qual a determinada aresta está associada e também nas outras arestas não pertencentes à face referida. Elas são diferentes de zero somente sobre as arestas em que estão associadas e sobre as faces as quais estão ligadas e também são diferentes de zero sobre o volume. As funções das arestas são definidas como:

$$\psi_{r0(n)}(\xi,\eta,\zeta) = \psi_{c0}\psi_{c1}f_n(\psi_{c1}-\psi_{c0});$$
$$\psi_{r1(n)}(\xi,\eta,\zeta) = \psi_{c1}\psi_{c2}f_n(\psi_{c2}-\psi_{c1});$$

$$\psi_{r2(n)}(\xi,\eta,\zeta) = \psi_{c2}\psi_{c0}f_n(\psi_{c0}-\psi_{c2});$$
$$\psi_{r3(n)}(\xi,\eta,\zeta) = \psi_{c0}\psi_{c3}f_n(\psi_{c3}-\psi_{c0});$$
$$\psi_{r4(n)}(\xi,\eta,\zeta) = \psi_{c1}\psi_{c3}f_n(\psi_{c3}-\psi_{c1});$$
$$\psi_{r5(n)}(\xi,\eta,\zeta) = \psi_{c2}\psi_{c3}f_n(\psi_{c3}-\psi_{c2}).$$

Onde f_n é um conjunto de funções escolhidas pelo usuário (por exemplo, funções polinomiais de lagrange, legendre, entre outras. (ver subseção 3.5.4)), sendo $n = 0,...,p_a-2$, onde p_a é a ordem polinomial relacionada à aresta.

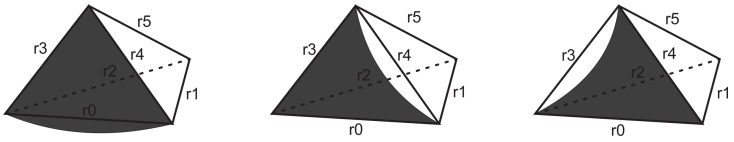


Figura 3.31 Funções de forma das arestas da face Fk de um tetraedro

O número de funções de forma hierárquicas das arestas de um tetraedro é:

$$Nfa = 6*(p_a-1)$$

- **Funções de forma de face**: As funções de forma das faces do tetraedro vistas em [23] estão associadas a uma face do elemento. Elas são iguais a zero em todas as arestas e vértices e são diferentes de zero somente nas faces associadas e sobre o volume. As funções das faces são definidas como:

$$\psi_{f0(n1,n2)}(\xi,\eta,\zeta) = \psi_{c0}\psi_{c1}\psi_{c2}f_{n1}(\psi_{c1}-\psi_{c0}-\psi_{c2}-\psi_{c3})f_{n2}(\psi_{c2}-\psi_{c0}-\psi_{c1}-\psi_{c3});$$
$$\psi_{f1(n1,n2)}(\xi,\eta,\zeta) = \psi_{c0}\psi_{c2}\psi_{c3}f_{n1}(\psi_{c2}-\psi_{c0}-\psi_{c1}-\psi_{c3})f_{n2}(\psi_{c3}-\psi_{c0}-\psi_{c1}-\psi_{c2});$$
$$\psi_{f2(n1,n2)}(\xi,\eta,\zeta) = \psi_{c0}\psi_{c1}\psi_{c3}f_{n1}(\psi_{c1}-\psi_{c0}-\psi_{c2}-\psi_{c3})f_{n2}(\psi_{c3}-\psi_{c0}-\psi_{c1}-\psi_{c2});$$
$$\psi_{f3(n1,n2)}(\xi,\eta,\zeta) = \psi_{c1}\psi_{c2}\psi_{c3}f_{n1}(\psi_{c2}-\tfrac{1}{3}\psi_{c0}-\psi_{c1}-\psi_{c3})f_{n2}(\psi_{c3}-\tfrac{1}{3}\psi_{c0}-\psi_{c1}-\psi_{c2}).$$

Para $0 \leq n1+n2 \leq p-3$. Onde p_f é a ordem polinomial relacionada à face, sendo $p_f \geq 3$.

Onde f_{n1} e f_{n2} são conjuntos de funções escolhidas pelo usuário (por exemplo, funções polinomiais de lagrange, legendre, entre outras. (ver subseção 3.5.4)).

O número de funções de forma hierárquicas das faces de um tetraedro é:

$$Nff = 4*\frac{(p_f-2)*(p_f-1)}{2}$$

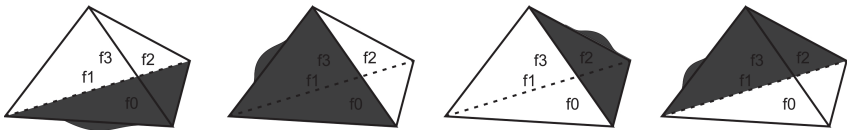


Figura 3.32 Funções de forma das faces do tetraedro

- **Funções de forma de volume**: As funções de forma do volume (tetraedro) vistas em [23] são diferentes de zero somente no interior do elemento e são iguais a zero em todas as faces, arestas e vértices. Estas funções são definidas como:

  $\psi_{v0(n1,n2,n3)}(\xi,\eta,\zeta) = \psi_{c0}\psi_{c1}\psi_{c2}\psi_{c3}f_{n1}(\psi_{c1}-\psi_{c0}-\psi_{c2}-\psi_{c3})f_{n2}(\psi_{c2}-\psi_{c0}-\psi_{c1}-\psi_{c3})f_{n3}(\psi_{c3}-\psi_{c0}-\psi_{c1}-\psi_{c2})$.

  Para $0 \leq n1+n2+n3 \leq p_v-4$, onde p_v é a ordem polinomial relacionada ao volume e $p \geq 4$.

  Onde f_{n1} e f_{n2} e f_{n3} são conjuntos de funções escolhidas pelo usuário (por exemplo, funções polinomiais de lagrange, legendre, entre outras. (ver subseção 3.5.4)).

  O número de funções de forma hierárquicas dos volumes é:

$$Nfv = \frac{(p_v-3)*(p_v-2)*(p_v-1)}{6}$$

3.5.4 Exemplos de polinômios de Legendre

Nas figuras 3.33, 3.34, 3.35 e 3.36, têm-se alguns exemplos de polinômios de legendre, vistos em [24]:

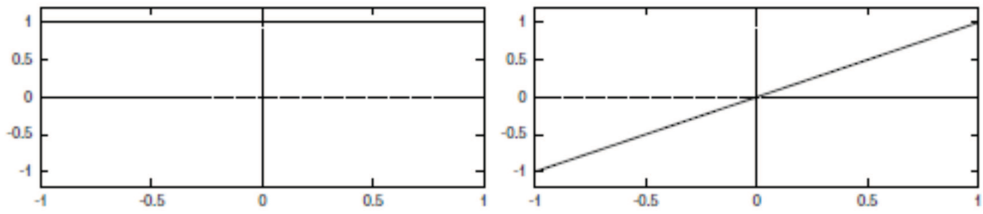


Figura 3.33 Polinômios de Legendre de grau 0 e 1

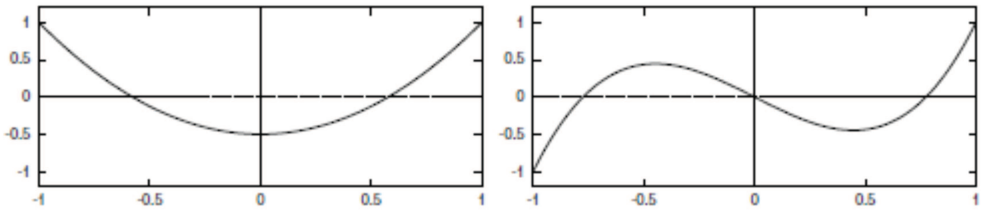


Figura 3.34 Polinômios de Legendre de grau 2 e 3

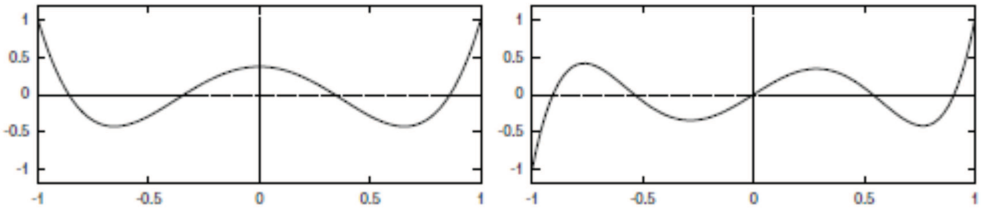


Figura 3.35 Polinômios de Legendre de grau 4 e 5

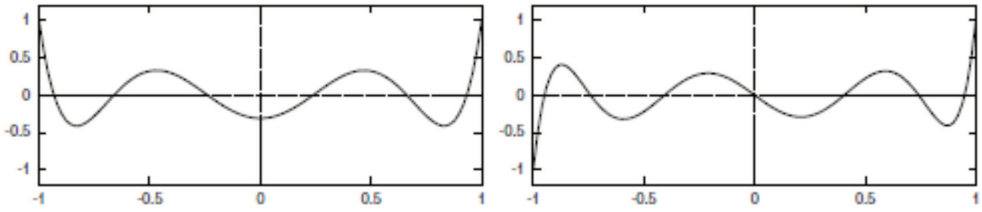


Figura 3.36 Polinômios de Legendre de grau 6 e 7

3.6 Compatibilização das funções de forma em relação as orientações dos entes da malha

Nesta seção será apresentada uma forma relativamente simples de compatibilizar funções de forma de um determinado elemento da malha, sendo que para isto utilizou-se a metodologia vista em [25].

As orientações dos entes da malha apresentadas na seção 3.4 são importantes, uma vez que se houver diferença de orientação de um ente em relação a orientação especificada no elemento de referência (orientação global), esta orientação afetará diretamente nas equações de algumas funções de forma definidas no elemento em questão. As modificações necessárias para solucionar este problema são simples de serem implementadas.

Segue abaixo exemplo de uma função de grau par e uma função de grau ímpar (Figura 3.37):

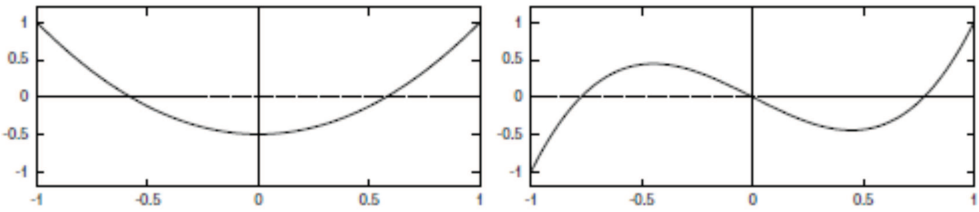


Figura 3.37 Polinômios de Legendre de grau 2 e 3

Observe que a função de grau par (ver Figura 3.37) é simétrica, ou seja, se próximo a um vértice ela possuir valor positivo, então próximo ao outro vértice também irá possuir valor positivo, por isso a orientação do ente em questão torna-se irrelevante para a função de forma.

Agora observe que a função de grau ímpar (ver Figura 3.37) não é simétrica, ou seja, se próximo a um vértice esta função possuir valor positivo, então próximo ao outro vértice esta função pussui valor negativo, ou vice-versa. Então a orientação de cada ente é importante para as funções f_n (ver seção 3.5), pois poderá acarretar modificações nas funções de forma.

3.6.1 Transformação das funções de forma da aresta

Considere a aresta $e' \subset \partial e_g$ com a seguinte orientação $e' < v_i', v_j' >$ e a aresta $e \subset \partial \hat{e}_g$ com a seguinte orientação $e < v_i, v_j >$. A aresta e é equipada por uma flag $o(e) = \{1,0\}$ como visto em [25], isto significa que ela possui orientação igual $(o(e) = 1)$ ou oposta $(o(e) = 0)$ a orientação estabelecida pelo elemento de referência (orientação predefinida). Para expressão abaixo considere $x(v_m)$, para $m = i, j$ sendo as coordenadas dos vértices e que $\hat{x} : \hat{e}_g \to e_g$ é o mapeamento afim do elemento geométrico de referência $\hat{e}_g$ para o elemento geométrico real e_g, então:

$$o(e) = \begin{cases} 1, & \text{se} \quad \hat{x}(x(v_i)) = x(v_i'); \ \hat{x}(x(v_j)) = x(v_j'). \\ 0, & \text{se} \quad \hat{x}(x(v_i)) = x(v_j'); \ \hat{x}(x(v_j)) = x(v_i'). \end{cases}$$

Considere duas faces vizinhas F'_k e F_k (ver Figura 3.38), sendo que a face F'_k enxerga a aresta e da seguinte forma $e' < v'_i, v'_j >$ e a face F_k enxerga a aresta e da seguinte forma $e^k < v_i, v_j >$, a aresta e é comun a duas fazes vizinhas. Neste caso onde uma aresta possui orientação diferente da orientação predefinida, será necessário um ajuste nas funções de forma da aresta.

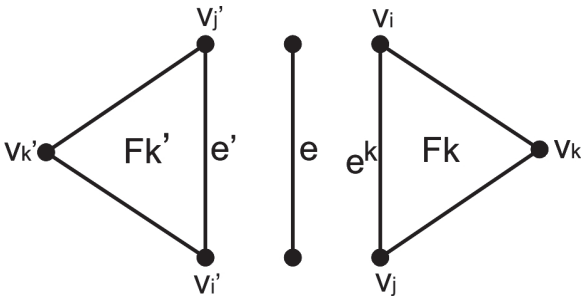


Figura 3.38 Orientação da aresta em relação a face

Se $o(e) = 1$, todas as funções de forma das arestas permanecem inalteradas. Caso contrário teremos que transformá-las através de uma função bijetora, ou seja, o mapeamento $\hat{x}^o_e : \hat{e}_g \to \hat{e}_g$, o qual inverte a parametrização da aresta e. Isto é realizado de forma bastante simples como se segue: Considere o mapeamento idêntico $I_t : e_g \to e_g$ visto em [25], o qual pode ser expresso como uma combinação linear das funções dos vértices com as coordenadas dos vértices:

$$\mathrm{I_t}(\xi,\eta) = \sum_{k=0}^{2} \psi_{ck}(\xi,\eta)\ \mathrm{x(v_k)};$$

Com isto obtemos a função bijetora $\hat{x}^o_e$ permutando as funções dos vértices v_i e v_j da aresta e na equação acima. Por exemplo, a expressão do mapeamento $\hat{x}^o_e$, após a permutação das funções $\psi_{c0}(\xi,\eta)$ e $\psi_{c1}(\xi,\eta)$ fica da seguinte forma:

$$\mathrm{\hat{x}^o_e}(\xi,\eta) = \psi_{c1}(\xi,\eta)\ \mathrm{x(v_0)} + \psi_{c0}(\xi,\eta)\ \mathrm{x(v_1)} + \psi_{c2}(\xi,\eta)\ \mathrm{x(v_2)};$$

O ajustamento das orientações das funções de forma através do mapeamento $\hat{x}^o_e$, reduz-se apenas a modificar o sinal das funções de graus ímpares $grau \geq 3$ como visto em [25], ou seja, multiplica-se por (-1) as funções de graus ímpares.

$$\psi_{ri(n)}(\xi,\eta)\ \circ\ \hat{x}^o_e = \begin{cases} \psi_{ri(n)}(\xi,\eta), & \text{para n = 2,3,... se o(e) = 1} \\ (-1)^n \psi_{ri(n)}(\xi,\eta), & \text{para n = 2,3,... se o(e) = 0.} \end{cases}$$

OBSERVAÇÃO: Para as arestas da malha 3D utiliza-se o mesmo procedimento utilizado em 2D.

3.6.2 Transformação das funções de forma da face

Agora a situação torna-se mais interessante, uma vez que as funções de forma da face triangular varia de acordo com a orientação da face relacionada aos vértices. É importante lembrar que um dos vértices desempenha um papel especial na definição da orientação, tal

orientação começa a partir do vértice de menor índice. Considere a orientação da face $f' < v'_i, v'_j, v'_k >$ (índice(v'_i) < índice(v'_j) < índice(v'_k)) e $f < A,B,C >$, onde $f' \subset \partial e_g$ e $f \subset \partial \hat{e}_g$. Na Figura 3.39, tem-se as orientações da face.

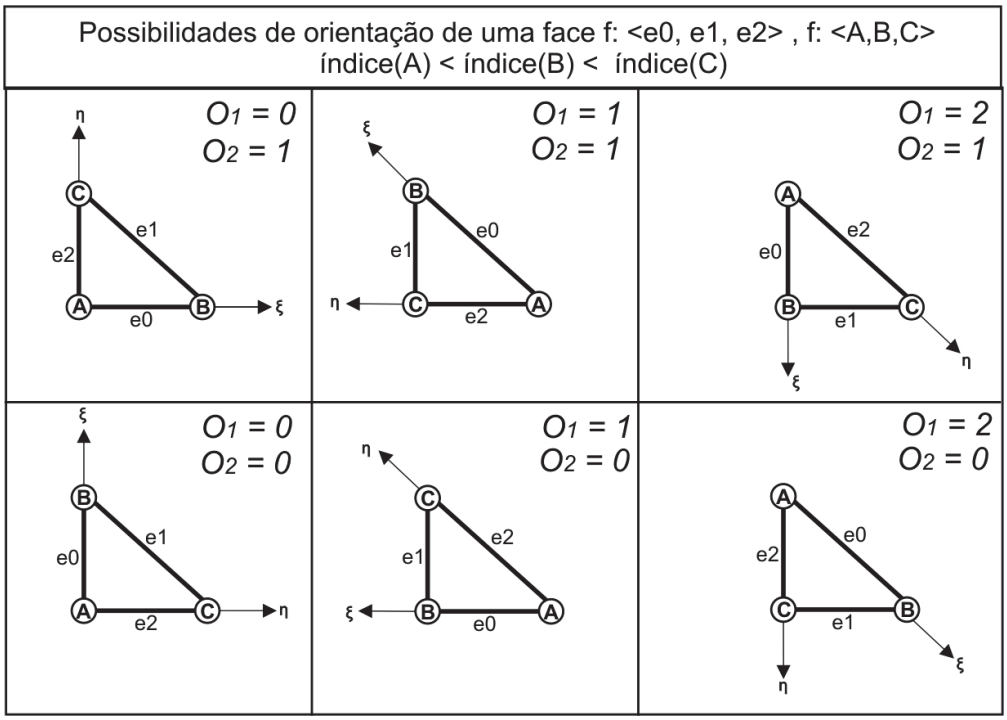


Figura 3.39 Orientações da face

As faces são equipadas com duas orientações *flags* $o_1(f) = \{0,1,2\}$ e $o_2(f) = \{1,0\}$ como visto em [25] indicando a posição dos vértices e compatibilidade das faces f' e f. Se $o_1(f) = 0$ e $o_1(f) = 1$, as funções de forma das faces permanecem inalteradas. Caso contrário, nós temos que transformá-las por meio de um mapeamento $\hat{x}^o_e : \hat{e}_g \rightarrow \hat{e}_g$ visto em [25]. Segue abaixo a expressão referente ao mapeamento idêntico, que é expresso como uma combinação linear das funções de forma das faces com as coordenadas dos vértices.

$$\mathrm{I_t}(\xi,\eta,\zeta) = \sum_{k=0}^{3} \psi_{f_k}(\xi,\eta,\zeta)\ \mathrm{x(v_k)};$$

Se $o_2(f) = 1$ e $o_1(f) = 1$, o mapeamento $\hat{x}^o_e$ rotaciona os vértices A,B e C (ver Figura 3.39) uma única vez no sentido horário (Esta rotação permite uma permutação das funções dos vértices na equação acima). Se $o_2(f) = 1$ e $o_1(f) = 2$, os vértices são rotacionados duas vezes no sentido horário (ver Figura 3.39).

