

Universidade Federal de Pernambuco  
Centro de Informática  
Graduação em Ciência da Computação

---

Trabalho de Graduação

# **Ferramenta de Visualização de Simulações baseadas no Método do Elemento Finito**

**Autor:** Mardoqueu Souza Vieira (msv@cin.ufpe.br)

**Orientador:** Alejandro C. Frery (frery@cin.ufpe.br)

**Co-orientador:** Félix C. G. Santos (fcgs@demec.ufpe.br)

Recife-PE  
12 março de 2003

Trabalho aprovado como requisito parcial para conclusão da disciplina Trabalho de Graduação em Computação Visual, disciplina do Centro de Informática da Universidade Federal de Pernambuco. A banca examinadora é composta por:

---

Prof. Alejandro C. Frery  
(frery@cin.ufpe.br)  
Centro de Informática da UFPE

---

Prof. José Dias dos Santos  
(jds@cin.ufpe.br)  
Centro de Informática da UFPE

Recife-PE  
12 março de 2003

# Sumário

<b>1</b>	<b>Introdução</b>	p. 1
<b>2</b>	<b>O Problema</b>	p. 4
2.1	Descrição . . . . .	p. 4
2.2	O Sistema Plexus . . . . .	p. 6
2.2.1	Dados Resultantes de uma Simulação . . . . .	p. 8
<b>3</b>	<b>Ferramentas para Visualização</b>	p. 9
3.1	Visualização . . . . .	p. 9
3.1.1	Terminologia . . . . .	p. 9
3.1.2	Motivação . . . . .	p. 10
3.1.3	Processamento de Imagens, Computação Gráfica, e Visualização . . . . .	p. 11
3.1.4	Visualização Científica . . . . .	p. 12
3.1.5	Caracterizando os Dados de Visualização . . . . .	p. 13
3.1.6	Técnicas de Visualização . . . . .	p. 14
3.1.6.1	Classificação das Técnicas de Visualização . . . . .	p. 14
3.1.6.2	Técnicas de Visualização de Campos Escalares . . . . .	p. 14
3.1.6.3	Técnicas de Visualização de Campos Vetoriais . . . . .	p. 15
3.1.6.4	Técnicas de Visualização de Campos Tensoriais . . . . .	p. 16
3.1.7	Realidade Virtual . . . . .	p. 16
3.2	Classificação das Ferramentas para Visualização . . . . .	p. 17

3.3	Sistemas de Visualização . . . . .	p. 18
3.3.1	Arquitetura dos Sistemas de Visualização . . . . .	p. 19
3.3.2	Interface com o Usuário . . . . .	p. 19
3.3.2.1	Interface de Linguagem de Comandos . . . . .	p. 20
3.3.2.2	Interfaces Gráficas . . . . .	p. 20
3.3.3	Gerenciamento de Dados . . . . .	p. 21
3.3.3.1	Modelo de Dados . . . . .	p. 21
3.3.3.2	Gerenciamento de Dados Internos . . . . .	p. 21
3.3.3.3	Redução de Dados . . . . .	p. 22
3.3.4	Módulos de Visualização . . . . .	p. 23
3.3.5	Fluxo de Controle . . . . .	p. 23
3.3.5.1	Sistemas Turnkey . . . . .	p. 24
3.3.5.2	Construtores de Aplicação utilizando Interface de Programação Visual . . . . .	p. 24
3.3.5.3	Construtores de Aplicação utilizando Interface de Linguagem . . . . .	p. 26
3.3.6	Sistema Gráfico . . . . .	p. 27
3.3.6.1	Gerente de Base de Dados Gráficos . . . . .	p. 27
3.3.6.2	Gerente de Visualização . . . . .	p. 27
3.4	<i>Data Explorer</i> . . . . .	p. 27
3.4.1	Arquitetura . . . . .	p. 28
3.4.2	Modelo de Dados . . . . .	p. 31
3.4.3	Modelo de Execução . . . . .	p. 35
3.4.3.1	Fluxo de Dados . . . . .	p. 35
3.4.4	Módulos . . . . .	p. 37
3.4.5	Estendendo as funcionalidades do OpenDX . . . . .	p. 38

3.4.5.1	Adicionando Funcionalidade ao <i>Visual Program Editor</i> ou a Linguagem de <i>Script</i> . . . . .	p. 39
3.4.5.2	Escrevendo um Programa <i>Stand-alone</i> usando o Modelo de Dados do <i>Data Explorer</i> . . . . .	p. 40
3.4.5.3	Escrevendo um Programa <i>Stand-alone</i> usando os Módulos do <i>Data Explorer</i> . . . . .	p. 40
3.4.5.4	Controlando o <i>Executive</i> ou a Interface com o Usuário a partir de um Programa Separado . . . . .	p. 40
<b>4</b>	<b>Método do Elemento Finito</b>	p. 42
4.1	O Problema Exato . . . . .	p. 42
4.2	Discretização do Problema . . . . .	p. 43
<b>5</b>	<b>Visualização no Plexus</b>	p. 45
5.1	Arquitetura de Visualização para o Plexus . . . . .	p. 45
5.2	Projeto dos Componentes da Arquitetura de Visualização do Plexus para o <i>Data Explorer</i> . . . . .	p. 47
5.2.1	Especificação do Componente <i>Extractor</i> . . . . .	p. 47
5.2.2	Especificação do Componente <i>Viewer</i> . . . . .	p. 48
5.2.3	<i>PlexusViewer</i> . . . . .	p. 48
5.2.3.1	Cenário: <i>Field</i> . . . . .	p. 49
5.2.3.2	Cenário: <i>Group</i> . . . . .	p. 57
5.2.3.3	Cenário: <i>Series Group</i> . . . . .	p. 57
5.2.3.4	Cenário: Dados Inválidos . . . . .	p. 59
5.2.4	Estudo de caso: Simulações de Fenômenos em Órgãos Humanos . . . . .	p. 61
<b>6</b>	<b>Contribuições e Trabalhos Futuros</b>	p. 65
6.1	Contribuições . . . . .	p. 65

6.2 Trabalhos Futuros . . . . . p. 66

# ***Lista de Figuras***

1	Plexus Overview . . . . .	p. 7
2	Arquitetura de um sistema de visualização genérico . . . . .	p. 19
3	Modelo de módulo de visualização . . . . .	p. 23
4	Em uma aplicação de visualização modular, fluxos de dados específicos podem ser definidos entre módulos . . . . .	p. 25
5	Caixa de dialogo do Controle de Visualização . . . . .	p. 30
6	Componentes Compartilhados entre Campos Diferentes . . . . .	p. 34
7	Um exemplo de programa visual equivalente ao <i>script</i> da seção 3.3.2.1	p. 36
8	Arquitetura do <i>Data Explorer</i> . . . . .	p. 38
9	Diagrama UML de Componentes e <i>Deployment</i> da Arquitetura de Visualização do Plexus . . . . .	p. 47
10	Painel de Controle do <i>PlexusViewer</i> . . . . .	p. 50
11	Resultado da Visualização . . . . .	p. 51
12	Seleção de Resultado da Visualização . . . . .	p. 52
13	Leitura de Dados . . . . .	p. 53
14	Processamento de Fields . . . . .	p. 54
15	Entrada de Dados de Controle da Visualização . . . . .	p. 55
16	Visualização de Planos de Corte . . . . .	p. 56
17	Pick . . . . .	p. 57
18	Processamento de Grupos Genéricos . . . . .	p. 58
19	Processamento de Series Group . . . . .	p. 59
20	Mensagem de erro para os dados inválidos . . . . .	p. 60

21	Resultado da Aplicação da Técnica de Visualização do Contorno	p. 62
22	Resultado da Aplicação da Técnica de Visualização de Malhas pelas Arestas . . . . .	p. 62
23	Resultado da Aplicação da Técnica de Visualização Gradiente .	p. 63
24	Resultado da Aplicação da Técnica de Visualização <i>Contouring</i> .	p. 64

# **Resumo**

Com o desenvolvimento das tecnologias computacionais, simulações numéricas têm sido cada vez mais amplamente usadas em muitos campos da sociedade, devido à necessidade cada vez maior de criar simulações cada vez mais realistas. A visualização científica tem sido muito utilizada como ferramenta para tratar e analisar a grande quantidade de dados e informação resultantes de simulações numéricas. Este trabalho especifica a arquitetura de visualização para o sistema Plexus, um ambiente de desenvolvimento de simuladores baseados no Método do Elemento Finito, em UML. A arquitetura proposta é instanciada para o uso do sistema de visualização *Data Explorer*. Esta instanciação consiste da especificação dos componentes envolvidos e do desenvolvimento de programas visuais (baseados em fluxo de dados) para o *Data Explorer*.

Palavras chave: Simulações numéricas, Visualização Científica, Método do Elemento Finito, Programação baseada em Fluxo de Dados.

# ***Abstract***

With the development of computational technologies, numerical simulations have been more and more widely used in many fields of the society, due to the increasing necessity to create more realistic simulations. Scientific visualization is used as a tool to treat and to analyze the huge amount of the data and information emerging from the simulations. This work specifies an architecture of visualization for the Plexus system, an environment of development of simulators based in the Finite Element Method, in UML. The proposed architecture is also instantiated for the visualization system *Data Explorer*, with the specification of the involved components and the development of visual programs (data flow programming).

Keywords: Numerical Simulation, Scientific Visualization , Finite Element Method, Data flow programming.

# ***1 Introdução***

Com o desenvolvimento das tecnologias computacionais, simulações numéricas tem sido mais e mais amplamente usada em muitos campos da sociedade. Técnicas de simulação não somente têm papel importante no estudo científico, mas também ocupam um lugar muito importante na educação, na indústria militar, na bioengenharia, na indústria de entretenimento e em quase qualquer área que podemos imaginar.

A análise em engenharia é o processo de pegar uma dada informação de “entrada” definindo a situação física em mãos e, através de um conjunto apropriado de manipuladores, converter aquela entrada numa forma diferente de informação, a “saída”, que provê a resposta para algumas questões de interesse.

A meta das técnicas de visualização, quando usadas em conjunto com a análise em engenharia, é prover ferramentas computacionais para os usuários verem a “entrada” e a “saída”. Ao empregar realidade virtual, o objetivo não é apenas “ver” mas, sim, interagir e desenvolver um certo grau de envolvimento com a informação.

Embora existam várias classes de problemas de análise, vamos focar na classe que tipicamente prove os maiores desafios para a visualização da saída. Nesta classe de problemas a entrada consiste de alguns domínios físicos para os quais existem condições de contorno conhecidas e condições iniciais. O objetivo da análise é determinar o comportamento de uma ou mais variáveis (representando um fenômeno físico) no domínio.

O método usual para desenvolver e fazer a análise destas informações é selecionar, ou derivar, um modelo matemático apropriado para o problema físico que possa aceitar como entrada as propriedades dos materiais, condições de contorno e condições iniciais, e produzir como saída o comportamento das

variáveis. Os modelos matemáticos produzidos por esse processo são tipicamente conjuntos de equações diferenciais parciais.

Em alguns casos simples, a solução contínua exata do problema para estas equações pode ser determinada. Todavia, na maioria dos casos tal solução contínua não é simples de se derivar analiticamente. Para essas classes de problemas, onde soluções exatas não estão disponíveis, o advento dos computadores digitais teve um impacto profundo na maneira pela qual a busca das soluções é feita. As ferramentas computacionais têm, também, impacto na forma em que se faz projetos de engenharia.

Atualmente, a maioria das análises associadas com a solução de equações diferenciais parciais sobre um domínio são feitas usando procedimentos numéricos que aproximam o problema contínuo em termos de um sistema discreto. Isto produz um conjunto grande de equações algébricas que podem ser resolvidas usando o computador.

Um grande problema enfrentado pelos usuários de técnicas de análise numérica é que o volume e a forma da informação discreta produzida não promovem uma interpretação imediata nem fácil, particularmente quando se deseja alcançar um entendimento do comportamento dos parâmetros de interesse sobre o domínio de análise. As técnicas de visualização, quando construídas de forma adequada, representam a tecnologia chave necessária para extrair a informação desejada de grandes volumes de dados discretos produzidos por procedimentos numéricos de análise.

Neste trabalho serão estudadas, propostas, implementadas e avaliadas técnicas não apenas de visualização mas também de interação, no intuito de construir um sistema que ofereça recursos de realidade virtual para a extração de informação. Estas técnicas serão trabalhadas para o contexto do sistema Plexus, uma plataforma para criação e desenvolvimento de simuladores de amplo uso nas engenharias.

Este documento está estruturado da seguinte forma: o capítulo 2 descreve o problema a ser tratado descrevendo o contexto e as restrições. O capítulo 3 apresenta os conceitos de visualização importantes para este trabalho, discute uma classificação das ferramentas/ambientes para visualização incluindo um estudo dos sistemas de visualização, e apresenta também o sistema de visualização escolhido como plataforma de trabalho. O capítulo 4 descreve

de forma sucinta o Método de Elemento Finito, fonte principal dos dados a serem visualizados. O capítulo 5 apresenta os resultados obtidos, descrevendo uma proposta de arquitetura de visualização para o sistema Plexus. o sistema de visualização *Data Explorer* é utilizado para validar a arquitetura de visualização proposta. O capítulo 6 apresenta as conclusões deste trabalho organizadas em contribuições e trabalhos futuros.