Agora considere $o_2(f) = 0$, se $o_1(f) = 0$, o mapeamento $\hat{x}^o_e$ somente permuta os vértices B e C, se $o_1(f) = 1$ os vértices B e C são permutados e os vértices A,B e C são rotacionados uma vez no sentido horário (ver Figura 3.39). E finalmente se $o_1(f) = 2$ rotaciona uma vez no sentido horário os vértices A,B e C, ficando B,A e C.

Segue abaixo a função de forma da face f:

$$\psi_{f0(n1,n2)}(\xi,\eta,\zeta) = \psi_{cA}\psi_{cB}\psi_{cC} f_{n1}(\psi_{cB} - \psi_{cA} - \psi_{cC} - \psi_{cD}) f_{n2}(\psi_{cC} - \psi_{cA} - \psi_{cB} - \psi_{cD});$$

Por exemplo, Se $o_2(f) = 0$ e $o_1(f) = 2$, a nova função de forma da face f após a compatibilização através do mapeamento $\hat{x}_e^o$ é a seguinte:

$$\psi_{f0(n1,n2)}(\xi,\eta,\zeta) \circ \hat{x}_e^o = \psi_{cB}\psi_{cA}\psi_{cC} f_{n1}(\psi_{cA} - \psi_{cB} - \psi_{cC} - \psi_{cD}) f_{n2}(\psi_{cC} - \psi_{cB} - \psi_{cA} - \psi_{cD});$$

Considere o exemplo (ver Figura 3.40), onde tem-se dois tetraedros T1 e T2, sendo que T1 enxerga a face F, que é comum aos tetraedros como F' e T2 enxerga a face F como F^k. Neste caso a face F^k possui orientação diferente da orientação predefinida, então será necessário um ajuste nas funções de forma da face F^k.

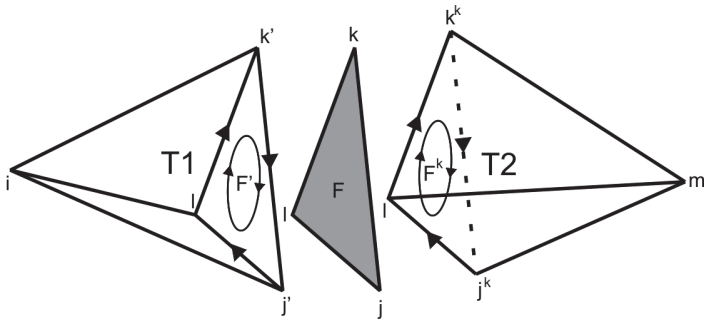


Figura 3.40 Orientação da face do tetraedro

3.7 Geração de malhas

Nesta seção será explicitado apenas aspectos gerenciais relacionados à geração de malhas, visto em [26] e [27]. E serão apresentadas também as relações de malhas com entes geométricos e de malha com malha. Para a geração da malha utiliza-se o método delaunay visto em [28].

Um fenômeno pode estar definido sobre uma dada geometria. Com intuito de definir representações discretas para seus campos vetoriais, malhas geométricas devem ser geradas para este fenômeno. A fim de que processos de cálculo específicos possam ser definidos em qualquer ente geométrico, um dos requisitos deve ser o de que cada entre geométrico deve possuir a sua malha independente dos demais. No entanto entes geométricos devem possuir malhas que coincidam com as malhas dos entes do seu contorno.

3.7.1 Relação de malha x geometria

Na Figura 3.41, tem-se uma geometria de dimensão 2 e suas respectivas malhas 0D, 1D e 2D.

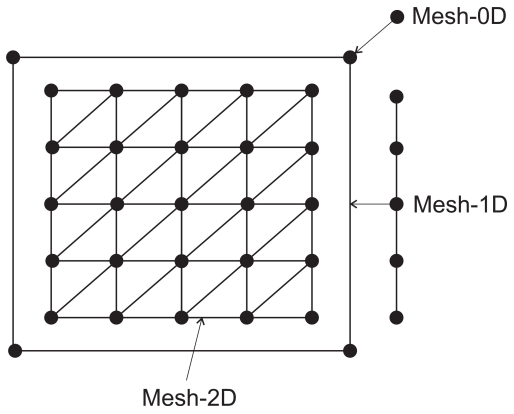


Figura 3.41 Malhas

Cada *GeomEntity* (ver Figura 3.42) se relaciona com uma ou mais malhas (de cada fenômeno ali definido), ou seja, um fenômeno pode possuir mais de uma malha compartilhada ou não.

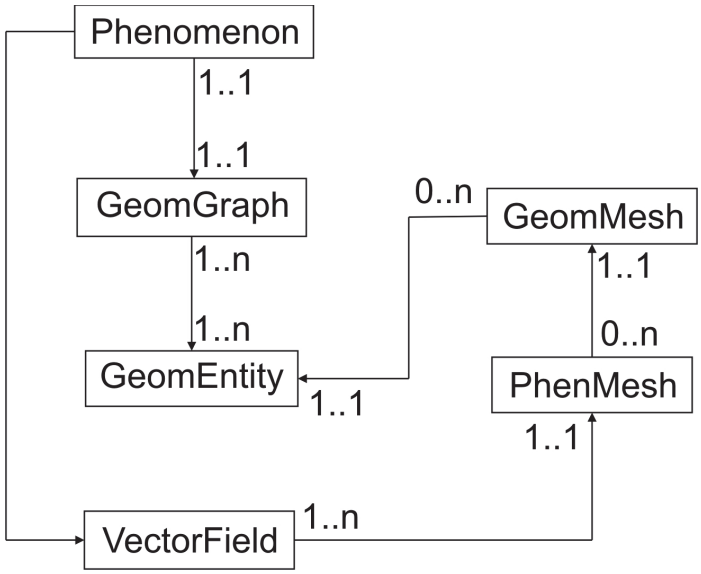


Figura 3.42 Diagrama da malha geométrica e do fenômeno

3.7.2 Relação de malha x malha

Considere uma malha global Mesh-Global (Figura 3.43) e a partir desta malha tem-se as malhas locais Mesh-A e Mesh-B, estas malhas se relacionam entre si através da relação Local-Global. Para estabelecer uma relação Local-Global é realizado um mapeamento da malha a qual os seus entes serão colapsados ou copiados a outra malha. Este mapeamento verifica se o ente já foi inserido ou não, e então renumera-o definindo uma identificação local. A relação Loca-Global é um vetor, onde seu índice é o "id"local e cada elemento deste vetor é o "id"global. A Relação Local-Global relaciona todos os entes, vértices, arestas, faces e volumes.

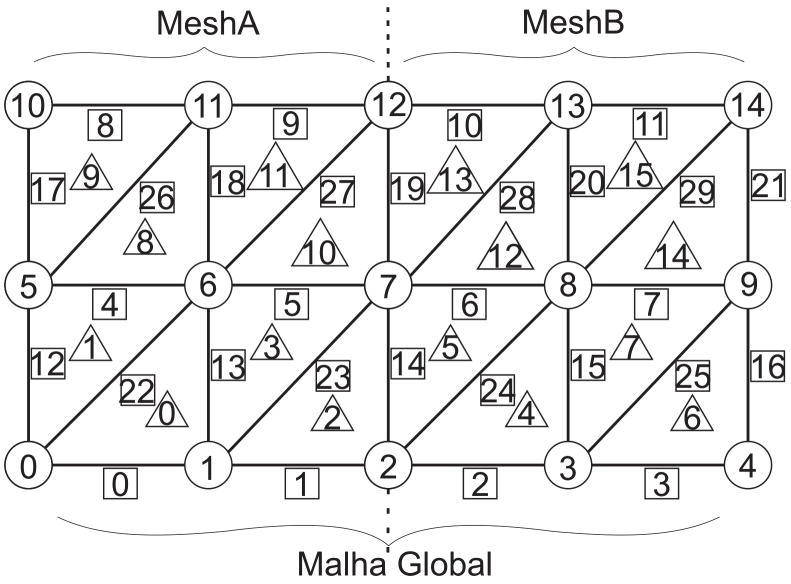


Figura 3.43 Malha global

Na Figura 3.44, tem-se a malha local Mesh-A, e a Figura 3.45 tem-se os vetores que representam as relações Locais-Globais de MeshA referentes aos vértices, arestas e faces.

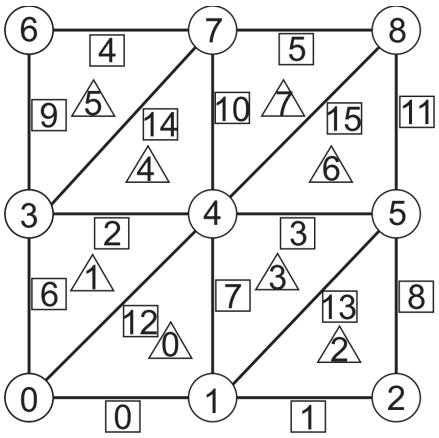


Figura 3.44 Malha Local- MeshA

RLG_V_MeshA = [0, 1, 2, 5, 6, 7, 10, 11, 12]

RLG_A_MeshA = [0, 1, 4, 5, 8, 9, 12, 13, 14, 17, 18, 19, 22, 23, 26, 27]

RLG_F_MeshA = [0, 1, 2, 3, 8, 9, 10, 11]

Figura 3.45 Relação Local-Global para vértices, arestas e faces

Na Figura 3.46, tem-se a malha local Mesh-B, e a Figura 3.47 tem-se os vetores que representam as relações Locais Globais de MeshB referentes aos vértices, arestas e faces.

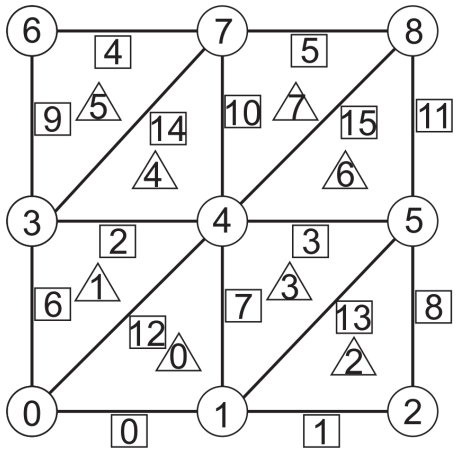


Figura 3.46 Malha Local - MeshB

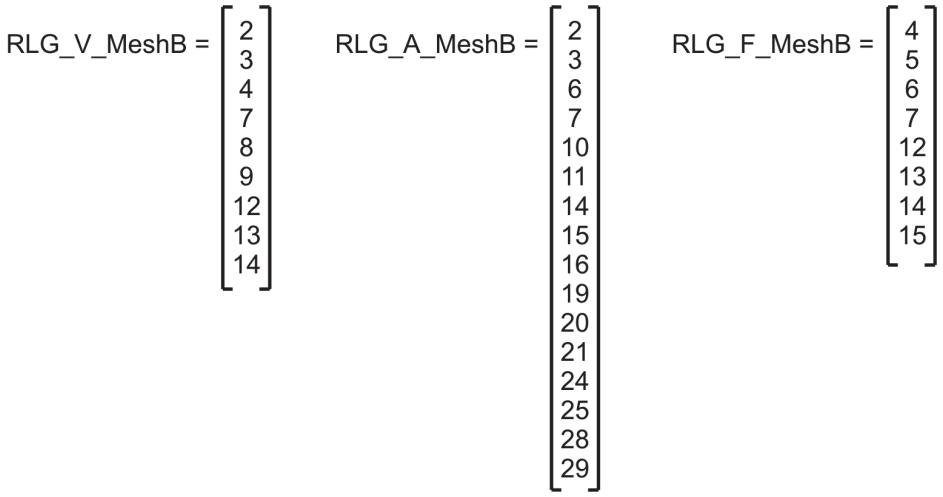


Figura 3.47 Relação Local-Global para vértices, arestas e faces

3.7.3 Algoritmo de geração de malhas

```
##Para cada Phenomenon:
   #Recuperar filhos do root do seu GeomGraph.
   #Instanciar malha a ser gerada.
   #Para cada GeomEntity filho:
        *Verificar se a malha deste GeomEntity já foi produzida
         por outro Phenomenon compartilhante
        *Se já foi gerada:
           **Recuperar e armazenar.
        *Senão:
           **Instanciar a malha de GeomEntity e iniciá-la
           **Recuperar 0D's de GeomEntity
           **Para cada 0D:
             -Se 0D já foi gerado
                  .Recuperar e armazenar
             -Senão:
                  .Gerar a malha
           **Inserir 0D na malha de GeomEntity
           **Recuperar 1D's de GeomEntity
           **Para cada 1D:
            -Se 1D já foi gerado
                  .Recuperar e armazenar
            -Senão:
                  .Gerar a malha
           **Inserir 1D na malha de GeomEntity
           **Recuperar 2D's de GeomEntity
```

```
            **Para cada 2D:
             -Se 2D já foi gerado
                  .Recuperar e armazenar
             -Senão:
                  .Gerar a malha
            **Inserir 2D na malha de GeomEntity
            **Armazenar 3D

##Fim da geração dos GeomEntity
##Unir malhas dos GeomEntity em uma única para
  o Phenomenon
```

3.8 Campos vetoriais do fenômeno

3.8.1 Malha do fenômeno

A malha do fenômeno é definida em relação a malha geométrica para obter a ordem polinomial das arestas, faces e volumes de cada elemento. E conectividades relacionadas aos vértices, arestas, faces e volumes.

A malha do fenômeno está presentada por:

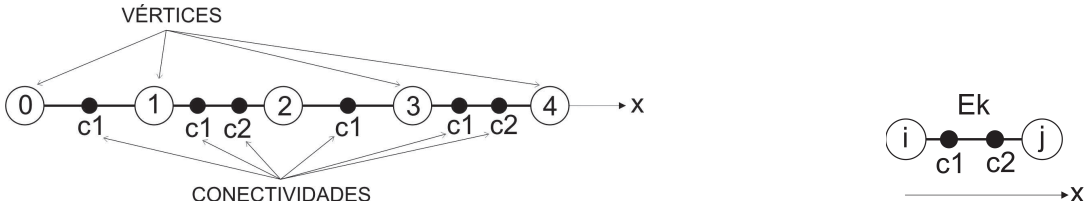
- **Nó:** Abstração que relaciona:
  - **Função de forma**: ψ_i (Ver seção 3.5);
  - **Valores nodais**: $\mathbf{c}_{j_i}$, estão relacionados com vértices, arestas e faces.

Para cada aresta, a malha do fenômeno fornece um número inteiro ≥ 1 que representa o grau polinomial naquela aresta, analogamente para faces e volumes.

Segue abaixo a quantidade de nós para cada ente pertencente a malha do fenômeno:

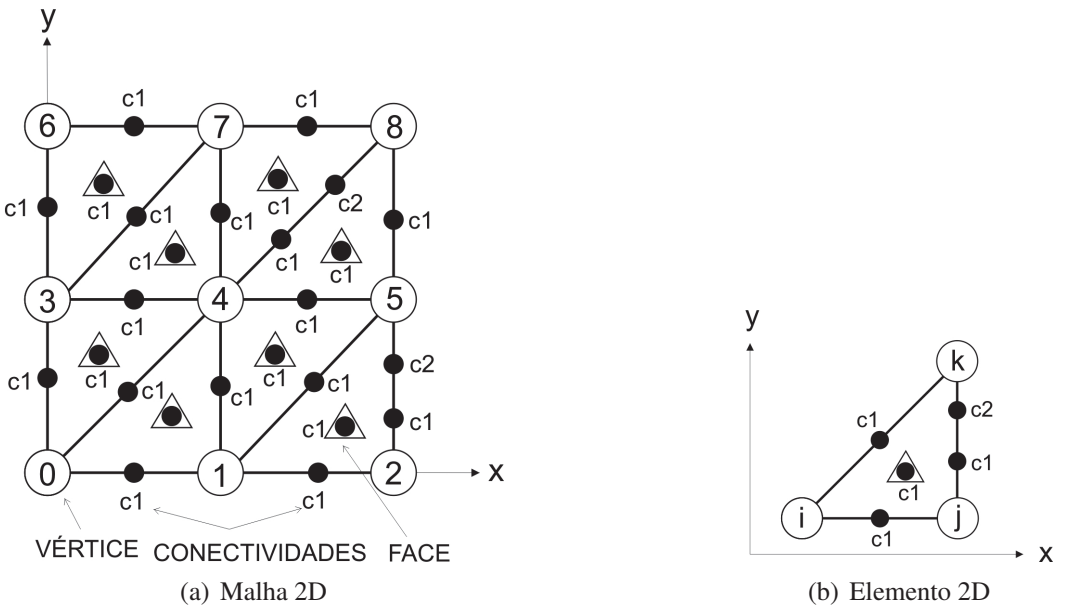
- **Vértices**: Possuem 1 nó / vértice;
- **Arestas**: Possuem $(p_a - 1)$ nós / aresta, onde p_a é a ordem polinomial relacionada à aresta;
- **Faces**: Possuem $\frac{(p_f-2)*(p_f-1)}{2}$ nós / face, onde p_f é a ordem polinomial relacionada à face;
- **Tetraedros**: Possuem $\frac{(p_v-3)*(p_v-2)*(p_v-1)}{6}$ nós / tetraedro, onde p_v é a ordem polinomial relacionada ao volume (tetraedro).