## 2 O Problema

Atualmente a complexidade das simulações numéricas é muito grande devido à necessidade cada vez maior de criar simulações cada vez mais realistas. Uma mostra disso é a especificação de uma arquitetura de composição de simulações desenvolvidas independentemente para criar simulações maiores. Esta arquitetura é conhecida como *High Level Architecture (HLA)* para simulações, e foi desenvolvida pela comunidade de simulação e financiada pelo Departamento de Defesa dos Estados Unidos. Uma descrição detalhada do HLA pode ser encontrada em (KUHL; WEATHERLY; DAHMANN, 1999).

Nesse contexto surge o Plexus um ambiente computacional dedicado ao desenvolvimento de simuladores para problemas multi-física (vários fenômenos físicos, acoplados ou não entre si, atuando simultaneamente) e multi-escala, que contempla definição, implementação, execução e visualização de simulações baseadas no Método do Elemento Finito (MEF), descrito na seção 4.

Na seção 2.1 será descrito o problema a ser tratado neste trabalho. O sistema Plexus será descrito brevemente, com as informações necessárias para o desenvolvimento deste trabalho, na seção 2.2.

### 2.1 Descrição

O problema consiste em encontrar uma arquitetura de visualização para o sistema Plexus que possibilite o acoplamento ao Plexus de uma grande quantidade de ambientes de visualização, para prover uma análise visual dos resultados de simulações. A especificação das técnicas de visualização que precisam ser implementadas para análise dos resultados de simulações do Plexus, também fazem parte do problema.

O ambiente de visualização a ser utilizado no desenvolvimento e validação

da arquitetura de visualização do Plexus deve atender a algumas restrições. Vamos descrever essas restrições em termos dos requisitos descritos a seguir:

- Deve ser capaz de representar e visualizar os dados gerados pelo Plexus. Os dados gerados pelo Plexus resultantes de simulações estão descritos na seção 2.2.
- Dever ser extensível, tanto no sentido de implementar outras funcionalidades ainda não disponíveis, por exemplo a ferramenta ser de código aberto, quanto no sentido de possibilitar a implementação de ferramentas de visualização para atender as necessidades do Plexus e para viabilizar o acoplamento com o Plexus.
- Deve oferecer a capacidade de desenvolver ferramentas características da Realidade Virtual (FRERY; KELNER, 2003), tais como, navegação facilitada e intuitiva, interação, exibição de diversas fontes de informação de forma coordenada etc.

A implementação de um sistema genérico que atenda todos esses requisitos para todos os tipos de problemas de visualização não parece ser uma linha de trabalho viável, dada a enorme variedade de problemas que cada domínio apresenta e a necessidade de construir visualizações específicas, como por exemplo, relacionadas a simulação de vôo de uma avião.

Assim sendo, neste trabalho será abordado o problema de desenvolver e adaptar ferramentas de visualização para a plataforma de visualização escolhida, que atendam parcial porém satisfatoriamente as necessidades de visualização e interação de dados que surgem do uso do Método de Elemento Finito.

A plataforma de visualização escolhida para desenvolver a arquitetura de visualização do Plexus foi o *Data Explorer*, que está descrito na seção 3.4.

Na próxima seção é descrito o sistema Plexus, o ambiente onde as simulações são desenvolvidas.

## 2.2 O Sistema Plexus

O Plexus é um sistema cujo objetivo principal é reduzir a complexidade e o custo envolvidos no desenvolvimento e na implementação de sistemas de simulação, provendo um ambiente flexível com técnicas eficientes para simulação de fenômenos acoplados.

Com relação a ser um ambiente de simulação específico, que usa o Método do Elemento Finito (MEF), descrito no capítulo 4, na solução de fenômenos acoplados, o sistema Plexus pode ser usado em vários tipos de aplicações, permitindo a modelagem de classes diferentes de simuladores, considerando um conjunto de funcionalidades pré-definidas como, por exemplo, um esquema de solução de fenômenos dependentes do tempo e adaptativos.

O sistema Plexus é dividido em 4 subsistemas, representando os processos principais. A Figura 1 ilustra a interação do usuário com os subsistemas. O subsistema 1 (*Administration/System Loading*), suporta o gerenciamento do sistema e a carga de dados gerais do sistema e metadados. O subsistema 2 (*Pre-processing*), é onde o usuário informa os dados do problema a ser simulado, e onde as estruturas dinâmicas para a simulação são construídas. O subsistema 3 (*Simulation Processing*) é onde os dados são processados para obter a solução e onde a verificação ocorre. O subsistema 4 (*Post-processing*) é onde a solução é processada para obter as quantidades de interesse para o usuário e para a visualização; este componente também trata da validação do sistema.

O sistema gerencia grandes volumes de dados, componentes previamente construídos, fenômenos, acoplamento entre fenômenos, componentes de algoritmos, definição de dados persistentes e dados para reuso de conhecimentos de simulação. Para dar suporte a níveis mais altos de abstração, flexibilidade, reusabilidade, e proteção dos dados disponíveis no Plexus, existe um Sistema de Gerenciamento de Dados (SGBD, ou DBMS por *DataBase Management System*), que mantém os dados abstratos gerais relacionados ao contexto do Plexus, aos algoritmos que fazem parte de estratégias diferentes de simulação, aos dados do problema a ser simulado e também os dados intermediários e resultantes da simulação.

A implementação de simulações no Plexus é representada com o uso de es-

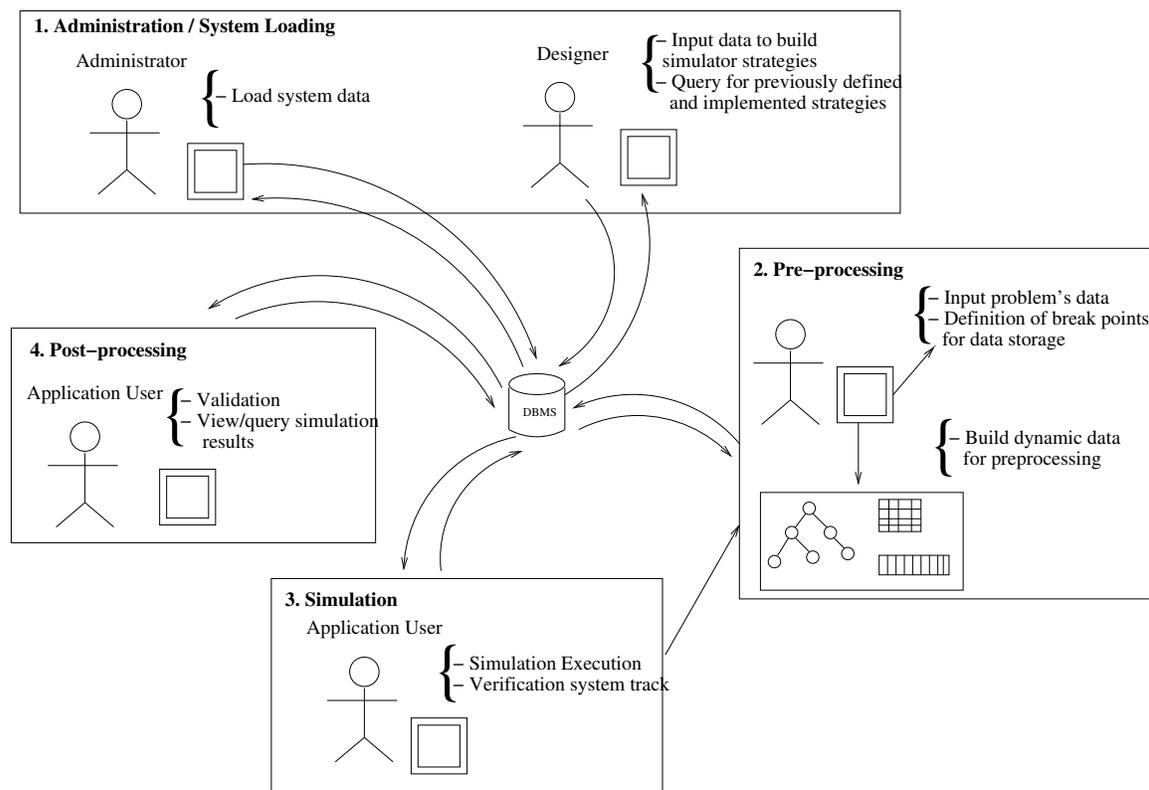


Figura 1: Plexus Overview

queletos de algoritmos e também com outras estruturas pré-definidas orientadas a objetos, como fenômenos computacionais, explorando o polimorfismo do MEF. O Plexus tem também métodos matemáticos que implementam geração de malhas, regras de integração, funções de forma (ou funções de interpolação), etc.

A representação computacional de um fenômeno físico é chamada *fenômeno computacional*. Um fenômeno computacional é descrito pelo seu campo vetorial e por formas fracas definidas nas suas entidades geométricas junto com as condições de contorno, que são também implementadas como fenômenos, chamados fenômenos fictícios, definidos na entidade geométrica respectiva a borda do domínio.

Mais informações sobre o Plexus podem ser encontradas nas seguintes referências: (SANTOS; LENCASTRE; ALMEIDA, 2001, 2002b, 2002a, 2002c; SANTOS; VIEIRA; LENCASTRE, 2003).

### **2.2.1 Dados Resultantes de uma Simulação**

Efetuar uma simulação no Plexus consiste em simular um conjunto de fenômenos físicos, onde nem todos fenômenos precisam ser visualizados. O resultado da simulação de cada fenômeno é uma malha de elementos finitos representando a geometria simulada, onde cada nó da malha está associado com um valor simulado do fenômeno. Este valor associado a cada nó da malha pode ser um escalar, um vetor, ou um tensor.

Os fenômenos podem ser dependentes do tempo, chamados fenômenos dinâmicos ou transientes, ou não, neste caso chamados de estático.

As malhas de elementos finitos podem ser adaptadas durante a simulação, tanto no caso de fenômenos dinâmicos quanto no de fenômenos estáticos.

Tendo em vista o volume das informações a serem visualizadas e a necessidade de prover ao experimentador de ferramentas para interagir com essas informações, é feita aqui uma proposta de aproveitar, propor e implementar ferramentas de realidade virtual no contexto do Plexus. Os principais conceitos de Realidade Virtual que serão relevantes para a discussão deste trabalho serão apresentados no contexto de visualização na seção 3.1.7.

## 3 *Ferramentas para Visualização*

Nesta seção vamos descrever os conceitos e técnicas de visualização.

A seção 3.1, apresenta os principais conceitos relacionados com visualização, a seção 3.2 apresenta uma classificação das ferramentas/ambientes de visualização, uma descrição dos sistemas de visualização está na seção 3.3 e uma descrição do *Data Explorer*, que foi o sistema de visualização escolhido como plataforma de visualização na seção 3.4.

### 3.1 *Visualização*

A visualização faz parte do nosso dia-a-dia, desde mapas de “tempo” a imagens geradas em jogos eletrônicos da indústria de entretenimento. Mas o que é visualização? De acordo com (SCHROEDER; KENMARTIN; LORENSEN, 1998):

“a visualização é o processo de explorar, transformar, e ver dados como imagens para obter entendimento e discernimento dos dados.”

#### 3.1.1 *Terminologia*

Terminologias diferentes são usadas para descrever a visualização. *Visualização científica* é o nome formal dado ao campo da ciência da computação que inclui interface com usuário, representação de dados e algoritmos de processamento, representações visuais, e outras apresentações sensoriais tais como som e toque.

O termo *visualização de dados* é outra denominação usada para referir-se

à visualização. A visualização de dados é geralmente interpretada como sendo mais geral do que visualização científica, uma vez que implica no tratamento de fontes de dados além das ciências e da engenharia.

Mais uma denominação surgida recentemente é *visualização de informação* (IEEE, 2003). Este campo empenha-se em visualizar informações abstratas, tais como, documentos em hipertexto na Web, estruturas de diretórios num computador, estruturas de dados, etc. Um grande desafio para os pesquisadores de visualização de informação é desenvolver sistemas de coordenadas, métodos de transformação e estruturas que organizem e representem os dados.

### **3.1.2 Motivação**

A visualização é uma ferramenta necessária para tratar e analisar a grande quantidade de informação no mundo atual. Satélites, supercomputadores, sistemas de digitalização a laser, e sistemas de aquisição de dados digitais geram, adquirem, e transmitem dados numa taxa muito alta: um satélite na órbita da terra transmite terabytes de dados diariamente, e supercomputadores são usados para modelar padrões de tempo em todo o planeta, por exemplo.

Sem a visualização, a maioria destes dados não seria “visto” nos discos dos computadores. A visualização oferece alguma esperança de extrair informações importantes escondidas nos dados.

Existe um outro elemento importante referente à visualização: ela tira vantagem das habilidades naturais do sistema de visão humano. Nosso sistema de visão é uma parte complexa e poderosa do nosso corpo. Nós usamos e confiamos nele em quase tudo que fazemos. Dado o ambiente em que nossos ancestrais viviam, não é surpresa que certos sensores se desenvolveram para ajudá-los a sobreviver.

Com a introdução dos computadores e a habilidade de gerar enormes quantidades de dados, a visualização oferece a tecnologia para fazer o melhor uso de nosso sentido visual extremamente desenvolvido. Certamente outras tecnologias tais como análise estatística e inteligência artificial, dentre outras, vão ter um papel importante em processamento de dados em larga-

escala. Todavia, dado que a visualização liga diretamente o sistema de visão e o cérebro humano, ela permanece uma tecnologia inigualável para entendimento e comunicação de dados.