As Figuras (3.48, 3.49 e 3.50) são as malhas do fenômeno para 1D, 2D e 3D e seus respectivos elementos:



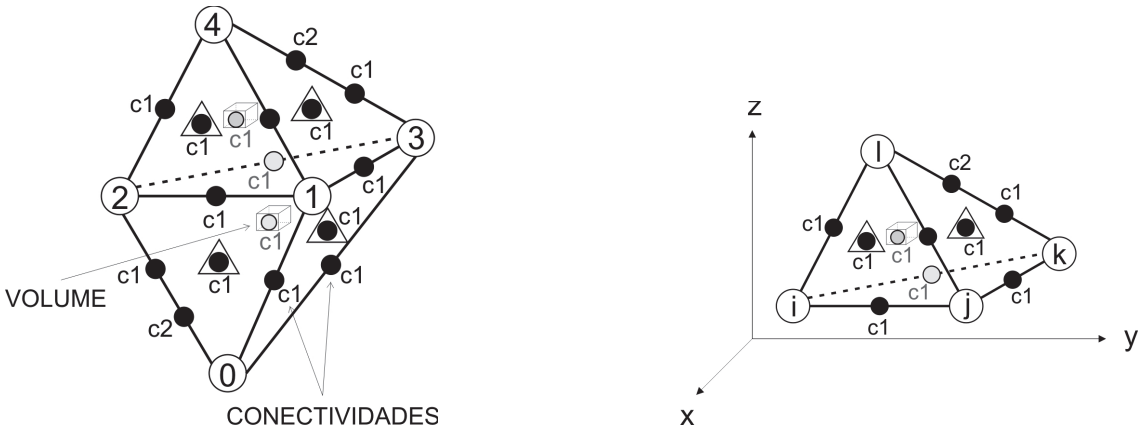
(a) Malha 1D (b) Elemento 1D

Figura 3.48 Malha do fenômeno para 1D



(a) Malha 2D (b) Elemento 2D

Figura 3.49 Malha do fenômeno para 2D



(a) Malha 3D (b) Elemento 3D

Figura 3.50 Malha do fenômeno para 3D

Os nós das aresta são numerados da seguinte forma: já é conhecido a numeração dos vértices (id's) e os nós das aresta são numerados de 0 até n_a, para:

$$n_a = n_v + \sum_{j=0}^{i-1} (P_j - 1) + 1,$$

onde n_a é o número total de nós das arestas, n_v são números de vértices e P_j é o grau polinomial de cada aresta.

Em cada aresta os nós com índices menores referem-se aos polinômios de graus menores, a numeração dos nós aumenta com o aumento do grau do polinômio.

Para uma certa face i, a relação de sua conectividade é $coneci = conec(i-1) + \frac{(P_{fi}-2)*(P_{fi}-1)}{2}$.

A numeração dos nós é importante para os cálculos dos campos vetoriais (ver seção 3.8.2), uma vez que é necessário o acesso dos valores nodais e das funções de forma.

3.8.2 Campos vetoriais

Na malha do fenômeno estão definidos os campos vetoriais (por exemplo, deslocamento, temperatura, etc). Existem três tipos de campos vetoriais, são eles:

- **Escalar**: É aquele em que o tamanho do campo vetorial não depende da malha;
- **Vetorial**: É aquele em que o tamanho do campo vetorial depende da malha e sua estrutura é vetorial;
- **Matricial**: É aquele em que o tamanho do campo vetorial depende da malha e sua estrutura é matricial.

Os campos vetoriais possuem dois parâmetros: o *ContDim* e o *NodalDim*.

Para campos escalares o *ContDim* representa o seu tamanho (como vetor).

Para campos vetoriais *ContDim* significa a dimenção da função contínua $\mathbf{u}(\mathbf{x}) = \sum_{i=0}^{Nff_e-1} \psi_i \, \mathbf{c}_{j_i}$ quando ela é recuperada em função de seus valores discretos ($\mathbf{c}_{j_i}$) e o *NodalDim* é a dimensão nodal de $\mathbf{c}_{j_i}$.

Para campos matriciais o *ContDim* é a quantidades de linhas das submatrizes relacionada a cada nó e o *NodalDim* é a quantidade de colunas das submatrizes, pois as matrizes possuem campos que são indexados em relação aos índices dos nós. No caso de temperatura *ContDim* = 1 e *NodalDim* = 1 e no caso de elasticidade *ContDim* = 2 e *NodalDim* = 2 para 2D.

O campo vetorial é representado por:

$$\mathbf{u}(\mathbf{x}) = \sum_{i=0}^{Nff_e-1} \psi_i \, \mathbf{c}_{j_i};$$

Onde $\mathbf{u}$ é o campo vetorial, $\mathbf{x}$ é a variável de localização no espaço e Nff_e é o número de função de forma do elemento.

O fenômeno pode possuir:

- Um ou mais campos vetoriais;

- Uma ou mais malhas geométricas; e
- Uma ou mais malhas do fenômeno.

OBSERVAÇÃO: Campos vetoriais de fenômenos diferentes em um certo domínio geométrico, poderão possuir a mesma malha geométrica, mas as respectivas malhas podem ser diferentes.

CAPÍTULO 4

Processos de Integração Numérica

A integração numérica é um processo numérico frequentemente utilizado para o cálculo de escalares, vetores e matrizes pelo método do elemento finito.

Neste capítulo serão definidos todos os procedimentos relacionados à integração numérica realizada no nível do Phenomenon. Serão explicitados os dados intrínsecos e extrínsecos de integração e como eles serão encapsulados.

4.1 Procedimentos básicos de integração numérica

Nesta seção serão apresentadas apenas explicações básicas necessárias ao entendimento da integração numérica de uma função, a qual o usuário deseja calcular e que representa uma quantia de seu interesse.

O cálculo de uma quantia definida sobre um determinado elemento e da malha pode ser entendido como o cálculo de uma integral:

$$\mathscr{I}(\nabla\mathbf{u},\mathbf{u},\phi_j) = \int_{e_g} f(\nabla\mathbf{u},\mathbf{u},\mathbf{x})\phi_j \, d\Omega,$$

onde $\mathbf{u}$ é o campo vetorial (visto na seção 3.8), ϕ_j é a função de forma teste vista em [29], $\mathbf{x}$ é a variável de localização no espaço. O campo vetorial $\mathbf{u}$ é dado como:

$$\mathbf{u}(\mathbf{x}) = \sum_{i=0}^{Nff_e-1} \psi_i \, \mathbf{c}_{j_i};$$

onde $\mathscr{N}_e = \{j_i\}_{i=0}^{Nff_e-1}$ é o conjunto de índices de todos os nós relacionados ao elemento do fenômeno e $\{\mathbf{c}_{j_i}\}_{i=0}^{Nff_e-1}$ é o conjunto de valores nodais. $\{\psi_i\}_{i=0}^{Nff_e-1}$ é o conjunto de funções de forma tentativa.

Tanto a malha geométrica, quanto a do fenômeno tem seus respectivos elementos finitos de referência. Nos elementos de referências são armazenados as seguintes informações: tipo do elemento (tetraedro, hexaedro, triângulo, quadrilátero, etc), dimensão da estrutura, método de integração associado ao elemento, ordem máxima de aproximação geométrica do elemento, ordem máxima de integração, dados intrínsecos de integração e funções de forma hierárquicas.

Seja $\hat{\mathbf{x}} : \hat{e}_g \to e_g$, $\mathbf{x} = \hat{\mathbf{x}}(\xi)$, $\xi \in \hat{e}_g$, o mapeamento afim do elemento geométrico de referência $\hat{e}_g$ para o elemento geométrico real e_g. Então:

$$\nabla\mathbf{u}(\mathbf{x}) = \mathbf{B}^{-T} \cdot \nabla_\xi \hat{\mathbf{u}}(\hat{\xi}(\mathbf{x}));$$

$$\nabla \psi_j(\mathbf{x}) = \mathbf{B}^{-T} \cdot \nabla_\xi \hat{\psi}_j(\hat{\xi}(\mathbf{x}));$$
$$\psi_j(\mathbf{x}) = \hat{\psi}_j(\hat{\xi}(\mathbf{x}));$$
$$\hat{\mathbf{u}}(\xi) = \sum_{i=0}^{Nff_e-1} \hat{\psi}_i(\xi)\, \mathbf{c}_{j_i};$$
$$d\Omega = \hat{j}(\xi) d\hat{\Omega},$$

onde $\mathbf{B} = \hat{\mathbf{J}} = \frac{\partial \hat{\mathbf{x}}}{\partial \hat{\xi}}$ é a matriz jacabiana; $\hat{\mathbf{u}}(\hat{\xi}) = \mathbf{u}(\hat{\mathbf{x}}(\xi))$; $\hat{\xi} = \hat{\mathbf{x}}^{-1}$, ∇_ξ é o operador gradiente com sua respectiva variável ξ e $\hat{j}(\xi) = \det(\hat{\mathbf{J}}(\xi))$ é o jacobiano.

A integral acima pode ser utilizada no elemento finito geométrico através do mapeamento $\mathbf{x} = \hat{\mathbf{x}}(\xi)$ como:

$$\hat{\mathscr{I}}(\nabla_\xi \hat{\mathbf{u}}, \hat{\mathbf{u}}, \hat{\phi}_j) = \int_{\hat{e}_g} f(\hat{\mathbf{J}}(\xi)^{-T} \cdot \nabla_\xi \hat{\mathbf{u}}, \hat{\mathbf{u}}, \hat{\mathbf{x}}(\xi)) \hat{\phi}_j \hat{j}(\xi)\, d\hat{\Omega}$$

Esta forma é particularmente atraente por causa da definição de todas as funções de forma agora poder ser feita no elemento finito de referência, independentemente do elemento real geométrico e.

Vamos agora dividir a integração de dados em quatro conjuntos de integração:

4.1.1 Dados intrínsecos de integração relacionados ao elemento geométrico

São os dados que podem ser calculados apenas com o conhecimento do elemento geométrico de referência, são eles:

1. Os pontos de integração
$$I_p = \{\xi_i\}_{i=0}^{N_{ip}-1};$$
e os respectivos pesos
$$I_\omega = \{\omega_i\}_{i=0}^{N_{ip}-1}.$$
Ambos são definidos através de um método de integração.

2. Sempre que a geometria é representada em um espaço gerado por um conjunto de funções de forma geométrica, isto torna necessário calcular as funções de forma no elemento de referência geométrico. Elas são compostas e indexadas da mesma forma como foram feitas para as funções teste e tentativa, e são dadas por:
$$Ge_{sf} = \{\mathbf{PSI}_{ij}\}_{i=0,j=0}^{i=Ngf_e-1,j=N_{ip}-1},$$
onde $\mathbf{PSI}_{ij} = \hat{\psi}_i(\xi_j)$; e seus gradientes:
$$GGe_{sf} = \{\mathbf{GPSI}_{ij}\}_{i=0,j=0}^{i=Ngf_e-1,j=N_{ip}-1},$$
onde $\mathbf{GPSI}_{ij} = \nabla_\xi \hat{\psi}_i(\xi_j)$. Aqui Ngf_e é o número de funções de forma geométrica para o elemento finito e e $\{\psi_i\}_{i=0}^{Ngf_e-1}$ é o conjunto de funções de forma geométrica para o elemento finito e.

4.1.2 Dados intrínsecos de integração relacionados ao fenômeno

Estes dados podem ser calculados apenas com o conhecimento do elemento de referência do fenômeno:

1. Funções de forma tentativa (dada pelo elemento de referência do fenômeno)

$$Tr_{sf} = \{\mathbf{Psi}_{ij}\}_{i=0,j=0}^{i=Nff_e-1,j=N_{ip}-1},$$

onde $\mathbf{Psi}_{ij} = \hat{\psi}_i(\xi_j)$; e seus gradientes:

$$GTr_{sf} = \{\mathbf{GPsi}_{ij}\}_{i=0,j=0}^{i=Nff_e-1,j=N_{ip}-1},$$

onde $\mathbf{GPsi}_{ij} = \nabla_\xi \hat{\psi}_i(\xi_j)$

2. Funções de forma teste (dada pelo elemento de referência do fenômeno)

$$Ts_{sf} = \{\mathbf{Phi}_{ij}\}_{i=0,j=0}^{i=Nff_e-1,j=N_{ip}-1},$$

onde $\mathbf{Phi}_{ij} = \hat{\phi}_i(\xi_j)$; e seus gradientes

$$GTs_{sf} = \{\mathbf{GPhi}_{ij}\}_{i=0,j=0}^{i=Nff_e-1,j=N_{ip}-1},$$

onde $\mathbf{GPhi}_{ij} = \nabla_\xi \hat{\phi}_i(\xi_j)$

4.1.3 Dados extrínsecos de integração relacionados ao elemento geométrico

São os dados de integração que só podem ser calculados com o conhecimento do elemento geométrico real e os dados intrínsecos de integração deste elemento. Não há dependência de campos vetoriais e são calculados pelo elemento geométrico de referência, tendo como entrada o elemento geométrico real e os dados intrínsecos, são eles:

1. Mapeamento do elemento geométrico $\hat{\mathbf{x}}$ nos pontos de integração

$$GeMap = \{\mathbf{gX}_i\}_{i=0}^{N_{ip}-1},$$

onde $\mathbf{gX}_i = \hat{\mathbf{x}}(\xi_i)$.

2. Matriz jacobiana nos pontos de integração.

$$JcMat = \{\mathbf{B}_i\}_{i=0}^{N_{ip}-1},$$

onde $\mathbf{B}_i = \frac{\partial \hat{\mathbf{x}}}{\partial \xi}(\xi_i)$.

3. Matriz jacobiana inversa transposta nos pontos de integração

$$JcMtMT = \{\mathbf{JcMT}_i\}_{i=0}^{N_{ip}-1},$$

onde $\mathbf{JcMT}_i = [\mathbf{B}_i]^{-T}$.

4. Jacobiano nos pontos de integração

$$Jc = \{Jcb_i\}_{i=0}^{N_{ip}-1},$$

onde $Jcb_i = \det(\mathbf{B}_i)$.

4.1.4 Dados extrínsecos relacionados ao fenômeno

São os dados relacionados aos valores dos campos vetoriais e suas derivadas. Para o cálculo desses valores são necessários:

1. Valores do campo vetorial nos pontos de integração:

$$U_{vf} = \{\mathbf{U}_i\}_{i=0}^{N_{ip}-1},$$

e seu gradiente

$$GU_{vf} = \{\mathbf{GU}_i\}_{i=0}^{N_{ip}-1},$$

onde $\mathbf{U}_i = \hat{\mathbf{u}}(\xi_i)$ e $\mathbf{GU}_i = \nabla_\xi \hat{\mathbf{u}}(\xi_i)$

O cálculo dos dados intrínsecos de integração relacionado ao fenômeno e ao elemento geométrico é feito apenas uma vez, mas o cálculo dos dados extrínsecos é realizado para cada elemento finito da malha. Então a integral é aproximada por:

$$\mathscr{I}(\nabla\mathbf{u}, \mathbf{u}, \phi_j) = \int_{e_g} f(\nabla\mathbf{u}, \mathbf{u}, , \mathbf{x})\phi_j \, d\Omega =$$

$$= \hat{\mathscr{I}}(\nabla_\xi\hat{\mathbf{u}}, \hat{\mathbf{u}}, \hat{\phi}_j) = \int_{\hat{e}_g} f(\hat{\mathbf{J}}(\xi)^{-T} \cdot \nabla_\xi\hat{\mathbf{u}}, \hat{\mathbf{u}}, \hat{\mathbf{x}}(\xi))\hat{\phi}_j \hat{j}(\xi) \, d\hat{\Omega} \approx$$

$$\approx \mathscr{I}_{N_{ip}} = \sum_{i=0}^{N_{ip}-1} \omega_i * Jcb_i * f_i * Vol(\hat{e}_g);$$

onde $Vol(\hat{e}_g)$ é o volume do elemento e

$$f_i = f(\mathbf{JcMT}_i * \mathbf{GU}_i, \mathbf{U}_i, \mathrm{Phi}_{ji}, \mathbf{gX}_i).$$

$$f_i = f([\mathbf{B}_i]^{-T} * \nabla_\xi\hat{\mathbf{u}}(\xi_i), \hat{\mathbf{u}}(\xi_i), \hat{\phi}_i(\xi_j), \hat{\mathbf{x}}(\xi_i)).$$

4.2 Encapsular detalhes do baixo nível

A fim de encapsular todos os detalhes sobre os procedimentos do baixo nível e permitir ao usuário se concentrar em dar informações sobre a função f, uma separação de interesses tem que ser feita, então separamos os procedimentos descritos acima em quatro tipos:

1. **Processos relacionados aos dados intrínsecos**: Representam os cálculos dos dados intrínsecos do elemento geométrico e do fenômeno, ou seja, o cálculo de I_p; I_ω; Ge; GGe; Tr_{sf}; GTr_{sf}; Ts_{sf} e GTs_{sf}.

   I_p e I_ω são definidos através de um método de integração.

Como a malha geométrica é do tipo afim, ou seja, as funções de forma geométrica *Ge* são funções de grau polinomial 1, então suas derivadas *GGe* são constantes.

Os dados intrínsecos de integração são armazenados na matriz $\texttt{MAT}_{\texttt{intr}}$, a qual é implementada em linguagem C++ [30]. As linhas desta matriz representam os pontos de integração. Nas primeiras k colunas encontram-se a organização das funções de forma (ordem) nos pontos de integração, as colunas seguintes referem-se as componentes do gradientes de cada uma das funções de forma nos pontos de integração.

Suponha que o usuário esteja calculando uma determinada quantia de um elemento e e necessite da função de forma Tr_{sf} e sua derivada GTr_{sf} de um certo elemento, então o usuário recupera a malha do fenômeno através do campo vetorial deste fenômeno. A malha do fenômeno fornece as ordens polinomiais para arestas, faces e volume do elemento e as conectividades de cada nó. Com isto, definem-se as funções de forma do elemento que podem ser recuperadas da matriz $\texttt{MAT}_{\texttt{intr}}$.

2. **Processos relacionados às funções de forma teste e tentativa**: Estes processos irão calcular todos os dados referentes às funções de forma teste e tentativa em relação ao elemento finito real juntamente com a matriz jacobiana inversa transposta e o jacobiano. Isso realmente significa que o cálculo dos dados extrínsecos, não depende de informações do campo vetorial, desde que:

$$\psi_j(\mathbf{x}) = \hat{\psi}_j(\hat{\xi}(\mathbf{x}))$$

e desde que o ponto de integração no elemento finito real seja

$$\mathbf{x}_i = \hat{\mathbf{x}}(\xi_i),$$

então

$$\mathrm{psi}_{ij} = \psi_j(\mathbf{x}_i) = \hat{\psi}_j(\xi_i),$$

para $i = 0,\ldots,N_{ip}-1$ e $j = 0,\ldots,Nff_e-1$. O mesmo pode ser feito para as funções de forma teste, obtendo:

$$TrlSf = \{\mathrm{psi}_{ij}\}_{i=0,j=0}^{i=Nff_e-1,j=N_{ip}-1} = Tr_{sf}$$

e

$$TstSf = \{\mathrm{phi}_{ij}\}_{i=0,j=0}^{i=Nff_e-1,j=N_{ip}-1} = Ts_{sf}$$

A fim de calcular os gradientes das funções teste e tentativa em cada ponto de integração, nós consideramos a seguinte relação:

$$\frac{\partial(.)}{\partial \mathbf{x}}(\mathbf{x}) = \frac{\partial\hat{(.)}}{\partial \xi}(\hat{\xi}(\mathbf{x}))\cdot\mathbf{B}^{-1}(\xi)$$

e portanto

$$\frac{\partial(.)}{\partial \mathbf{x}}(\mathbf{x}_i) = \frac{\partial\hat{(.)}}{\partial \xi}(\xi_i)\cdot\mathbf{B}^{-1}(\xi_i)$$

$i = 1, \ldots, N_{ip} - 1$. Isso permite a seguinte definição

$$GTstSf = \{\mathbf{Gphi}_{ij}\}_{i=0,j=0}^{i=Nffe-1,j=N_{ip}-1} = \{\mathbf{JcMT}_i * \mathbf{GPhi}_{ij}\}_{i=0,j=0}^{i=Nffe-1,j=N_{ip}-1};$$

para o gradiente das funções teste e

$$GTrlSf = \{\mathbf{Gpsi}_{ij}\}_{i=0,j=0}^{i=Nffe-1,j=N_{ip}-1} = \{\mathbf{JcMT}_i * \mathbf{GPsi}_{ij}\}_{i=0,j=0}^{i=Nffe-1,j=N_{ip}-1},$$

para o gradiente das funções tentativas. Neste procedimento está incluido o peso de integração modificado

$$mI_\omega = \{\Omega_i\}_{i=0}^{N_{ip}-1} = \{\omega_i * Jcb_i\}_{i=0}^{N_{ip}-1} * Vol(\hat{e}).$$

Também, os pontos de integração relacionados ao elemento real serão

$$IpX = \{\mathbf{X}_i\}_{i=0}^{N_{ip}-1} = \{\mathbf{x}_i\}_{i=0}^{N_{ip}-1} = GeMap;$$

É importante mencionar que o cálculo do $\mathbf{X}_i$, $i = 0, \ldots, N_{ip} - 1$ pode ser muito complexo, dependendo se a malha geométrica define uma malha paramétrica ou não. O ponto é que a malha paramétrica está localizada no espaço paramétrico, e não no espaço real. A fim de obter a malha real para esse caso, o objeto *GeomEntity* tem que mapear os elementos finitos do espaço paramétrico para o espaço geométrico real. Se for esse o caso, dois mapeamentos são necessários, isto é, $\hat{\mathbf{x}}_p : \hat{\Omega}_e \to \Omega_p$ e $\hat{\mathbf{x}}_g : \Omega_p \to \Omega_e$, onde $\hat{\mathbf{x}}_p$ mapeia o elemento finito de referência $\hat{\Omega}_e$ para o elemento finito paramétrico Ω_p e $\hat{\mathbf{x}}_g$ mapeia o elemento finito paramétrico para o elemento finito real Ω_e. Neste caso $\hat{\mathbf{x}}(\xi) = \hat{\mathbf{x}}_g(\hat{\mathbf{x}}_p(\xi))$ para todo $\xi \in \hat{\Omega}_e$. O elemento finito real é responsável pela construção de $\hat{\mathbf{x}}_p$ e seu gradiente em um determinado conjunto de pontos em $\hat{\Omega}_e$ com a ajuda dos dados geométricos intrínsecos (*Ge*; *GGe*) e o objeto *GeomEntity* é responsável pela construção de $\hat{\mathbf{x}}_g$ e seu gradiente em um determinado conjunto de pontos em Ω_p. A matriz jacobiana é construida da seguinte forma:

$$\mathbf{B}_i = [\frac{\partial \hat{\mathbf{x}}_g}{\partial \mathbf{x}_p}(\hat{\mathbf{x}}_p(\xi_i))] \cdot [\frac{\partial \hat{\mathbf{x}}_p}{\partial \xi}(\xi_i)]$$

A matriz Jacobiana inversa transposta é

$$JcMtMT = \{\mathbf{JcMT}_i\}_{i=0}^{N_{ip}-1},$$

onde $\mathbf{JcMT}_i = [\mathbf{B}_i]^{-T}$.

e o Jacobiano nos ponto de integração

$$Jc = \{Jcb_i\}_{i=0}^{N_{ip}-1},$$

onde $Jcb_i = \det(\mathbf{B}_i)$.

Portanto, definimos os seguintes conjuntos de dados: IpX; $TrlSf$; $TstSf$; $GTrlSf$; $GTstSf$; mI_ω; $JcMtMT$ e Jc.

3. **Processos relacionados ao campo vetorial**: Este conjunto de procedimentos irá calcular os dados extrínsecos de integração, que estão relacionados com os campos vetoriais e suas derivadas em relação às variáveis do elemento finito real. Eles são os campos vetoriais **u** nos pontos de integração,

$$u_{vf} = \{\mathbf{u}_i\}_{i=0}^{N_{ip}-1} = U_{vf} = \{\mathbf{U}_i\}_{i=0}^{N_{ip}-1},$$

e seu gradiente

$$Gu_{vf} = \{\mathbf{Gu}_i\}_{i=0}^{N_{ip}-1}.$$

Eles são calculados da seguinte forma:

$$\hat{\mathbf{u}}(\xi) = \sum_{j=0}^{Nff_e-1} \mathbf{c}_{n_j} \hat{\psi}_j(\xi)$$

e, portanto,

$$\mathbf{u}(\mathbf{x}_i) = \hat{\mathbf{u}}(\xi_i)$$

ou

$$\mathbf{u}_i = \hat{\mathbf{u}}(\xi_i) = \sum_{j=0}^{Nff_e-1} \mathbf{c}_{n_j} \mathrm{psi}_{ji}$$

e

$$\nabla_x \mathbf{u}(\mathbf{x}_i) = \mathbf{B}^{-T}(\xi_i) \cdot \nabla_\xi \hat{\mathbf{u}}(\xi_i)$$

ou

$$\nabla_x \mathbf{u}(\mathbf{x}_i) = \sum_{j=0}^{Nff_e-1} \mathbf{B}^{-T}(\xi_i) \cdot \nabla_\xi \hat{\psi}_j(\xi_i) \cdot \mathbf{c}_{n_j}^T$$

ou, finalmente,

$$\mathbf{Gu}_i = \nabla_x \mathbf{u}(\mathbf{x}_i) = \sum_{j \in \mathscr{N}_e} \mathbf{Gpsi}_{ji} \cdot \mathbf{c}_j^T$$

para todo $i = 1, \ldots, N_{ip} - 1$.

Note que o conjunto de valores nodais $\{\mathbf{c}_j\}_{j \in \mathscr{N}_e}$ devem ser recuperados para cada elemento finito e na malha do fenômeno relacionada com o campo vetorial **u**, a fim de obter um vetor de valores nodais. $\mathscr{N}_e$ contém Nff_e, que é o número de funções de forma tentativa definida no elemento e e necessário para calcular $\mathbf{u}_e$. O conjunto (i) é usado para numerar e gerar as conectividades e recuperar as funções de forma e suas derivadas a partir dos dados intrínsecos de integração relacionados ao fenômeno (levando em conta Tr_{sf}, GTr_{sf}, Ts_{sf} e GTr_{sf}), enquanto o outro é usado para recuperar o vetor de valores nodais para o elemento e. Os valores dos conjuntos $\{\mathbf{c}_j\}_{j \in \mathscr{N}_e}$ coincide com aqueles do vetor de valores nodais.

4. **Processos relativos à integração**: Este conjunto irá calcular a integral de uma dada função f. Isto é composto da função f em cada ponto de integração com o respectivo elemento finito real

$$F = \{f_i\}_{i=0}^{N_{ip}-1},$$

onde

$$f_i = f(\mathbf{Gu}_i, \mathbf{u}_i, \mathrm{phi}_{ji}, \mathbf{X}_i)$$

e o processo de integração em si

$$\mathscr{I}_{N_{ip}} = \sum_{i=0}^{N_{ip}-1} \Omega_i * f_i.$$

4.3 Exemplo de transferência de calor

Considere o problema de transferência de calor em uma chapa de aço levando em conta o processo de condução e convecção, onde $K(u)$ é a matriz de condutividade:

$$-\nabla(K(u)\cdot\nabla u) + Cu = f$$

Observe que a condutividade térmica do material depende da temperatura a que ele está submetido, por isso ela é não-linear.

Então, pelo método do elemento finito vamos obter a seguinte forma fraca:

$$B(u,v) = \int_\Omega \left[\nabla v^T \cdot K(u) + Cvu\right]\, d\Omega + \int_{\Gamma_M} huv\, d\Gamma =$$

$$\int_\Omega fv\, d\Omega + \int_{\Gamma_N} Qv\, d\Gamma + \int_{\Gamma_M} hu_\infty v\, d\Gamma = L(v)$$

Na expressão acima, considera-se a condição de contorno de Neumann, ou seja, os valores das derivadas são especificados no contorno do domínio Γ_N. Há também a condição de contorno de Cauchy ou condição de contorno Mista (no domínio Γ_M).

$$B(u_h, v_h) = L(v_h),\ \ \forall\, u_h \in S_h \quad e \quad v_h \in S_{oh};\quad u_h = \sum_{i=0}^{Nff-1} u_i \psi_i$$

Isto é um exemplo, e para o método de Newton Raphson necessita-se do cálculo da matriz Jacobiana e do Resíduo. Será ilustrado o cálculo da matriz jacobiana.

Como v_h é linear e apenas u_h está preso a não-linearidade, então podemos considerar o Resíduo da seguinte forma:

$$R_i(u_h) = B(u_h, \phi_i) - L(\phi_i),\ \ \forall\, \phi_i \in S_{oh}$$

Segue abaixo a definição da Matriz Jacobiana J:

$$J_{ij} = \frac{\partial R_i}{\partial u_j},\ \ i,j = 0,...,n-1.$$

Onde u_k é o campo vetorial de temperatura e $u_k = \sum_{i=0}^{Nff-1} u_i \psi_i$

$$J_{ij} = \frac{\partial B(u_k,\phi_i)}{\partial u_j} = \frac{\partial}{\partial u_j}\left[\int_\Omega \left[\nabla \phi_i^T K(u_k)\cdot\nabla u_k + Cu_k\phi_i\right]\, d\Omega + \int_{\Gamma_M} hu_k\phi_i\, d\Gamma\right]$$

$$= \int_\Omega \left[\nabla \phi_i^T \cdot \left(\frac{\partial K(u_k)}{\partial u} \cdot \nabla u_k + K(u_k) \nabla(\cdot) \right) + C\phi_i \right] \cdot \underbrace{\frac{\partial u_h}{\partial u_j}}_{\psi_j} \, d\Omega + \int_{\Gamma_M} h\phi_i \underbrace{\frac{\partial u_h}{\partial u_i}}_{\psi_j} \, d\Gamma$$

$$J_{ij} = \underbrace{\int_e \nabla \phi_i^T \cdot \frac{\partial K(u_k)}{\partial u} \cdot \nabla u_k \psi_j \, d\Omega}_{q_1} + \underbrace{\int_e \nabla \phi_i^T \cdot K(u_k) \cdot \nabla \psi_j \, d\Omega}_{q_2} + \underbrace{\int_e C\phi_i \psi_j \, d\Omega}_{q_3} + \underbrace{\int_{\Gamma_M \cap \partial e} h\phi_i \psi_j \, d\Gamma}_{q_4}$$

Observe que $q3$ e $q4$ não dependem de campos vetoriais, seguem abaixo as resoluções das quantias $q1$ e $q2$:

Considere o gradiente da função teste em cada ponto de integração, onde np é o número de pontos de integração:

$$\left\{ \nabla_x \phi^{(r)} \right\}_{r=0}^{np-1} ; \ \left\{ \nabla_x \psi^{(r)} \right\}_{r=0}^{np-1}$$

O usuário necessita de:

- O campo vetorial; e
- A malha do fenômeno.

Tendo isto o sistema deve calcular para o determinado elemento finito:

$$\left\{ U_k^{(r)} \right\}_{r=0}^{np-1} \longrightarrow u_k(x_r) = \sum_{s=0}^{n_e-1} u_s(k) \psi_s(x_r)$$

$$\left\{ DU_k^{(r)} \right\}_{r=0}^{np-1} \longrightarrow \nabla u_k(x_r) = \sum_{s=0}^{n_e-1} u_s(k) \nabla_x \psi_s(x_r)$$

Então:

$$q_1 = \sum_{r=0}^{np-1} \left[\nabla_x \phi_i^T(x_r) \cdot \frac{\partial K(u_k(x_r))}{\partial u} \cdot \nabla_x u_k(x_r) \psi_j(x_r) \right] \overline{w}_r$$

$$q_2 = \sum_{r=0}^{np-1} \left[\nabla_x \phi_i^T(x_r) \cdot K(u_k(x_r)) \cdot \nabla_x \psi_j(x_r) \right] \overline{w}_r$$

$$\overline{w}_r = w * Jcb * Vol(\hat{e})$$

CAPÍTULO 5

Campos Vetoriais Discretos, Operações de Álgebra Linear e Resolvedores

Neste Capítulo serão definidos os campos vetoriais discretos, as operações de álgebra linear e os resolvedores lineares. Além das operações que serão apresentadas, outras operações poderão ser definidas pelo usuário.

5.1 Campos vetoriais discretos

Campos vetoriais discretos são aqueles que podem estar incluidos em sistemas acoplados. Cada campo vetorial que depender de uma malha, o sistema tem que dizer qual a malha ele está vinculado. Se em uma determinada malha é conhecido o tamanho do campo vetorial associado a ela, então o sistema pode alocar memória para o mesmo.

Já no caso de matrizes têm-se os fenômenos (Phen) que são os donos e os fenômenos que são acoplados podendo serem os mesmos ou diferentes. O tamanho a ser alocado para a matriz é *NodalDim x Quantidade de nós* daquele campo vetorial do fenômeno.

Considere os campos vetoriais pertencente a um determinado fenômeno:

$$\mathbf{u}(\mathbf{x}) = \sum_{i=0}^{m-1} \psi_i \, \mathbf{c}_{j_i} \quad e \quad \mathbf{v}(\mathbf{x}) = \sum_{i=0}^{s-1} \phi_i \, \mathbf{r}_{j_i},$$

onde $\{j_i\}_{i=0}^{m-1}$ é o conjunto de índices de todos os nós relacionados ao elemento do fenômeno e $\{\mathbf{c}_{j_i}\}_{i=0}^{m-1}$ e $\{\mathbf{r}_{j_i}\}_{i=0}^{s-1}$ são os conjuntos de valores nodais. ψ_i e ϕ_i são funções de forma, onde m e s são os números de funções de forma,

então, o vetor $\mathbf{V}$ que representa o campo vetorial discreto é:

$$\mathbf{V} = \begin{bmatrix} \mathbf{c_0} \\ \mathbf{c_1} \\ . \\ . \\ . \\ \mathbf{c_{m-1}} \\ \mathbf{r_0} \\ \mathbf{r_1} \\ . \\ . \\ . \\ \mathbf{r_{s-1}} \end{bmatrix}$$

Na Figura 5.1 tem-se os diagramas das definições específica (campos vetoriais dos fenômenos) e genérica (campos vetoriais discretos). Na definição específica encontram-se os seguintes componentes:

- **Phen**: é o fenômeno;
- **GeomGraph**: é o grafo da geometria;
- **GeomEntity**: são os entes geométricos;
- **Phenomenon VectorField**: são os campos vetoriais que estão presentes no fenômeno, por exemplo, temperatura, deslocamento, etc;
- **PhenMesh**: é a malha do fenômeno, que é relacionada a uma malha geométrica;
- **RefElemGeom e RefElemPhen**: são os elementos de referências geométrico e do fenômeno; e
- **WeakForms**: são ferramentas para cálculos de quantias.

A definição genérica possui os seguintes componentes:

- **gPhen**: são os fenômenos, que representam o Group no nível do Phenomenon;
- **VectorField discrete**: são os campos vetoriais discretos. Estes campos podem ser escalares, vetoriais e matriciais; e
- **WeakForms**.

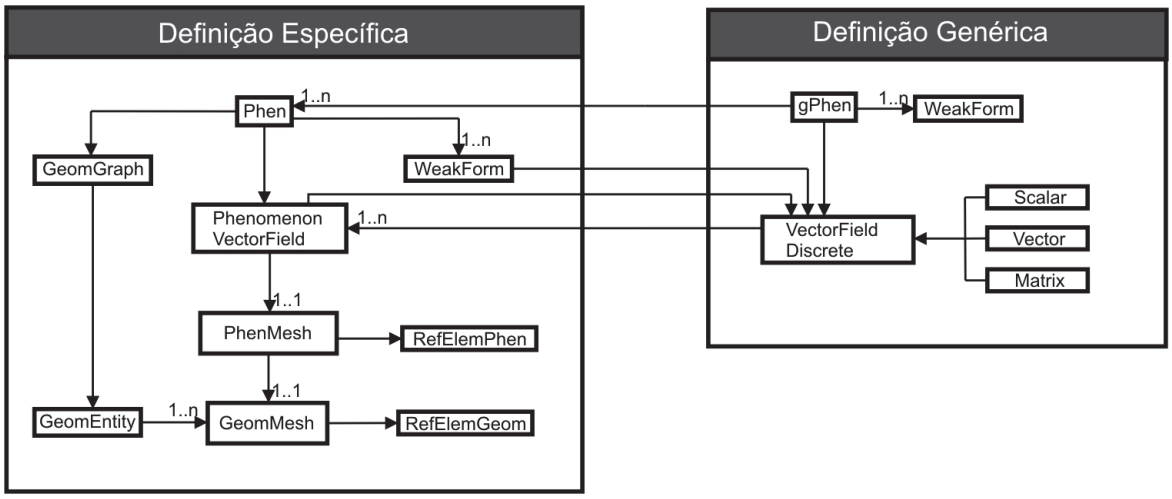


Figura 5.1 Diagrama das definições dos campos vetoriais discretos.