A visualização oferece também vantagens financeiras em relação a outros mecanismos de extração de informação. Nos mercados competitivos de hoje em dia, simulações computacionais junto com visualização pode reduzir o custo e o tempo para entrada no mercado de um produto. Os tempos necessários para criar e testar os protótipos do produto decorrem em grandes custos no projeto de um produto. Os métodos de projeto se esforçam para eliminar estes protótipos físicos, e substituí-los com equivalentes digitais. A prototipação digital requer habilidade para criar e manipular a geometria do produto, simular o projeto sob uma variedade de condições operacionais.

### **3.1.3 Processamento de Imagens, Computação Gráfica, e Visualização**

Não há distinções claras em torno das diferenças entre processamento de imagens, computação gráfica, e visualização. Segue abaixo uma definição possível desses termos, válida no contexto deste trabalho.

**Processamento de Imagens** é o estudo de imagens bidimensionais. Isto inclui técnicas para transformar (por exemplo aplicar rotações, escalas etc.), extrair informação, analisar, e melhorar imagens.

**Computação Gráfica** é o processo de criar imagens usando um computador. Isto inclui tanto técnicas bidimensionais de pintura e desenho, quanto técnicas sofisticadas redensificação tridimensional.

**Visualização** é o processo de explorar, transformar, e ver dados como imagens para obter entendimento e discernimento dos dados.

De acordo com essas definições podemos notar que existe sobreposição entre estas técnicas. A saída da computação gráfica é uma imagem, enquanto que a saída da visualização é freqüentemente produzida utilizando a computação gráfica. De forma geral podemos distinguir visualização da computação gráfica e de processamento de imagens de três maneiras:

- A dimensão dos dados é três ou mais. Muitos métodos conhecidos estão disponíveis para dados de duas dimensões ou menos; visualização é mais útil quando aplicada em dados de grandes dimensões.
- Visualização diz respeito propriamente com transformações de dados. Ou seja, a informação é criada e modificada repetidamente para melhorar o significado dos dados.
- Visualização é naturalmente interativa, incluindo o usuário diretamente no processo de criar, transformar, e ver os dados.

### **3.1.4 Visualização Científica**

Em muitos cálculos computacionais, a *visualização científica* tem um papel importante. Em grandes simulações numéricas, o usuário não é capaz tratar eficientemente a saída da simulação sem utilizar técnicas de visualização, por exemplo. Podemos encontrar exemplos em muitas áreas de pesquisa como a indústria automobilística, a dinâmica dos fluidos de aviões, os experimentos químicos, na bioengenharia, as simulações galácticas e muitas outras.

Nas aplicações numéricas, o resultado do processamento é somente um aglomerado de dígitos, que são dificilmente entendíveis pelo ser humano. Com o propósito de compreender esses dados gerados, eles devem ser convertidos em visualizações.

Algumas empresas tiram proveito da visualização científica, com o uso de simulações computacionais ao invés de testes no mundo real. Imagine os custos de fazer centenas de testes de colisão com carros; com o auxílio da visualização, estes testes simulados podem ser executados milhares de vezes sem gastar dinheiro com novos carros.

Os resultados podem ser descobertos mais facilmente em uma visualização controlada do objeto real. A visualização científica prover facilidades para converter dados em uma representação gráfica significativa, com o objetivo de para criar representações visuais expressivas e efetivas a partir de dados científicos.

### 3.1.5 Caracterizando os Dados de Visualização

Uma vez que pretendemos visualizar dados, claramente precisamos conhecer as principais características dos mesmos. Este conhecimento vai nos ajudar a criar modelos de dados úteis e sistemas de visualização poderosos. Sem um claro entendimento dos dados, pode-se projetar sistemas de visualização limitados e inflexíveis.

Primeiramente, os dados de visualização são *discretos*. Isto é porque usamos computadores digitais para obter, analisar, e representar nossos dados, e tipicamente medimos ou amostramos informação num número finito de pontos. Conseqüentemente, toda informação é necessariamente representada na forma discreta.

A segunda característica importante dos dados da visualização é a sua estrutura, que pode ser *regular* ou *irregular* (alternativamente, fala-se de dados *estruturados* ou *não-estruturados*).

Os dados regulares estão definidos sobre grades especificadas por regras simples, sendo o exemplo clássico o de uma imagem real que pode ser definida como uma função  $f: [0, \dots, m - 1] \times [0, \dots, n - 1] \rightarrow \mathbb{R}$  definida sobre uma grade discreta que é um produto cartesiano de intervalos de  $\mathbb{N}$ . Se fizermos uma amostragem em pontos igualmente espaçados em uma reta, não precisamos armazenar todos os pontos, somente a posição inicial do intervalo, o espaçamento entre os pontos, e o número total de pontos. A posição dos pontos é então conhecida implicitamente, e podemos tirar vantagem desse conhecimento para economizar memória.

Dados que não são regulares são irregulares. A vantagem dos dados irregulares é que podemos representar informação mais densamente onde ela muda rapidamente e menos densamente onde a mudança não é tão grande. Assim, dados irregulares nos permitem criar formas de representação adaptativas, que podem ser importantes devido aos recursos computacionais limitados.

Caracterizando os dados como regulares ou irregulares podemos fazer suposições interessantes sobre os mesmos. Como vimos anteriormente, podemos armazenar dados regulares de forma mais compacta. Tipicamente, podemos também fazer cálculos mais eficientemente com dados regulares do que

com dados irregulares. Por outro lado, dados irregulares nos dão mais liberdade na representação dos dados e pode representar que não tem padrões regulares. Finalmente, os dados têm uma dimensão topológica arbitrária.

A dimensão dos dados é importante porque ela implica em métodos apropriados para visualização e representação de dados. Por exemplo, em 1D usamos naturalmente em gráficos x-y, gráficos de barras, ou gráficos de pizza, e armazenamos como uma lista unidimensional de valores. Para dados 2D devemos armazenar os dados numa matriz, e visualizá-los com um gráfico de superfície deformado. Uma caracterização detalhada dos dados de visualização está descrita em (SCHROEDER; KENMARTIN; LORENSEN, 1998).

### **3.1.6 Técnicas de Visualização**

Uma técnica de visualização é usada para criar e manipular uma representação gráfica a partir de um conjunto de dados. Algumas técnicas são apropriadas somente para aplicações específicas enquanto outras são mais genéricas e pode ser usada em muitas aplicações. Esta seção foca em técnicas genéricas. Devemos ter sempre em mente que o objetivo da visualização não é entender os dados, mas entender os fenômenos subjacentes.

#### **3.1.6.1 Classificação das Técnicas de Visualização**

As classificações de técnicas de visualização são freqüentemente baseadas na dimensão do domínio da quantidade a ser visualizada, que é o número de variáveis independentes do domínio no qual a quantidade atua, e no tipo de quantidade, ou seja, escalar, vetorial, ou tensorial.

Como a maioria das técnicas de visualização apresentam algoritmos as dimensões 1, 2, 3 ou mais, assim podemos fazer uma classificação independentes da dimensão do domínio da quantidade, e focar apenas no o tipo da quantidade a ser visualizada.

#### **3.1.6.2 Técnicas de Visualização de Campos Escalares**

Existem muitas técnicas diferentes para visualizar dados escalares, pois são encontrados comumente em aplicações do mundo real, e porque dados es-

calares são fáceis de trabalhar. Segue abaixo a descrição de algumas técnicas de visualização de campos escalares:

**Mapeamento de Cores** é uma técnica comum de visualização de quantias escalares, que associa dados escalares a cores, e exibe as cores no sistema computacional. A associação é implementada pela indexação em um tabela de cores. Os valores escalares servem como índices da tabela.

**Contouring** é uma extensão natural do mapeamento de cores. Quando vemos uma superfície colorida com dados, os olhos freqüentemente separam áreas coloridas similarmente em regiões distintas. Quando contornamos dados, nós estamos efetivamente construindo uma fronteira entre estas regiões. Estas fronteiras correspondem a linhas de contorno (2D) ou superfícies (3D) de valor escalar constante.

Exemplos de contornos bidimensionais, também conhecido como *iso-lines*, incluem mapas de temperatura com linhas de temperatura constante (isotermas), ou mapas topológicos desenhados com linhas de elevação constante. Contornos tridimensionais são chamados *isosurfaces*, e podem ser aproximadas por muitas primitivas poligonais. Exemplos de *isosurfaces* incluem imagens médicas de intensidade constante correspondendo a tecidos do corpo humano tais como a pele, os ossos, ou outros órgãos. Outras *isosurfaces* abstratas tais como superfícies de pressão ou temperatura constante também podem ser criadas em dinâmica dos fluidos.

### 3.1.6.3 Técnicas de Visualização de Campos Vetoriais

Dados vetoriais são geralmente representações bidimensionais ou tridimensionais de direção e magnitude.

**Glyph** é uma técnica natural de visualização de vetores é desenhar uma linha orientada e cuja escala é ajustada para cada vetor. A linha inicia no ponto com o qual o vetor está associado e é orientada na direção do vetor. Tipicamente, a linha resultante deve ter tamanho proporcional à norma do vetor. Esta técnica é freqüentemente referida como sendo *hedgehog*.

Existem muitas variações desta técnica. Setas podem ser acrescentadas para indicar a direção da linha. As linhas podem ser coloridas de acordo

com a magnitude do vetor, ou qualquer outra quantidade escalar (ex. pressão ou temperatura). Também, ao invés de usar uma linha, “*glyphs*” orientados podem ser usados. Por *glyph* queremos dizer qualquer representação geométrica tais como um triângulo orientado ou um cone.

**Streamline** é uma linha tangente a direção da velocidade instantânea. Para visualizar isto em um fluxo, podemos imaginar o movimento de um pequeno elemento de fluido marcado. Por exemplo, poderíamos marcar uma gota de água com tintura fluorescente e iluminá-la usando um laser de modo que apresentasse fluorescência.

#### 3.1.6.4 Técnicas de Visualização de Campos Tensoriais

Os tensores são generalizações complexas de vetores e matrizes. Um tensor de *rank*  $k$  pode ser considerado uma tabela  $k$ -dimensional. Um tensor de rank 0 é um escalar, rank 1 é um vetor, rank 2 é uma matriz, e um tensor de rank 3 é um array retangular tridimensional. Tensores de mais alto rank são arrays retangulares  $k$ -dimensionais. Uma técnica de visualização para dados tensoriais é conhecida por *Tensor Ellipsoids* e uma descrição detalhada dessa técnica está em (SCHROEDER; KENMARTIN; LORENSEN, 1998).

#### 3.1.7 Realidade Virtual

Realidade Virtual é uma forma mais avançada de interface, que no âmbito computacional permite visualizar, manipular e explorar (ou interagir com) as informações e dados complexos em tempo real. Sistemas de realidade virtual permitem aproveitar o conhecimento intuitivo do usuário sobre a navegação no espaço tridimensional, para melhor entender o sistema.

Algumas características desejáveis da realidade virtual são:

**Imersão** é a sensação de estar dentro do ambiente. A idéia de imersão está ligada com o sentimento de se estar dentro do ambiente. Normalmente, um sistema imersivo é obtido com o uso de capacete de visualização, mas existem também sistemas imersivos baseados em salas com projeções das visões nas paredes, teto, e piso (FRERY; KELNER, 2003).

**Interação** é a possibilidade do usuário interferir com o que acontece no ambiente, e vice-versa. A idéia de interação está ligada com a capacidade do computador detectar as entradas do usuário e modificar instantaneamente o mundo virtual e as ações sobre ele (capacidade reativa). As pessoas gostam de ficar cativadas por uma boa simulação e de ver as cenas mudarem em resposta aos seus comandos. Esta é a característica mais marcante nos video-games.

**Envolvimento** é a capacidade do ambiente motivar o usuário a participar. A idéia de envolvimento, por sua vez, está ligada com o grau de motivação para o engajamento de uma pessoa com determinada atividade. O envolvimento pode ser passivo, como ler um livro ou assistir televisão, ou ativo, ao participar de um jogo com algum parceiro. A realidade virtual tem potencial para os dois tipos de envolvimento ao permitir a exploração de um ambiente virtual e ao propiciar a interação do usuário com um mundo virtual dinâmico.

## 3.2 Classificação das Ferramentas para Visualização

Existe uma grande quantidade de ferramentas que podem ser usadas para gerar imagens a partir de um conjunto de dados. Estas podem ser categorizadas pelo nível de abstração que elas provêm do hardware em que as imagens são exibidas, como segue:

**Bibliotecas Gráficas** consistem de *frameworks* de programação que provêm uma interface direta entre a aplicação e o sistema operacional/*device driver*/hardware gráfico do sistema em que a aplicação é executada. Elas, portanto, garantem que a aplicação pode ser executada em qualquer plataforma suportada pela biblioteca. As bibliotecas gráficas todavia provêm apenas um nível relativamente baixo de abstração, requerendo um grande esforço de programação para gerar aplicações capazes de produzir uma visualização específica. Estas bibliotecas são usadas para construir ferramentas que proporcionam abstrações de mais alto nível. Um exemplo de tal biblioteca de baixo nível, muito usada, é OpenGL (OPENGL, 2002).

**Kits de Ferramentas de Renderização** são tipicamente baseadas em bibliotecas gráficas tal como OpenGL, mas provêm uma abstração adicional de alto nível, oferecendo um conjunto rico de componentes de software que podem ser combinadas para produzir o resultado esperado. O esforço de programação requerido é, portanto, reduzido em comparação com as bibliotecas gráficas, mas estes kits de ferramentas continuam sendo ambientes de programação baseados em código, principalmente para uso de programadores de aplicação. Um exemplo de tal kit de ferramentas de renderização, muito usado, é Open Inventor (SILICON GRAPHICS, 2002).