5.2 Operações tipo BLAS

As operações tipo BLAS vistas em [31] (Basic Linear Algebra Subprograms) são realizadas por objetos `gPhen` da classe `gPhenomenon`, que são os representantes do **Group** no nível do **Phenomenon**. Para descrever as operações será utilizada a seguinte notação: r e α são escalares; **x** e **y** são vetores; **A** é uma matriz esparsa; **T** é uma matriz triangular; e **B** e **C** são matrizes densas. Seguem abaixo estas operações:

1. **Operações BLAS I:** são operações envolvendo vetores e/ou escalares. Seguem abaixo as descrições de algumas dessas operações:

Tabela 5.1 Operações BLAS I

sparse dot product	$r \leftarrow x^T y$ $r \leftarrow x^H y$
sparse vector update	$y \leftarrow \alpha x + y$
sparse gather	$x \leftarrow y\|_x$
sparse gather and zero	$x \leftarrow y\|_x;\ y\|_x \leftarrow 0$
sparse scatter	$y\|_x \leftarrow x$

Onde $\mathbf{y}|_\mathbf{x}$ refere-se a entrada de **y** que tem índices comuns com o vetor **x**.

2. **Operações BLAS II**: são operações envolvendo escalares e/ou vetores e/ou matrizes. Seguem abaixo as descrições de algumas dessas operações:

Tabela 5.2 Operações BLAS II

sparse matrix / vector multiply	$y \leftarrow \alpha Ax + y$ $y \leftarrow \alpha A^T x + y$ $y \leftarrow \alpha A^H x + y$
sparse triangular solve	$x \leftarrow \alpha T^{-1} x$ $x \leftarrow \alpha T^{-T} x$ $x \leftarrow \alpha T^{-H} x$

3. **Operações BLAS III**: são as operações envolvendo escalares e/ou matrizes. Seguem abaixo as descrições de algumas dessas operações:

Tabela 5.3 Operações BLAS III

sparse matrix / vector multiply	$C \leftarrow \alpha AB + C$ $C \leftarrow \alpha A^T B + C$ $C \leftarrow \alpha A^H B + C$
sparse triangular solve	$B \leftarrow \alpha T^{-1} B$ $B \leftarrow \alpha T^{-T} B$ $B \leftarrow \alpha T^{-H} B$

5.3 Blas Operator

Nesta seção será explicitado o operador que realiza operações de álgebra linear. Para entendimento, serão mostrados os diagramas que contém a definição da tarefa, a execução da tarefa BLAS e dados e execução da tarefa solver.

Na definição da tarefa encontram-se os seguintes componentes:

- **Phenomenon**: representa o fenômeno;
- **gPhen**: são os fenômenos, que representam o Group no nível do Phenomenon;
- **Campos vetoriais abstratos**; e
- **WeakForms**: são ferramentas para cálculo de quantias.

A execução da tarefa BLAS e dados possui os seguintes componentes:

- **BlasOperator <Par>**: é um operador que realiza as operações de álgebra linear para o `gPhen`, com isso há uma separação de interesses entre a parte controladora que fica com o `gPhen` e um operador que realiza as operações permitindo que haja uso de diferentes bibliotecas para o cálculo envolvendo escalares, vetores e matrizes. O parâmetro "**Par**"é utilizado para a definição da biblioteca utilizada no cálculo acima referido, podendo ser: GMM++, Seldon, SparseLib, IML++, entre outras.
- **Operações abstratas**: são as operações que envolvem escalares, vetores e matrizes. Estas operações utilizam os tipos abstratos como dados de entrada e são independentes da biblioteca. Elas definem, portanto, uma interface que permite a parte controladora (**gPhen**) definir abstratamente todas as operações que deseja realizar com a parte executante que é o **BlasOperator <Par>**.
- **Tipos abstratos**: são os tipos de vetores e matrizes. Os mesmos são independentes do tipo da biblioteca que está sendo utilizada, pois as operações abstratas utilizam estes tipos para cálculos que envolvem escalares, vetores e matrizes que são especializados pelo **BlasOperator <Par>**.

Na execução da tarefa solver contém os seguintes componentes:

- **Solver**: é um objeto utilizado pelas WeakForms para resolver sistemas algébricos lineares.
- **SolverBlasInterFace<Par, solv Type>**: é uma interface que tem como dados de entrada a biblioteca e o tipo do **solver**. O **SolverBlasInterFace** realiza a seguinte operação $Solver(\mathbf{A}, \mathbf{x}, \mathbf{b}, \mathbf{ParVec})$, onde **A** é uma matriz, **x** é o vetor solução, **b** é o lado direito e **ParVec** são parâmetros dependendo do tipo do **solver**. Outros dados como definição do precondicionador podem ser inseridos através destes parâmetros.

Na Figura 5.2 tem-se os diagramas referente à definição da tarefa, execução da tarefa BLAS e dados, e execução da tarefa solver.

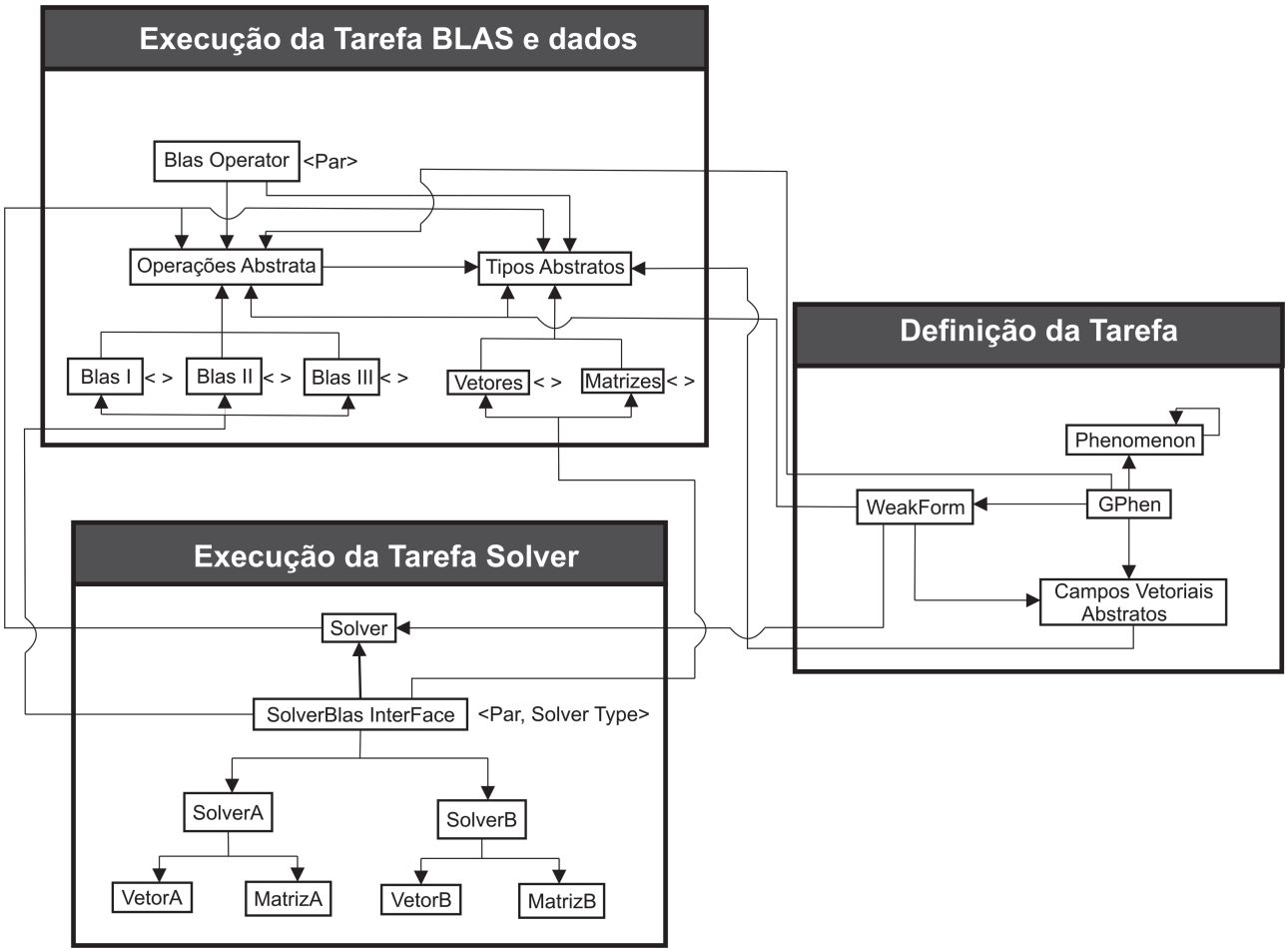


Figura 5.2 Diagramas de definição e execução da tarefa.

5.4 Solução de Sistemas Algébricos Lineares

O sistema de equações lineares pode ser descrito como demonstrado abaixo, onde **A** é a matriz, **x** é um vetor solução e **b** é o lado direito, sendo que **x** e **b** podem também ser matrizes.

$$\mathbf{Ax} = \mathbf{b} \tag{5.1}$$

Um **solver** pode depender de vários parâmetros, sendo que a quantidade de parâmetros depende do tipo do solver. Os vetores **x** e **b** podem ser substituidos por matrizes, este caso significa que várias soluções serão obtidas. Com os estados globais e os parâmetros estabelecidos, tem-se então a definição do objeto **solver**. O **solver** é um componente que herda a classe **solver**. Esta classe encapsula a execução do **solver** e é capaz de receber os parâmetros e

utilizá-los adequadamente além de armazená-los. Seguem exemplos de Solvers: GMRES, CG (Conjugate Gradient), LeastSquare, BICGSTAB, Quasi minimal residual, entre outros.

5.5 WeakForm

Os objetos **WeakForm** são ferramentas para cálculo de quantias e representam parte da lei de comportamento discreto, cálculo de vetores e matrizes, condições de contorno e outros procedimentos. Uma **WeakForm** pode armazenar parâmetros, os quais estão relacionados com os dados específicos da simulação (por exemplo: funções para definição de contorno, parâmetros para cálculo de uma quantia, etc).

Todas as operações citadas acima são realizadas por objetos **WeakForm** localizados no **gPhen**. E elas são características do simulador e não dependem dos dados da simulação para sua realização.

Os objetos **WeakForm** quando solicitados para serem executados, os mesmos não dependem da geometria, malha ou de qualquer outro dado de uma simulação. Todos estão prontos para serem executados logo após a instanciação do simulador.

CAPÍTULO 6

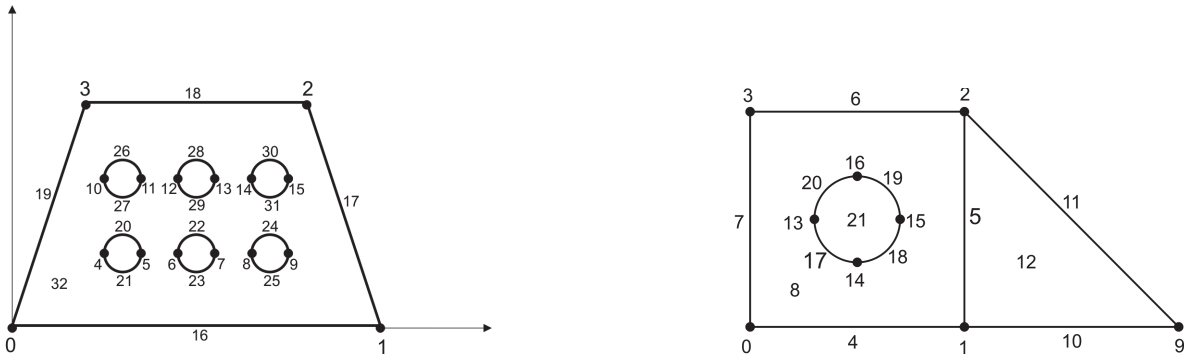
Resultados e Análises

Neste capítulo serão apresentados os resultados relacionados à geração da malha geométrica considerando estrutura de dimensão 2 imersa no espaço de dimensão 2. E será realizado um estudo de caso das facilidades e dificuldades nas implementações referente à construção do simulador e da simulação. E também serão apresentadas as contribuições para trabalhos futuros.

6.1 Malha geométrica

Para a geração das malhas foram definidas as seguintes geometrias:

- **Geometria 1** (ver Figura 6.1(a)): Esta geometria possui uma superfície (id 32), dezesseis curvas (id's 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30 e 31) e oito pontos (id's 0, 1, 2, 3, 9, 13, 14, 15). A geometria 1 representa uma placa de aço contendo seis furos, ou seja, não possuem materiais em seu interior (vazio). A descrição do arquivo que carrega esta geometria para a simulação está na seção B.2 (do apêndice B).
- **Geometria 2** (ver Figura 6.1(b)): Esta geometria possui três superfícies (id´s 8, 12 e 21), dez curvas (id's 4, 5, 6, 7, 10, 11, 17, 18, 19 e 20) e nove pontos (id's 0, 1, 2, 3, 9, 13, 14, 15 e 16). A superfície cujo contorno externo é circular, possui material em seu interior. A descrição do arquivo que carrega esta geometria para a simulação está na seção B.2 (do apêndice B).



(a) Geometria 1

(b) Geometria 2

Figura 6.1 Geometrias

Na figura 6.2 tem-se a malha geométrica da geometria 1. Esta malha possui 2303 vértices, 6668 arestas e 4360 faces:

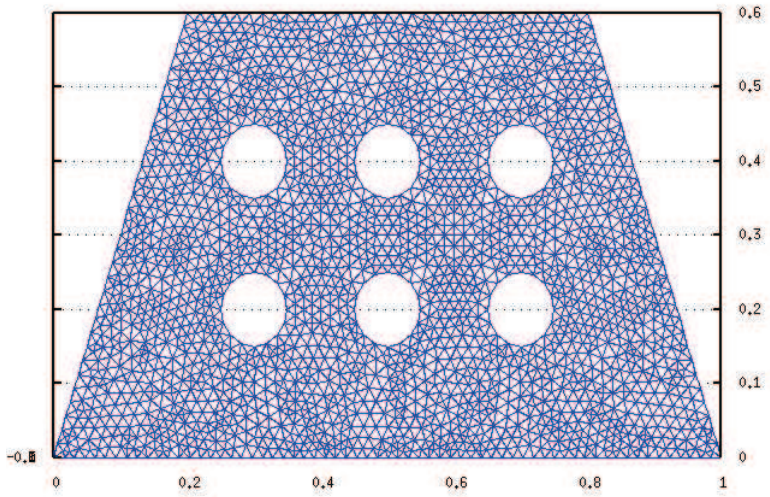


Figura 6.2 Malha geométrica da geometria 1

A partir da geometria 2 e através do gerador de malha, gerou-se a malha global (ver Figura 6.3), que é a malha do fenômeno A. Este fenômeno está atuando nas superfícies (id's 8 e 12) (ver Figura 6.1(b)):

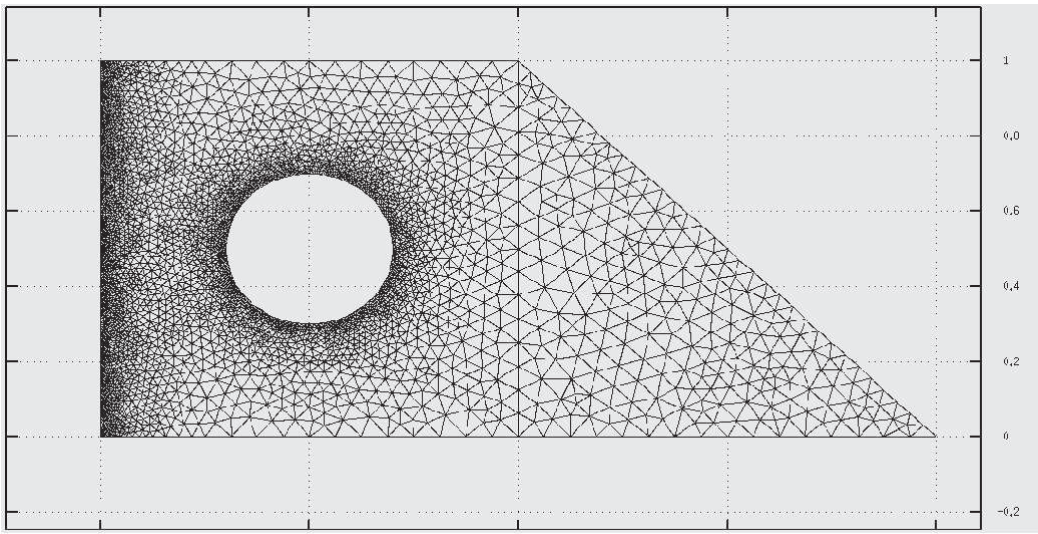


Figura 6.3 Malha do fenômeno A

A Figura 6.4 é a malha do fenômeno B. Este fenômeno está atuando nas superfícies (id's 8 e 21) (ver Figura 6.1(b)):

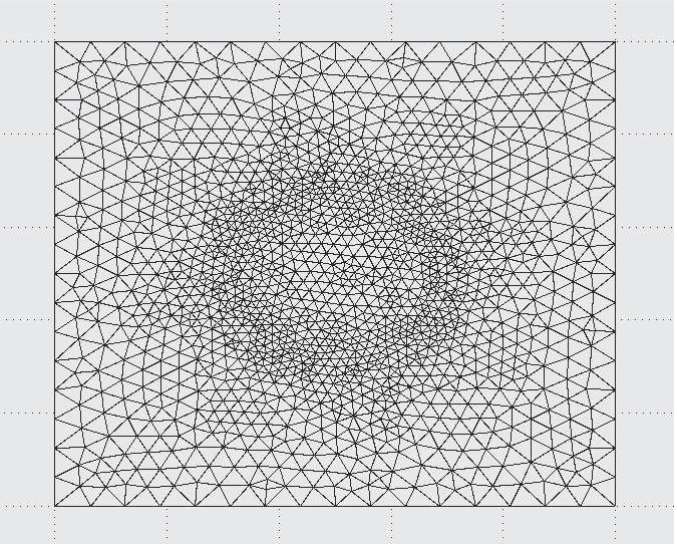


Figura 6.4 Malha do fenômeno B

Na Figura 6.5, consideramos o compartilhamento de malhas, ou seja, a malha do fenômeno B compartilha malha com fenômeno A nos contornos 6 e 7 para este caso (outras partes da geometria poderia também existir compartilhamento). Observe que nestes contornos as malhas de ambos fenômenos são idênticas, e nas demais regiões são diferentes. A primeira malha de qualquer fenômeno que for gerada primeira em relação as malhas que serão compartilhadas, será a referência, ou seja, não haverá modificação nas regiões onde acontecerá o compartilhamento. O compartilhamento é mútuo, porém a malha referência fica fixa e as demais se adequarão as regiões compartilhadas.