**Sistemas de Visualização** estendem as capacidades dos kits de ferramentas de renderização provendo um conjunto de unidades funcionais que podem ser configuradas numa interface gráfica para produzir programas de visualização e de processamento de dados sem necessidade de programação direta. Modelos, grafos e mapeamentos sofisticados e altamente configuráveis podem ser produzidos a partir de conjunto de dados generalizados usando somente as facilidades providas pela interface com o usuário. OpenDX (IBM, 2002) e AVS (AVS, 2002) são exemplos de sistemas de visualização muito usados pela comunidade de visualização.

Muitas das ferramentas disponíveis em cada uma destas três categorias seriam capazes de prover as funcionalidades requeridas por este projeto. O uso de biblioteca gráfica ou kit de ferramentas de renderização iria todavia requerer muita programação para prover as funcionalidades de interface com o usuário, providas por sistemas de visualização e assim foi decidido basear este projeto no uso de um sistema de visualização adequado.

Na seção 3.3 vamos descrever em detalhes alguns dos principais conceitos dos sistemas de visualização.

### **3.3 Sistemas de Visualização**

Os sistemas atualmente disponíveis de visualização numérica comerciais abertos podem ser usados para visualizar dados resultantes de análises utilizando o método do elemento finito. Estes sistemas se diferenciam entre si pelas funcionalidades oferecidas, pela complexidade de desenvolvimento re-

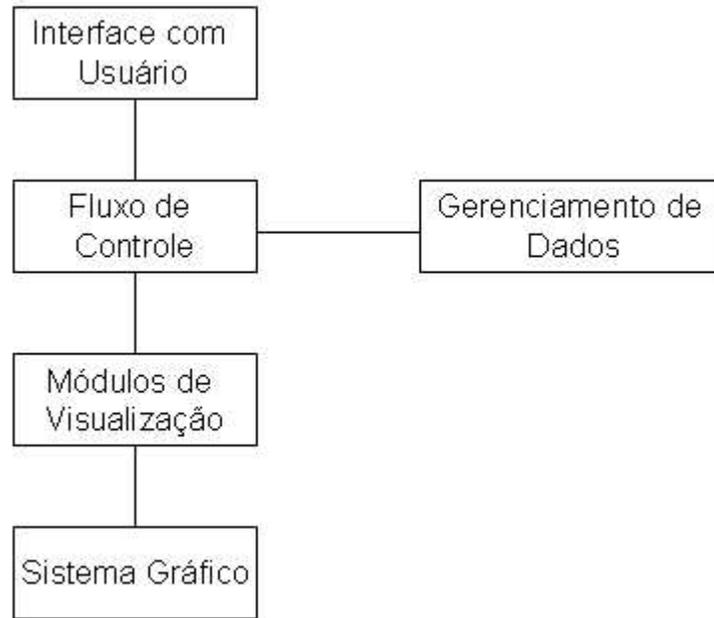


Figura 2: Arquitetura de um sistema de visualização genérico

querido e pela flexibilidade de uso. Este capítulo discute os principais aspectos dos sistemas de visualização que serão empregados para a sua comparação.

### 3.3.1 Arquitetura dos Sistemas de Visualização

A arquitetura de um sistema de visualização descreve os componentes do sistema e os relacionamentos entre os componentes. A arquitetura genérica de um sistema de visualização, segundo proposta de (GALLAGHER, 1995) está representada na Figura 2. Cada um dos componentes será descrito nas próximas seções.

### 3.3.2 Interface com o Usuário

A Interface com usuário de um sistema de visualização é um componente que define os mecanismos pelos quais o usuário interage com o sistema; especificando ações para serem executadas, parâmetros de entrada para módulos, modificação de atributos, entre outras interações. A interface com o usuário é extremamente importante, pois ela é a visão do sistema de visualização apresentada para o usuário. Vamos descrever dois tipos de interfaces com usuário: de linha de comando e gráfica.

### 3.3.2.1 Interface de Linguagem de Comandos

Prover uma interface textual para o sistema de visualização. Se a linguagem é concisa, o usuário é experiente, e digita rápido, este tipo de interface pode ser um método de interação muito eficiente. Embora fácil de implementar, este tipo de interface é geralmente inadequado por causa do tempo necessário para um novo usuário aprender a linguagem de comandos e os erros gerados ao escrever comandos. Uma linguagem de comandos é útil, principalmente quando usada em conjunto com uma interface gráfica, onde existe um relacionamento um-para-um entre os comandos e a interface gráfica. Quando uma ação é executada usando a interface gráfica um comando “equivalente” à ação pode ser gerado. Tal criação automática de comandos é conhecida como *scripting*. Um exemplo de script é mostrado abaixo:

```
data = Import("watermolecule");
isosurface = Isosurface(data);
camera = AutoCamera(isosurface,width=10);
Display(isosurface,camera);
// We scale the object by a factor of 2 in the y direction,
// and translate by 1 in each dimension
matrix = [[1 0 0][0 2 0][0 0 1][1 1 1]];
isosurface = Transform(isosurface, matrix);
Display(isosurface,camera);
```

### 3.3.2.2 Interfaces Gráficas

Uma interface gráfica clássica deve prover uma interface visual para o sistema de visualização com mecanismos de interação através de, pelo menos, um mouse. O usuário interage com o sistema através da representação gráfica dos objetos, tais como botões, barras de rolagem, menus e ícones. As interfaces gráficas são mais fáceis de aprender e menos suscetíveis a erros do que interfaces de linguagem de comandos. O usuário não precisa lembrar dos nomes nem da sintaxe dos comandos, pois as operações disponíveis são exibidas (SHNEIDERMAN, 1998).

### **3.3.3 Gerenciamento de Dados**

O componente de Gerenciamento de Dados inclui os mecanismos para importar dados de fontes externas para o sistema de visualização e para manipular os dados internamente. Ele inclui o acesso aos resultados da análise e o gerenciamento da memória interna.

#### **3.3.3.1 Modelo de Dados**

O modelo de dados para um sistema de visualização é a organização interna dos campos nos quais estão organizadas as informações de análise. Este modelo deve descrever a malha de análise, as condições iniciais e de contorno, bem como os resultados da análise. O projeto do modelo de dados é extremamente importante para a usabilidade de um sistema de visualização. O projeto deve refletir muito bem o tipo e a estrutura dos dados usados em análises. Os fatores diretrizes do projeto são eficiência de armazenamento e suporte completo para análise.

#### **3.3.3.2 Gerenciamento de Dados Internos**

Por gerenciamento de dados internos estaremos nos referindo aos mecanismos utilizados para tratar as estruturas de dados usadas pelo sistema de visualização. A eficiência das ferramentas de gerenciamento de memória pode variar de plataforma para plataforma. Por esta razão, pode ser mais eficiente gerenciar objetos específicos do sistema de visualização. Um objeto é um estrutura de dados que armazena um conjunto particular de informações, tal como uma malha de elementos finitos. Ao invés de somente liberar o espaço usado por um objeto o sistema de gerenciamento de dados pode guardar uma lista de objetos liberados para serem reusados. O projetista deve conhecer o comportamento do uso dos dados por um sistema de visualização para poder manipular os dados mais eficientemente do que as ferramentas genéricas de gerenciamento de memória como, por exemplo, a função `malloc` da biblioteca padrão C.