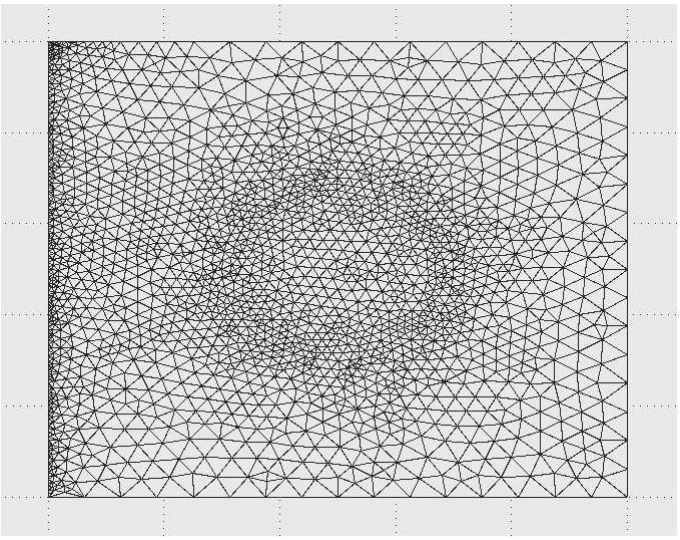


Figura 6.5 Malha compartilhada

6.2 Estudo de caso: Facilidades e dificuldades no desensolvimento de simuladores para simulação de sistemas multi-físicas

Nesta seção será apresentado um estudo de caso com o aluno Felipe Augusto Cruz, que faz parte do Programa de Pós-graduação em Engenharia Mecânica na Área de Projetos/Mecânica Computacional na modalidade Mestrado. Felipe está inserido no projeto de desenvolvimento de simuladores para simulação de sistemas multi-físicas pelo método do Elemento Finito e tem como orientador o Professor José Maria Andrade Barbosa.

O objetivo deste estudo de caso é analisar as facilidades e dificuldades obtidas nas implementações utilizadas no desenvolvimento de simuladores do MPhyScas. Segue abaixo uma sequência de perguntas e respostas:

- *Pergunta*: Desde quando você faz parte deste projeto?
  - *Resposta*: Desde agosto de 2010.
- *Pergunta*: Qual foi a sua maior motivação que levou você a participar deste projeto?
  - *Resposta*: Acessando sites científicos e lendo diversos artigos publicados de engenharia, pude perceber que sistemas multi-físicas é um assunto bastante atual e de uma enorme abrangência para os problemas de engenharia. Observando esta importância e a possibilidade de participar de um programa, que desenvolveria simuladores para os sistemas acima referido achei algo bastante desafiador e resolvi ingressar nesta jornada.
- *Pergunta*: Você já participou de algum outro projeto anterior a este. Se sim, isto facilitou na sua capacidade de desenvolvimento deste projeto, por quê?
  - *Resposta*: Sim, participei de um projeto que desenvolveu uma modelagem física e matemática da dinâmica de um guindaste utilizado em plataformas off-shore. Isto me ajudou bastante, pois tive a possibilidade de aprender algumas linguagens de programação (C, C++, Pyton, entre outras), alguns softwares de visualização gráficas e pude aprimorar um raciocínio lógico. Tudo isto contribuiu para que eu adquirisse capacidade necessária para desenvolver um projeto de uma relavante complexidade.
- *Pergunta*: Todos os passos relacionado ao desenvolvimento do projeto foi planejado de forma sistemática e o planejamento foi mantido até o final do projeto?
  - *Resposta*: Houve sim um planejamento inicial, porém no decorrer das implementações surgiram algumas dificuldades que necessitaram de mundanças no planejamento para adequação do problema.
- *Pergunta*: Qual foi a sua primeira atividade relacionada ao projeto?
  - *Resposta*: Estudar as teorias que seriam aplicadas no desenvolvimento dos problemas relacionados aos sistemas multi-físicas, por exemplo, elasticidade com difusão.

- *Pergunta*: Qual o primeiro simulador construido por você?
  - *Resposta*: O primeiro simulador, o qual construi foi o *SimulatorIn*, que é um simulador de problemas relacionado à elasticidade. Este simulador foi utilizado posteriormente no auxílio da construção de outro simulador *SimulatorEldiff* que envolve elasticidade e difusão.
- *Pergunta*: Houve alguma evolução apartir da construção do primeiro simulador?
  - *Resposta*: Sim, pois no primeiro Simulador os pârametros de cada fenômeno eram especificados nos *PhenParameters*, além disto, os parâmetros eram dados de entrada nas *WeakForms*. Isto não era viável, pois para alterar algum parâmetro, como por exemplo, o módulo de elasticidade, coeficiente de poisson, entre outros, seria necessário fazer alterações em todas as *WeakForms* que receberiam esses dados como parâmetros de entrada. Já nos Simuladores seguintes as alterações eram feitas em apenas em um único lugar, ou seja, apenas os valores definidos dentro dos *PhenParameters*, sendo este último, agora como dado de entrada nas *WeakForms*.
- *Pergunta*: Houve diferença no tempo de execução do programa quando existiram alterações nos dados da simulação?
  - *Resposta*: Sim, quando é realizado o refinamento da malha (o tempo de execução aumenta); quando é alterado os parâmetros do resolvedor (pré-condicionador, método utilizado, números de iterações e tolerância, entre outros.), para este caso o tempo de execução irá variar de acordo com qual ou quais destes parâmetros forem alterados.
- *Pergunta*: Na sua opinião, o que poderia ser realizado para diminuir consideravelmente o tempo de excecução do programa?
  - *Resposta*: A paralelização do MPhyScas irá reduzir bastante o tempo de execução.
- *Pergunta*: Qual a biblioteca que foi utilizada para os cálculos de operações envolvendo escalares, vetores e matrizes? e por que da escolha desta biblioteca em relação as outras?
  - *Resposta*: GMM++, pois esta biblioteca possui Solvers e de fácil implementação. Isto facilitou bastante nosso trabalho e obtivemos ótimos resultados nas simulações.
- *Pergunta*: Qual foi a maior dificuldade que você se deparou ao longo do desenvolvimento do trabalho?
  - *Resposta*: Apesar do MPhyScas possuir um sistema de Checkup satisfatório (eficiente) no que diz respeito a tempo de teste, existem erros que estão relacionados à formulação do problema, ou seja, pequenos erros ocorrido na formulação do problema e que leva um bom tempo para serem identificados e corrigidos.
- *Pergunta*: Durante a realização do trabalho existiu alguma evolução no que diz respeito a tempo de desenvolvimento?

– *Resposta*: Sim, após a conclusão do primeiro simulador adquiriu-se uma certa experiência, que foi observada no desenvolvimento dos outros simuladores. Houve uma evolução relacionada ao tempo de desenvolvimento, pois os mesmos erros acontecidos nos códigos dos simuladores anteriores eram identificados e corrigidos com uma certa facilidade devido a experiência adquirida.

Realizando uma análise nas dificuldades acima apresentada, chegou-se a uma conclusão que o simulador desenvolvido para simulação de sistemas elasto-plástico difusivo é satisfatório no que diz respeito a reusabilidade, ou seja, pequenas alterações no código dos algoritmos não possibilita uma extensa reprogramação em todo o código. Isto é importante para sistemas multi-físicas acoplado, onde um determinado fenômeno atuando em um certo domínio geométrico é dependente de outro fenômeno, por isso é necessário uma representação computacional (abstração) eficiente.

Chegou-se também a conclusão que é necessário a paralelização dos processos no MPhyScas, a fim de de possibitar uma diminuição no tempo de execução das várias tarefas que são realizadas no baixo nível.

6.3 Contribuição para trabalhos futuros

Este trabalho proporcionará uma certa contribuição para o andamento no desenvolvimento do Programa MPhyScas (trabalhos futuros), visto que foi gerada uma documentação relativa a:

- A organização da estrutura de dados, tanto para os dados de entrada da construção do Simulador e da Simulação, quanto para os dados geométricos (geometria, elemento de referência, funções de forma, entre outros.);

- A organização dos procedimentos de integração numérica e a definição de uma interface que permite realizar operações de álgebra linear.

Com esta documentação facilitará o desenvolvimento das implementações dos problemas que envolvam estruturas tridimensionais, pois está documentado toda estrutura de dados para o problema acima mencionado.

Esta documentação também facilitará na implementação do MPhyScas Paralelo, pois está descrita a definição da malha geométrica e da malha de cada fenômeno que poderá ser utilizada como malhas particionadas na programação paralela.

A partir de agora é possível projetar um melhor suporte no uso das funções de forma, ou seja, permitir que o usuário forneça uma equação, e o sistema a partir da característica do campo vetorial possa calcular e armazenar os dados extrínsecos de integração, que serão utilizados na integração numérica de uma quantia de interesse.

CAPÍTULO 7

Conclusões e Trabalhos Futuros

7.1 Conclusões

Foi observado que os simuladores baseados no Método do Elemento Finito (MEF) podem ser moldados em uma arquitetura de camadas: laços iterativos globais, articulação de resolvedores, resolvedores e fenômenos. Esta divisão é importante, mas no entanto não fornece uma visão de como pode ser descrita a dependência entre as camadas, o que é muito importante na definição de abstrações para padronizar a forma como processos e dados nas camadas se comportam e interagem. Foi mostrado que problemas multi-físicas acoplados tornam o desenvolvimento de simuladores bastante complexo devido ao fato de que a produção e montagem de matrizes e vetores locais para cada fenômeno podem estar acopladas com outros fenômenos, significando que o cálculo das quantias necessita de informações de outros fenômenos, onde essas informações são definidas no nível da solução, fazendo com que mudanças no algoritmo de solução possam exigir extensa reprogramação em ambas as camadas.

Com o intuito de contornar estas dificuldades, foi apresentada uma forma de representar computacionalmente os dados para simulação de sistemas multi-físicas pelo MEF, objetivando uma interação entre as camadas do MPhyScas e evitando que uma simples alteração nos dados dos algoritmos gere uma extensa reprogramação nas demais camadas.

Foram apresentadas as estruturas de dados geométricos (representação da geometria, malha geométrica e do fenômeno, funções de forma, etc), as operações no baixo nível: desde a integração numérica até as operações álgebra linear e resolvedores de sistemas algébricos lineares.

Nos resultados foram apresentados visualizações gráficas de algumas malhas geométricas que foram geradas de acordo com os dados apresentados. Foi realizada uma análise das dificuldades e facilidades nas implementações do desenvolvimento de simuladores e das simulações, através de um estudo de caso com o aluno Felipe Algusto Cruz, que faz parte do Programa de Pós-graduação em Engenharia Mecânica na Área de Projetos/Mecânica Computacioanal na modalidade Mestrado. O mesmo está inserido no projeto de desenvolvimento de simuladores para simulação de sistemas multi-físicas pelo método do Elemento Finito.

Foi concluido que o simulador desenvolvido para sistemas multi-físicas é satisfatório. Concluiu-se também a necessidade da paralelização dos processos realizados no MPhyScas para diminuir o tempo de execução das tarefas no baixo nível.

7.2 Trabalhos Futuros

No nível da arquitetura, será implementado uma nova versão para o sistema, considerando:

- Desenvolvimento de uma interface gráfica.

No nível das implementações, pretende-se:

- Implementar o paralelismo do MPhyScas;
- Implementar problemas tridimensionais;
- Considerar problemas mais complexos, por exemplo, evolução do dano em materiais devido aos carregamentos mecânicos e químicos, entre outros.

Referências Bibliográficas

[1] ROSS, C. T. F. Finite element programs in structural engineering & continuum mechanics. Albion Engineering Science Series, 1996.

[2] SANTOS, F. C. G.; BRITO, E. R. R. J.; BARBOSA, J. M. A. Coping with data dependence and sharing in the simulatin of coupled phenomena. *International Congress on Computational and Applied Mathematics - ICCAM*, Leuven, Belgium, I, 2006.

[3] COMMITTEE on Theoretical and Applied Mechanics. *A report of the United States Association for Computational Mechanics*, Research Directions in Computational Mechanics, 2000.

[4] SANTOS, F. C. G.; BRITO, E. R. R. J.; BARBOSA, J. M. A. Dealing with coupled phenomena in the finite element method. *XXVII Latin American Congress on Computational Methods in Engineering*, Belém, Brasil, p. 461, 2006.

[5] SANTOS, F. C. G.; BRITO, E. R. R. J.; BARBOSA, J. M. A. Phenomenon computational pattern: coupling relationship between phenomena on multi-physics simulation. *Industrial Simulation Conference 2006*, Palermo, Italy, I, p. 182–187, 2006.

[6] REDE de Pesquisa Cooperativa em Modelagem Computacional. Disponível em: <http://www.demec.ufpe.br/rpcmod/historico.htm>.

[7] SEED, G. An introduction to object-oriented programming in c++: With application in computer graphics. Springer, 1996.

[8] BRITO, E. R. R. J. *Desenvolvimento de um simulador para problemas Multi-físicas*. Tese (Mestrado em Engenharia Mecanica), 2007.

[9] REDDY, J. N.; GARTLING, D. K. The finite element method in heat transfer and fluid dynamics. CRC Press, 1994.

[10] LIU, C. et al. Deformation behavior of solid polymer during hot embossing process. *Microelectronic Engineering*.

[11] BLANK, M.; GOODMANB, R. Electromagnetic fields stress living cells. *Pathophysiology*.

[12] LENCASTRE, M. *Conceptualisation of an Environment for the Development of FEM Simulators*. Tese (Doutorado em Ciências da Computação) — Universidade Federal de Pernambuco, Recife, Pernambuco, 2004.

[13] SANTOS, F.; LENCASTRE, M.; RODRIGUES, I. Fem simulator based on skeletons for coupled phenomena. *Proceedings of the 2nd Latin American Conference on Pattern Languages of Programming*, Brazil, I, 2002.

[14] SANTOS, F.; LENCASTRE, M.; VIEIRA, M. Workflow for simulators based on finite element method. *Proceedings of the International Conference on Computational Science (ICCS)*, Melbourn, Australia and Saint Petersburg, Russia, 2003.

[15] SANTOS, F.; LENCASTRE, M. An approach for fem simulator development. *Journal of Computational ans Applied Mathematics*, Holanda, v. 185, n. 2, p. 326–346, 2006.

[16] ERINGEN, A. C.; G., E. E. On nolocal elasticity. *International Journal of Engineering Science*, n. 10, p. 233–248, 1972.

[17] DRIEMEIER, L. *Contribuição ao estudo da localização de deformação associada ao emprego de modelos constitutivos de dano e plasticidade*. Tese (Doutorado em Engenharia Mecânica) — Escola de Engenharia de São Carlos, São Paulo, 1999.

[18] CHOPIN, C. E.; P., T. J. Modeling of coupled deformation-diffusion in non-porous solids. *International Journal of Engineering Science*, Elsevier Science Ltd, n. 37, p. 1–24, 1999.

[19] KONG, X. A. A data design approach for object-oriented. *Computing and Structure*.

[20] LIRA, P. R. M.; CARVALHA, D. K. E. A computational methodology for automatic two-dimensional anisotropic mesh generation and adaptation.

[21] BAKER, T. J. Developments and trends in three dimensional mesh generation. *Applied Numerical Mathematics*, v. 5, p. 275–304, 1989.

[22] CELES, W.; PAULINO, G. H.; ESPINHA, R. A compact adjacency-based topological data structure for finite element mesh representation. *INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING*, USA, v. 64, p. 1529–1556, 2005.

[23] DEVLOO, P. R. B. On the definition of high order shape functions for finite elements. 1999.

[24] SOLIN, P.; KAREL, S.; IVO, D. Higher-order finite element methods. p. 42–43, 2004.

[25] SOLIN, P.; KAREL, S.; IVO, D. Higher-order finite element methods. p. 180–188, 2004.

[26] GEORGE, P. Automatic mesh generation: Application to finite element methods. New York, USA, p. 344, 1991.

[27] FREY, P. J.; GEORGE, P. L. Mesh generation application to finite elements. Hermes Science, 2000.

[28] MAGALHAES, G. M.; PASSARO, A.; ABE, M. N. Geração de malha de delaunay orientada a objetos. *Workshop de Computação*, São José dos Campos, SP, v. 17, p. 73–79, 2000.

[29] BABUSKA, I.; UDAY, B.; JOHN, E. O. On principles for the selection of shape functions for the generalized finite element method. *Computer methods in applied mechanics and engineering*.

[30] STROUSTRUP, B. A linguagem de programação c++. Bookman, 2002.

[31] JACK, J. D.; VICTOR, E. Numerical linear algebra algorithms and software. *Journal of Computational and Applied Mathematics*.

APÊNDICE A

Algoritmo para o Problema Elasto-plástico com Difusão

Neste apêndice será explicitado o algoritmo para resolução do problema elasto-plástico com difusão, demonstrado abaixo. A Figura A.1, apresenta a hierarquia entre as camadas, onde:

- **Kernel:** representado por **KRNL** e seu algoritmo **KRNLALGTH**;
- **Block:** representado por **BLK***i*, $i = 0, 1$ e seus algoritmos **ALGTH**;
- **Group:** representado por **Group** j, $j = 0, 1, 2$;
- **GroupTask:** representado por **GT***k*, $k = 0, ..., 9$, pertencente ao **Group0**, **GT***l*, $l = 0, ..., 12$, pertencente ao **Group1** e **GT0** pertencente ao **Group2**;
- **Phenomenon:** representado por **Phen***m*, $m = 0, 1, 2$;
- **ExecCode:** representado por **X***n*, $n = 0, ..., 9$, pertencente ao **Phen0**, **X***o*, $o = 0, ..., 14$, pertencente ao **Phen1** e **X***p*, $p = 0, 1, 2$, pertencente ao **Phen2**;
- **WeakForm:** representado por **WF***q*, $q = 0, ..., 6$, pertencente ao **Phen0**, **WF***r*, $r = 0, ..., 6$, pertencente ao **Phen1** e **WF***s*, $s = 0, 1$, pertencente ao **Phen2**.

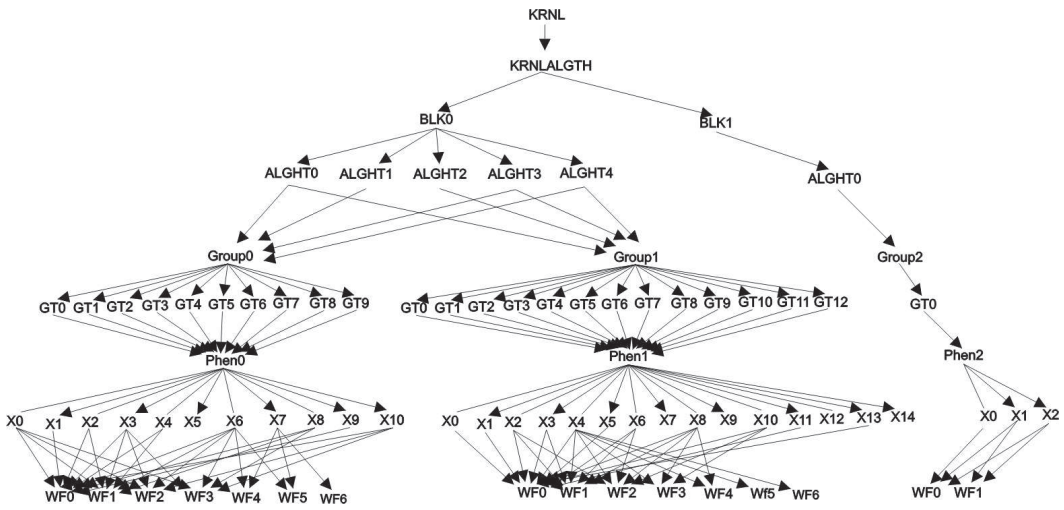


Figura A.1 Hierarquia das camadas

- GetParamtr(Interval I);

- BLK 0 ALGTH 0 - Initialize

(Group1 - GT0 - X0)
- Initialize $\mathbf{C}_n = \mathbf{C}_{ini}$;
-
(Group0 - GT0 - X0)
- Initialize $\mathbf{U}_n = \mathbf{U}_{ini}$;
- Initialize $X_n = X_{ini}$; (At the integration points)
- Initialize $\varepsilon_n^p = \varepsilon_{ini}^p$; (At the integration points)
- Initialize $p_n = p_{ini}$; (At the integration points)

- BLK 0 ALGTH 1 - Calculate Δt_b

- Calculating the Δt_b initial;

$\Delta t = \Delta t_b$;

- BLK 0 ALGTH 2 - CALCULATE $\dot{C}_n$

(Group1 - GT2)
- Zero $M = 0$;
- Zero $B = 0$;
- Zero $G = 0$;
-
(Group1 - GT3 - X0)
- Calculate $Gr = \nabla U_n$;
- Calculate $tr(\sigma)$;
-
(Group1 - GT3 - X1)
- Calculate $B \quad += \int_e \rho_i \varphi_J d\Omega$;
- Calculate $B \quad += \int_e D[\nabla \varphi_J^T \nabla \varphi_i d\Omega]$;
- Calculate $B \quad += \int_e D[\frac{e^2(\lambda+\mu)}{\alpha} \nabla \varphi_J^T \nabla \varphi_i \mathbf{C}_n d\Omega]$;
- Calculate $\mathbf{B} \quad += -\int_e \frac{C}{\alpha} D[\nabla \varphi_J^T . \nabla \mathrm{tr}\sigma] \varphi_i d\Omega$;
- Calculate $\mathbf{G} \quad += \int_e g \varphi_J d\Omega$;
- Calculate G = blas_Mvaxpy(target(G),Mat(B), vec$_1$(C_n), vec$_2$(G),$\alpha(-1)$,$\beta(1)$);
- Introduce Boundary conditions for $\dot{\mathbf{C}}_n$;(modify M and G)
-
(Group1 - GT4 - X2)
- Solve $\mathbf{M}\dot{\mathbf{C}}_n = \mathbf{G}$;

$t_1 = t_0 = 0.0$;

- BLK 1 ALGTH 0 - Record

(Group2 - GT0 - X0)
- Record mesh;
-
(Group2 - GT0 - X1)
- Record C_n;
- Record U_n;
-
(Group2 - GT0 - X2)
- Record t_1 and Δt;

While $(t_1 < I)$

$t_1 = t_0 + \Delta t$;

- BLK 0 ALGTH 3 - Zero, Swap

(Group0 - GT2)
- Zero $K = 0$;
- Zero $F = 0$;
-
(Group1 - GT5)
- Zero $M = 0$;
- Zero $B = 0$;
- Zero $G = 0$;
-
(Group1 - GT6 - X3 - gPh1)
- Update $C_K = C_n$;
-
(Group0 - GT3 - X1 - gPh0)
- Update $U_K = U_n$;
- Update $X_K = X_n$; (At the integration points)
- Update $\varepsilon_K^p = \varepsilon_n^p$; (At the integration points)
- Update $p_K = p_n$; (At the integration points)

- BLK 0 ALGTH 4 - Elasticity, Plasticity .....

tol;
$tol_{\Delta p}$;
$nItersMax$;
$nItersMax_{\Delta p}$;

npt;
- Initialize err = 1.0;
While $((err > tol)\&\&(nIters \leq nItersMax))$

- BLK 0 ALGTH 4.1 - Elasticity .....

(Group0 - GT4 - X2 - W0)
- Calculate $\mathbf{K} \quad += \int_e \varepsilon^T(\Psi_J).\mathbf{C}.\varepsilon(\Psi_i)d\Omega$;
- Calculate $\mathbf{F} \quad += \int_e \Psi_J.\mathbf{b}.d\Omega$;
- Calculate $\mathbf{F} \quad += \int_e \mathrm{e}(3\lambda+2\mu)(C_K - C_0)tr(\nabla\Psi_i)\,d\Omega$;
- Calculate $\mathbf{F} \quad += \int_e 2\mu\varepsilon_K^p : \nabla\Psi_i d\Omega$; (At the integration points)
- Introduce Boundary conditions (modify K and/or F);
(Group0 - GT4 - X3 - gPh0)
- Solve $\mathbf{K}.\mathbf{U}_K = \mathbf{F}$ (Calculating $\mathbf{U}_K$);
(Group0 - GT5 - X4 - W1)
- Calculate $Gr = \nabla U$;
- Calculate $\varepsilon_K = \frac{1}{2}\left(Gr + Gr^T\right)$;
- Calculate $te = tr(\varepsilon_K)$;
- Calculate $\sigma_K = \lambda teI + 2\mu\varepsilon_K$;
- Calculate $S_q = \sigma_K - e\,(3\lambda+2\mu)\,(C_K - C_0)\,I$; (At the integration points)
- Calculate $tr(\sigma) = \sigma_{tr}$;

- BLK 0 ALGTH 4.2 - Plasticity .....

(Group0 - GT6 - X5 - W2)
- Zero $ipVec$
- Loop on all the integration points
    - Calculate $S_K = S_q - 2\mu\varepsilon_K^p$;
    - Calculate $ts = tr\,(S_K)$;
    - Calculate $S_K = S_K - \frac{1}{3}tsI$;
    - Calculate $L = (S_K - X_K)$;
    - Calculate $J = \sqrt{\frac{3}{2}L : L}$;
    - Calculate $z_0 = e^{-bp_K}$;
    - Calculate $r_K = d\,(1 - z_0)$;
    - Calculate $f = J - \sigma_y - r_K$;
    If $f < 0$
        - Update $X_{(K+1)} = X_K$;
        - Update $\varepsilon^p_{(K+1)} = \varepsilon^p_K$;
        - Update $p_{(K+1)} = p_K$;
    Else
        - Initialize nIters$_{\Delta p}$ = 0;

- Initialize $\mathrm{err}_{\Delta p} = 1.0$;
- Calculate $N_K = \frac{L}{J}$;
- Calculate $\Delta\varepsilon$ =blas_AddMat(target($\Delta\varepsilon$),$\mathrm{Mat}_1(\varepsilon_K)$, $\mathrm{Mat}_2(\varepsilon_n)$,$\alpha(1)$,$\beta(-1)$);
- Calculate $A1 = (N_K : X_K)$
- Calculate $tc = tr(\Delta\varepsilon)$;
- Calculate $A2 = tcI - 3\Delta\varepsilon$
- Calculate $A3 = A2 : N_K$;
- Calculate $\Delta p_{cur} =$

$$= \frac{\mu A3}{\left[-\frac{9}{2}\left(\mu + \frac{1}{2}\right) + \frac{3}{2}\frac{\varphi}{a}A1 - dbz_0\right]}$$

- Calculate $te = tr(\varepsilon_K)$;
- Calculate $S1' = 2\mu\varepsilon_K - \frac{2}{3}teI$;
- Calculate $DS = -2\mu N_K$
- Calculate $z_1 = \sigma_y + d$;
- $ipVec[indice] = 1$;

End

End of loop

-

(Group0 - GT6 - X5 - W3)

- *Virtual_LisPlas* from *ipVec*;

While $((Virtual\_LisPlas! = \phi)\&\&(\mathrm{nIters}_{\Delta p} \leq \mathrm{nItersMax}_{\Delta p}))$

For $(it = Virtual\_LisPlas.begin(); it! = Virtual\_LisPlas.end(); it++)$ (At the integration points)

- Calculate $z_2 = p_K + \Delta p_{cur}$
- Calculate $YY = z_1 - de^{-bz_2}$
- Calculate $SS = S1' + \Delta p_{cur} DS$
- Calculate $z_3 = 1 + \frac{\varphi}{a}\Delta p_{cur}$
- Calculate $XX = X_K + N_K \Delta p_{cur}$
- Calculate $XX = \frac{1}{z_3}XX$
- Calculate $LL = (SS - XX)$;
- Calculate $JJ = \sqrt{\frac{3}{2}LL : LL}$
- Calculate $F1 = JJ - YY$
- Calculate $DY = dbe^{-bz_1}$
- Calculate $DX = \frac{\varphi}{a}\frac{XX}{z_3} + \frac{N_K}{z_3}$
- Calculate $DX = DS - DX$
- Calculate $DJ = \frac{3}{2}(N_K : DX)$
- Calculate $F2 = DJ - DY$
- Calculate $\Delta p_{new} = \Delta p_{cur} - \frac{F1}{F2}$
- Calculate $err_{\Delta p_{new}} = \frac{|\Delta p_{new} - \Delta p_{cur}|}{|\Delta p_{new}|}$

If $(err_{\Delta p_{new}} < tol_{\Delta p})$

$ipVec[indice] = 0$;

End

- Update $\Delta p_{cur} = \Delta p_{new}$;

End

n Iters$_{\Delta p}$ $++$;

End

If (nIters$_{\Delta p}$ $>$nItersMax$_{\Delta p}$)

- break simulation

End

-

(Group0 - GT6 - X5 - W4)

Recuperar $ipVec, \Delta p_cur, ...$

int $sz = ipVec->size()$

For (int $i = 0; i < sz; i++$) (At the integration points)

- Calculate $\Delta\varepsilon^p = N_K \Delta p_{cur}$;
- Calculate $\varepsilon^p_{(K+1)} = \varepsilon^p_n + \Delta\varepsilon^p$;
- Update $p_{(K+1)} = p_n + \Delta p_{cur}$;
- Update $X_{(K+1)} = X_n \left(1 - \frac{\varphi}{a}\Delta p_{cur}\right) + \Delta\varepsilon^p$;

End

- BLK 0 ALGTH 4.3 - Assembly System

(Group1 - GT7 - X4)

- Calculate $\mathbf{M} \quad += \int_e \Psi_i \varphi_J d\Omega$;
- Calculate $\mathbf{B} \quad += \int_e D[\nabla\varphi_j^T \nabla\varphi_i d\Omega]$;
- Calculate $\mathbf{B} \quad += \int_e D[\frac{e^2(\lambda+\mu)}{\alpha}\nabla\varphi_j^T \nabla\varphi_i \mathbf{C}_K d\Omega]$;
- Calculate $\mathbf{B} \quad += -\int_e \frac{C}{\alpha} D[\nabla\varphi_j^T . \nabla tr(\sigma_K)]\varphi_i d\Omega$;
- Calculate $\mathbf{G} \quad += \int_e g\varphi_J d\Omega$;

- BLK 0 ALGTH 4.4 - Calculate C_K

$par = \frac{2}{\Delta t}$;

-

(Group1 - GT8 - X5 - gPh1)

- Calculate M =blas_AddMat(target(M),Mat$_1$(M), Mat$_2$(B),$\alpha(par)$,$\beta(1)$);
- Calculate C_{aux} =blas_AddVec(target(C_{aux}), vec$_1$(C_n), vec$_2$($\dot{C}_n$),$\alpha(par)$,$\beta(1)$);
- Calculate G =blas_Mvaxpy(target(G),Mat(M), vec$_1$(C_{aux}), vec$_2$(G), $\alpha(1)$, $\beta(1)$);

-

(Group1 - GT8 - X6 - W0)

- Introduce Boundary conditions (modify $\mathbf{M}$ and $\mathbf{G}$);

-
(Group1 - GT8 - X7 - gPh1)
- Solve $\mathbf{MC}_K \quad = \mathbf{G}$;

- BLK 0 ALGTH 4.5 - Calculate Error

(Group0 - GT7 - X6 - gPh0)
- Calculate $D = U_{K+1} - U_K$;
- Calculate $a = ||D||$;
- Calculate $b = ||U_{K+1}||$;
- err $U_{K+1} \quad = \frac{a}{b}$;
-
(Group1 - GT9 - X8 - gPh1)
- Calculate $D = C_{K+1} - C_K$;
- Calculate $a = ||D||$;
- Calculate $b = ||C_{K+1}||$;
- err $C_{K+1} \quad = \frac{a}{b}$;
-
- err = max $\{$ err $U_{K+1}, \quad$ err $C_{K+1}\}$;

- BLK 0 ALGTH 4.6 - Swap (U_{K+1}, C_{K+1})

(Group0 - GT8 - X7 - gPh0)
$U_{K+1} \quad = U_K$;
-
(Group1 - GT9 - X10 - gPh1)
$C_{K+1} \quad = C_K$;

$nIters++$;

End

If $(nIters < nItersMax)$

- BLK 0 ALGTH 4.7 - Calculate $\dot{C}_n$

(Group1 - GT10 - X11 - gPh1)
- Calculate C_{aux1} =blas_AddVec(target(C_{aux1}), vec$_1$($C_{(K+1)}$), vec$_2$(C_n),$\alpha(1)$,$\beta(-1)$);
- Calculate $\dot{C}_n$ =blas_AddVec(target($\dot{C}_n$), vec$_1$(C_{aux1}), vec$_2$($\dot{C}_n$),$\alpha(par)$,$\beta(-1)$);

- BLK 0 ALGTH 4.8 - Swap $(U_n, C_n, X_n, \varepsilon_n^p, p_n)$

(Group1 - GT11 - X12 - gPh1)

- $\mathbf{C}_n = \mathbf{C}_{(K+1)}$;
-
(Group0 - GT9 - X8 - gPh0)
- $\mathbf{U}_n = \mathbf{U}_{(K+1)}$;
- $X_n = X_{(K+1)}$; (At the integration points)
- $\varepsilon_n^p = \varepsilon_{(K+1)}^p$; (At the integration points)
- $p_n = p_{(K+1)}$; (At the integration points)

Else

Break Simulation

End

- BLK 1 ALGTH 0 - Record

(Group2 - GT0 - X0)
- Record mesh;
-
(Group2 - GT0 - X1)
- Record C_n;
- Record U_n;
-
(Group2 - GT0 - X2)
- Record t_1 and Δt;

- BLK 0 ALGTH 1 - Calculate Δt_b

- Calculating the Δt_b initial;

$\Delta t = \Delta t_b$;

$t_0 = t_1$;

End

APÊNDICE B

Estrutura de Dados de Entrada

Existem dois tipos de de dados de entrada: os que são usados para a construção do simulador e os que são usados para construção da simulação.

B.1 Construção do simulador

Os dados são constituídos de todas as opções de algoritmos, fenômenos e métodos desejados para o simulador, sendo definido por um arquivo feito na extensão *.xml*. Em seguida serão apresentados todos os processos realizados em cada camada do MPhyScas.

B.1.1 Kernel

Esta etapa é iniciada com a identificação do **Kernel** (*id*). E logo então são declarados os dados expostos e requeridos pelo **Kernel** através de *krnlSystDt*, contendo seus índices (*KrnlIndex*) e suas definições (*def*). Em seguida é apresentado uma lista de **Block's** pertencentes ao **Kernel**.

O algoritmo do **Kernel** (*KernelAlgth*) é identificado pelo seu "*id*"e seu código "*code*", este algoritmo possui: os parâmetros do sistema (*SysPar*) com suas definições (*def*) e seus valores (*value*), os índices locais para os dados expostos aos **Block's** (*localGlobalExposedDt*) e a definição do **Block** pelo algoritmo através de um índice local (*localId*) e sua identificação (*blkId*). Com a definição do **Block** são declarados os dados requeridos aos **Block's** (*localGlobalRequiredDt*), com suas definições (*def*) e seus índices locais(*localInd*). E também é apresentado uma lista de algoritmos (*BlkAlgthVec*) com seus índices (*indInBlk*) no **Block** em questão e seus índices locais (*localInd*). Segue abaixo a definição do **Kernel**:

```
-<Kernel id="0">
  -<krnlSystData size="5">
          <!-- data exposed by the Kernel -->
      <exposedDt krnlIndex="0" def="kb1"/> . . .
         <!-- data required by the Kernel -->
      <requiredDt krnlIndex="2" def="b01"/> . . .
   </krnlSystData>
      <!-- list of Blocks -->
   <Block id="0"/> . . .
      <!-- End list of Block ids -->
  -<KernelAlgth id="0" code="0">
```

```
      <sysPar def="nStops" value="0e0"/> . . .
         <!-- local indices for Block exposed data -->
      <localGlobalExposedDt def="kb1" localInd="0"/> . . .
     -<localGlobal blkId="0" localId="0">
            <!-- local indices for required data from this Block-->
         <localGlobalRequiredDt def="b01" localInd="0"/> . . .
            <!-- list of local id and respective BlockAlgth id -->
         <BlkAlgthVec localInd="0" indInBlk="0"/> . . .
      </localGlobal>
      .
      .
      .
   </KernelAlgth>
</Kernel>
<!-- End Kernel description -->
```

B.1.2 Block

Esta etapa é iniciada com a identificação do **Block** (*id*). E logo então são declarados os dados expostos e requeridos pelo **Block** através de *blkSystDt*, contendo seus índices (*blkIndex*) e suas definições (*def*). Em seguida é apresentado uma lista de **Group's** pertencentes ao **Block** e uma lista de estados globais (*glbSts*) para o objeto **Group** pertencente ao **Block** em questão, com sua identificação (*id*).

Os algoritmos do **Block** (*BlockAlgth*) são identificados pelos seus "*id*"e seus códigos "*code*", estes algoritmos possuem: os parâmetros do algoritmo (*algthPar*) com suas definições (*def*) e seus valores (*value*), os índices locais (*localId*) para os dados expostos dos **Block's** (*localGlobalExposedDt*) com sua definição (*def*), os índices locais (*localInd*) para os dados requeridos (*localGlobalRequiredDt*) que não vem de um **Group** com sua definição (*def*) e a definição do **Group** pelo algoritmo através de sua identificação (*grpId*). Com a definição dos **Group's** são declarados os dados requeridos aos Group's (*localGlobalRequiredDt*) com suas definições (*def*) e seus índices locais(*localInd*), e também é apresentado uma lista de objetos **GroupTasks** com seus índices (*indInGrp*) no **Group** em questão e seus índices locais (*localInd*). Segue abaixo a definição do **Block**:

```
-<Block id="0">
  -<blkSystDt size="7">
         <!-- data exposed by this Block -->
      <exposedDt def="b01" blkIndex="0"/>
      . . .
         <!-- data required by this Block -->
      <requiredDt def="kb1" blkIndex="2"/>
      . . .
   </blkSystDt>
```

```
      <!-- List of Groups and respective data, which are required by
      this Block-->
   <Group id="0"/>
   . . .
      <!-- list of codes for global states owned by Group objects in
      this Block -->
   <glbSts id="0"/>
   . . .
      <!-- list of BlockAlgths -->
  -<blkAlgth id="0" code="0">
      <algthPar def="par0" value="-1e0"/>
      . . .
         <!-- local indices for Block exposed data -->
      <localGlobalExposedDt def="b01" localInd="0"/>
      . . .
         <!-- local indices for required data that does not come from
         a Group-->
      <localGlobalRequiredDt def="kb1" localInd="0"/>
      . . .
     -<localGlobal grpId="0">
           <!-- local indices for required data from this Group -->
         <localGlobalRequiredDt def="g01" localInd="0"/>
         . . .
            <!-- list of local indices for GroupTask objects used in
            this algth -->
         <localGlobalGrpTsk localInd="0" idInGrp="0"/>
      </localGlobal>
   </blkAlgth>
  +<blkAlgth id="1" code="1">
</Block>
   .
   .
   .
```

B.1.3 Group

Esta etapa é iniciada com a identificação do **Group** (*id*). E logo então são declarados os dados expostos e requeridos pelo **Group** através de *grpSystDt*, contendo seus índices (*grpIndex*) e suas definições (*def*).

O **Gphen** (representante do **Group** no nível do **Phenomenon**) é identificado pelo seu "*id*"e seu código "*code*", ele possui: os dados requeridos (*requiredDt*) que não pertence ao **Phenomenon** com seus índices (*grpIndex*) e suas definições (*def*), os resolvedores (*solver*) com seu (*id*), seu código (*code*) e sua definição (*def*), uma lista de **Phenomenon** com sua identificação (*id*) e uma lista de **WeakForms** (*WFexec*) com as suas *execCode* e suas definições (*def*).

É apresentada uma lista de estados globais (*glbSts*) com suas identificações (*id*), suas definições (*def*), seus sub tipos (*subType*) e seus tipos (*type*) e uma lista de objetos **Phenomenon** que compartilham estados globais (*phnShring*) com seus *VectorFieldId* e identificação (*PhnId*).

Apresenta o **GroupTask** através de sua identificação (*id*) e definição (*def*). Segue abaixo a definição do **Group**:

```
-<Group id="0">-
   -<grpSystDt size="5">
         <!-- data exposed by this Group -->
      <exposedDt def="g01" grpIndex="0"/>
      . . .
         <!-- data required by this Group -->
      <requiredDt def="kb2" grpIndex="2"/>
      . . .
    </grpSystDt>
   -<gPhen id="0" code="0">
         <!-- required data that does not come from a Phen -->
      <requiredDt def="kb2" grpIndex="2"/>
      . . .
      <solver id="0" def="ex_0" code="0"/>
      . . .
         <!-- List of Phenomena -->
     -<Phenomenon id="1">
         <requiredDt def="p11" localIndex="0"/>
      </Phenomenon>
         <!-- list of WeakForms for gPhen - They are at of
         simulator -->
     -<WFexec def="allocAll" execCode="0">
       -<WeakForm id="0" def="computeSizes" code="0" ready="yes"
       index="0">
          <wfData def="ComputeSizes" flag="0" data="0"/>
        </WeakForm>
        . . .
      </WFexec>
          .
          .
          .

    </gPhen>
       <!-- list of codes for global states owned by this Group -->
    -<glbSts id="0" def="U_1">
       <glbData subType="0" type="1"/>
          <!-- list of Phen objects sharing this global state -->
       <phnShring vectorFieldId="1" phnId="1"/>
```

```
            <!-- <coupled phnId = "0" vectorFieldId = "0"/> only in the
            case of matrix -->
      </glbSts>
           .
           .
           .
     -<GroupTask id="0" def="allocInit">
         <grpTskDt def="allocAll" execCode="0" phenId="0"/>
         . . .
      </GroupTask>
      . . .
</Group>
    .
    .
    .
```

B.1.4 Phenomenon

Esta etapa é definida com a identificação do **Phenomenon** (*id*), sua definição (*def*) e seu código (*code*). E em seguida são declarados os dados expostos (*exposedDt*) e requeridos (*requiredDt*) pelo **Phenomenon** através do *phnSystDt*, contendo seus índices (*phnIndex*) e suas definições (*def*).

São apresentados os parâmetros do **Phenomenon** (*phenParam*) com sua identificação (*id*), suas definições (*def*) e seus códigos (*code*), também é a presentada a dimensão (*dims*) que contém a dimensão do espaço (*spaceDim*) e a dimensão da estrutura (*strutDim*), o campo vetorial (*vectorField*) com sua identificação (*id*), sua malha do **Phenomenon** (*phenMesh*), seu *nvDim*, seu *contDim* e seu estado global (glbSt) .

O *PhenMesh* é identificado pelo seu (*id*), pela máxima ordem do **Phenomenon** (*maxPhenOrder*), o código da função de forma (*phenShapeFuncCode*), o identificador da malha geométrica (*geomMeshId*) e o identificador do gerador da malha do **Phenomenon** (*phenMeshGenId*).

O *geomMesh* é identificado pelo seu (*id*) e o identificador do gerador da malha geométrica (*geomMeshGenId*).

O *geomMeshGenerator* é identificado pelo seu (*id*), seu código (*code*) e a máxima ordem do elemento (*maxGeomOrde*) e o código da função de forma (*geomShapeFuncCode*).

O *phenMeshGenerator* é identificado pelo seu (*id*) e seu código (*code*).

O método de integração (*integMethd*) é identificado pela máxima ordem de integração (*maxIntegOrder*) e o código do método de integração (*code*).

A **WeakForm** necessária a inicialização de um campo vetorial, possui sua definição (*def*) e sua *execCode*. Segue abaixo a definição do Phenomenon:

```
-<Phenomenon id="1" def="phen_dot" code="1">
   -<phnSystDt size="3">
          <!-- data exposed by this Phen -->
```

```
      <exposedDt def="p11" phnIndex="0"/>
         <!-- data required by this Group -->
      <requiredDt def="kb2" phnIndex="1"/>
      . . .
   </phnSystDt>
   <phenParam id="0" def="par_3" code="0"/>
   . . .
   <dims spaceDim="0" strutDim="0"/>
      <!-- For "scalars" (nvDim = 0), contDim provides dimension. -->
      <!-- OBs.: globalStateId is different from vectorFieldId -->
   <vectorField id="0" phenMesh="0" nvDim="0" contDim="1" glbSt="2"/>
   . . .
      <!-- The number of phenMeshes are known at simulator definition.
      Each VectorField should reffer to one of them or none.-->
   <phenMesh id="0" maxPhenOrder="0" phenShapeFuncCode="0"
   geomMeshId="0" phenMeshGenId="0"/>
   . . .
      <!-- The number of geomMeshes are known at simulator definition.
      Each phenMesh should reffer to one of them.-->
   <geomMesh id="0" geomMeshGenId="0"/>
      <!-- only one geomMeshGenerator -->
   <geomMeshGenerator id="0" code="0" maxGeomOrder="0"
   geomShapeFuncCode="0"/>
      <!-- There might be more than one phenMeshGenerator-->
   <phenMeshGenerator id="0" code="0"/>
      <!-- Only one integration method per Phenomenon-->
   <integMethd maxIntegOrder="0" integMethCode="0"/>
      <!-- weak form for initialization of a vector field 1 -->
  -<WFexec def="init" execCode="1">
    -<WeakForm id="4" def="init_0" code="0" ready="yes" index="0">
          <!-- if ready == Yes -> weakForm is ready to be used
          (no need for simulation data) -->
       <wfData def="vecFToInit" flag="0" data="1"/>
       . . .
     </WeakForm>
     . . .
  </WFexec>
</Phenomenon>
    .
    .
    .
```

B.2 Construção da Simulação

Na construção da simulação foi utilizado um arquivo feito na extensão *.xml*.

O *GeomSim.xml* é o arquivo que carrega a geometria fornecida para a simulação. É necessária sua identificação pelo seu (*id*), sua dimensão no espaço (*espaceDim*) e da estrutura (*strutDim*).

São definidas a quantidade de volumes (*volQtty*), de superfícies (*surfQtty*), de curvas (*curvQtty*) e de pontos (*pntsQtty*).

É fornecido o **GeomEntity 2D** (*ge2D*) com seus parâmetros (*parameters*), seu tipo (*Typ*), seu código (*code*) e sua identificação (*id*). Para a região plana, o *ge2D* fornecem tambem os filhos internos (*intChldrn*) e externos (*extChldrn*). Para os GeomEntity 1D e OD, também são fornecidos os parâmetros (*parameters*), os tipos (*Typ*), seus códigos (*code*), suas identificações (*id*).

Segue abaixo exemplo de carregamento de uma geometria (Figura B.1) utilizada na simulação.

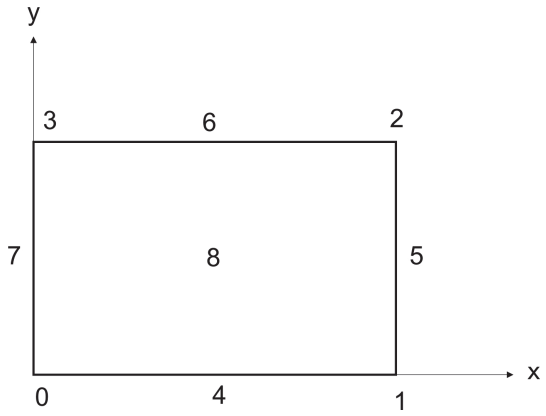


Figura B.1 Geometria de uma placa em 2D

```
 -<GeomSim id ="0" ifFile="no" spaceDim ="2" strutDim ="2">
       <!-- Global Geometry -->
     <volQtty qtty="0"/>
     <surfQtty qtty="1"/>
     <curvQtty qtty="4"/>
     <pntsQtty qtty="4"/>
-<ge2D parameters="no" typ="0" code="2" id="8">
          <!--plane region-->
       -<extChldrn>
          <childG id="4"/>
          <childG id="5"/>
          <childG id="6"/>
          <childG id="7"/>
```

```
    </extChldrn>
   -<intChldrn/>
 </ge2D>
-<ge1D parameters="no" typ="1" code="1" id="4">
       <!-- straight line-->
    <childG id="0"/>
    <childG id="1"/>
 </ge1D>
-<ge1D parameters="no" typ="1" code="1" id="5">
       <!-- straight line-->
    <childG id="1"/>
    <childG id="2"/>
 </ge1D>
-<ge1D parameters="no" typ="1" code="1" id="6">
       <!-- straight line-->
    <childG id="2"/>
    <childG id="3"/>
 </ge1D>
-<ge1D parameters="no" typ="1" code="1" id="7">
       <!-- straight line-->
    <childG id="3"/>
    <childG id="0"/>
 </ge1D>
-<ge0D parameters="yes" code="0" id="0">
       <!-- point-->
    <geomPar value="0.0e0" def="coord_x"/>
    <geomPar value="0.0e0" def="coord_y"/>
 </ge0D>
-<ge0D parameters="yes" code="0" id="1">
       <!-- point-->
    <geomPar value="1.0e0" def="coord_x"/>
    <geomPar value="0.0e0" def="coord_y"/>
 </ge0D>
-<ge0D parameters="yes" code="0" id="2">
       <!-- point-->
    <geomPar value="1.0e0" def="coord_x"/>
    <geomPar value="0.8e0" def="coord_y"/>
 </ge0D> -<ge0D parameters="yes" code="0" id="3">
       <!-- point-->
    <geomPar value="0.0e0" def="coord_x"/>
    <geomPar value="0.8e0" def="coord_y"/>
 </ge0D>
  </ge0D>
```

```
</GeomSim>
```

Após a definição da geometria, serão carregados os dados da simulação nas respectivas camadas:

B.2.1 Kernel

Nesta etapa é necessária a identificação do algoritmo (*id*) e seu código (*code*), sendo que todos os parâmetros definidos no simulador poderão sofrer alterações para uma determinada simulação. Segue abaixo a inserção no **Kernel** dos dados para a simulação.

```
     <!-- Kernel description -->
-<Kernel id="0">
  -<KernelAlgth id="0" code="1">
       <sysPar value="10e0" def="Intervalo"/> .
           .
           .
           .
   </KernelAlgth>
 </Kernel>
      <!-- End Kernel description -->
```

B.2.2 Block

Nesta etapa é necessária a identificação do algoritmo (*id*) e seu código (*code*), sendo que todos os parâmetros definidos no simulador poderão sofrer alterações para uma determinada simulação. Segue abaixo a inserção no **Block** dos dados para a simulação.

```
     <!-- Block description -->
-<Block id="0"> -
     <blkAlgth id="0" code="3">
             <!-- Initialize Datas -->
     </blkAlgth>
    -<blkAlgth id="1" code="4">
             <!-- Compute Delta_t -->
         <algthPar value="10e0" def="Intervalo"/>
            .
            .
            .
     </blkAlgth>
    -<blkAlgth id="4" code="7">
         <algthPar value="1.0e0" def="err"/>
```

```
            .
            .
            .
     </blkAlgth>
 </Block>
    .
    .
    .
```

B.2.3 Group

Nesta etapa é necessária a identificação de cada **Group** através de seu (*id*). É definido o **gPhen** com sua identificação (*id*). É necessário também a definição de cada *solver* e *WFexec*, suas identificações através de seus (*id's*) e suas definições (*def's*). Os parâmetros de cada *solver* poderão sofrer alterações em uma determinada **WeaForm**. Segue abaixo a inserção no **Group** dos dados para a simulação.

```
-<Group id="0">
   -<gPhen id="0">
       <solver id="0" def="Solver_0"/>
      -<WFexec def="Solver_0" execCode="7">
         -<WeakForm id="10" def="WF_gPhenSolver_0">
             <wfParam value="1.0e-6" def="tol"/>
                .
                .
                .
          </WeakForm>
       </WFexec>
     </gPhen>
 </Group>
-<Group id="1">
   -<gPhen id="2" code="0">
       <solver id="1" code="1" def="Solver_1"/>
       .
       .
       .
      -<WFexec def="Compute_G_Solver_1" execCode="8">
            .
            .
            .
       </WFexec>
     </gPhen>
```

```
 </Group>
-<Group id="2">
        <!-- no simulation data for this Group -->
 </Group>
   .
   .
   .
```

B.2.4 Phenomenon

Nesta etapa é necessária a identificação de cada **Phenomenon** através de seu (*id*) e seu código (*code*) e sua definição (*def*). São definidos os parâmetros do **Phenomenon** (*phenParam*), sendo necessária a identificação através de seus (*id's*), suas definições (*def's*) e seus dados (*parData*). São definidos: o grafo da geometria (*geomGraph*) e os dados de geração da malha geométrica *geomMeshGenData*, todos identificados através de seus (*id's*). São definidos também as *WeakForms*, sendo especificados seus códigos (*code*) e suas definições (*def's*). Segue abaixo a inserção no **Phenomenon** dos dados para a simulação.

```
  -<Phenomenon id="1" code="1" def="Elastic_Plastic">
     -<phenParam id="0" def="pP_Tstep0">
         <parData value="10.0e0" def="numTimeSteps"/>
      </phenParam> -<phenParam id="1" def="pP_KHard">
         <parData value="0.0e0" def="a_Kin"/>
             .
             .
             .
      </phenParam>
     -<phenParam id="2" def="pP_ElasticTensor">
         <parData value="2.1e11" def="E_young"/>
             .
             .
             .
      </phenParam>
         <!-- geometry relation for this phen -->
         <!-- The geomComponent will be composed of all geomEntities
         listed as geomRel items -->
     -<geomGraph id="0">
         <geomRel geomId="8"/>
         <boundingBox y1="4.0e0" x1="4.0e0" y0="-1.0e0" x0="-1.0e0"/>
      </geomGraph>
         <!-- geometric mesh sharing phenId = phenomenon object id;
         geomEntId = id of geometric entity where sharing takes
         place -->
```

```
  -<geomMeshShring phenId="3">
       <geomEnt geomEntId="8"/>
   </geomMeshShring>
  -<geomMeshGenData id="0">
       <gmgData def="geomTol" data="1.0e-6"/>
          .
          .
          .
   </geomMeshGenData>
  -<phenMeshGenData id="0">
       <pmgData def="initOrder" data="1.0e0"/>
           .
           .

   </phenMeshGenData>
       <!-- data for weak forms, which are dependent of geometric
       data -->
       <!-- more than one object could be created (cloned) carrying
       different geom data -->
  -<WFexec def="Compute_KF" execCode="5">
     -<WeakForm code="1" def="WF_PhenComputeK" ready="yes" newId="0"
     originalId="0" index="0">
           .
           .
           .
      </WeakForm>
   </WFexec>
       .
       .
       .
   </WFexec>
</Phenomenon>
     .
     .
     .
```