### 3.3.3.3 Redução de Dados

A visualização de grande quantidade de dados em computadores pessoais pode ser às vezes quase impossível, mesmo em máquinas com grande quantidade de memória (RAM e Vídeo), e grande poder de processamento (gráfico ou não). Por 'grande' queremos dizer que a quantidade de para ser visualizado é muito maior do que a capacidade (em termos de memória e de processamento) da máquina na qual a visualização está sendo executada. 'Grande' para uma máquina fraca podem ser 20000 elementos, para uma máquina de última geração pode ser 500000 elementos. Alguns problemas de dinâmica dos fluidos computacional têm malhas com mais de um milhão de elementos e várias centenas de passos no tempo, ou seja, centenas de milhões de elementos para serem visualizados em poucos segundos. Visualizar tais malhas mesmo em máquinas muito bem dotadas pode ser difícil.

A redução da demanda computacional para uma visualização pode ser alcançada pela definição subconjuntos de elementos pela especificação da área de interesse na malha, tais como uma caixa. Somente os elementos dentro da caixa serão usados na computação da visualização. Para reduzir a quantidade de dados usados pelo sistema de visualização de dados para somente os elementos dentro da caixa pode ser empregado um banco de dados. Por exemplo, o usuário pode selecionar uma asa de avião, extrair os dados da área e visualizá-los.

Um outro método de redução da quantidade de dados necessária para produzir uma visualização, chamado *extracts*, é descrito por Globus (GLOBUS, 1992). Um *extract* é um sub-conjunto de um conjunto de dados intermediários entre um objeto gráfico (ex. um polígono) e o resultado. O *extract* armazena a geometria de um objeto gráfico com os resultados para os vértices da geometria, possivelmente para vários passos no tempo. Por exemplo, se quiséssemos visualizar um *slice* de um campo de pressão de uma malha de elementos finitos em 100 passos no tempo, poderíamos criar um conjunto de *extracts* que contem a geometria dos polígonos que formam o *slice* e a pressão em cada vértice dos polígonos a cada passo no tempo. Os *slices* poderiam ser animados no tempo por somente um mapeamento de pressões no passo do tempo apropriado para cores, e a renderização dos polígonos do *slice*. Somente uma fração do armazenamento requerido para todo o conjunto de dados

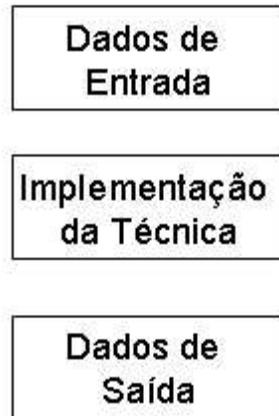


Figura 3: Modelo de módulo de visualização

é usado pelo *extracts*.

### 3.3.4 Módulos de Visualização

Um módulo de visualização é um conjunto de componentes de software que implementam uma técnica específica de visualização. Exemplos disto são contornar uma superfície ou extrair uma superfície de mesma propriedade (*isosurface*) de um volume. Os componentes de um módulo são mostrados na Figura 3. O exemplo mais simples de um módulo é uma função C. As entradas são os argumentos da função, a implementação da técnica de visualização é representada pelo executável da função, e os dados de saída são outros argumentos da função.

### 3.3.5 Fluxo de Controle

O componente de Fluxo de Controle define como os módulos de visualização são executados e como os dados fluem para os módulos no sistema. A implementação de Fluxo de Controle é uma das principais características de sistemas de visualização. Vamos classificar em três tipos: sistemas *Turnkey*, construtores de aplicação usando interface de programação visual e construtores de aplicação utilizando uma interface de linguagem de programação ou *script*.

### 3.3.5.1 Sistemas Turnkey

É sistema de visualização para o usuário final, que prover a interface e os módulos que permitem ao usuário executar visualizações sem qualquer esforço de programação. Estes sistemas são geralmente fáceis de usar, e permitem a usuários novatos visualizar dados rapidamente. Alguns sistemas são adaptados para uma aplicação em particular, tal como mecânica dos fluidos, com interfaces projetadas com esse propósito.

Sistemas *turnkey* podem ser classificados como sistemas que provêm várias instancias de módulos ou nos quais não provêm. Na maioria dos sistemas o fluxo de controle prover apenas uma instancia de um dado módulo. Assim o usuário fica limitado a um conjunto de dados por técnica de visualização (por exemplo, extrair uma iso-superfície de um campo escalar).

Um sistema com um fluxo de controle que prove várias instancias de um dado módulo significa que um módulo pode executar com qualquer número de parâmetros de entrada, produzindo resultados para cada uma das entradas. O usuário pode instanciar várias técnicas de visualização, cada um com um conjunto de dados diferentes. Assim, por exemplo, você pode gerar iso-superfícies de vários campos escalares ao mesmo tempo. Esta arquitetura é mais difícil de projetar e implementar pois o estado dos módulos devem ser guardados e recuperados em cada passo no tempo. Um exemplo deste tipo de sistema é o *Wavefront's Data Visualizer* (BRITAIN, 1995).

Como as funcionalidades dos sistemas *turnkey* geralmente não podem ser modificadas pelo usuário, o projeto de tal sistema deve prover funcionalidades suficientes para satisfazer os requerimentos da maioria dos usuários. Esta limitação é vista como uma desvantagem pois é impossível satisfazer os requisitos de todos usuários. Por outro lado, a facilidade de uso e a habilidade de customizar tais sistemas para uma disciplina em particular pode contrabalançar as desvantagens.

### 3.3.5.2 Construtores de Aplicação utilizando Interface de Programação Visual

Um construtor de sistemas de visualização prove ao usuário um conjunto de ferramentas para interativamente construir uma aplicação de visualização.

Um conjunto básico de módulos de visualização, é geralmente provido, para suportar escrita de módulos customizados.

Podemos classificar os construtores de aplicação pela interface de programação utilizada para construir aplicações: Interface de programação visual ou interface de linguagem. A maioria dos construtores de aplicação disponível atualmente, usa interface de programação visual. Alguns desses sistemas são: AVS (AVS, 2002), Iris Explorer (GROUP, 2002) e OpenDX (IBM, 2002). A interface de programação visual permite ao usuário construir interativamente uma aplicação interagindo com uma representação bidimensional dos módulos de visualização.

Usando uma ferramenta interativa para construir um grafo direcionado, o usuário pode selecionar um conjunto de módulos de uma paleta, conectar os módulos, e criar aplicações de visualização complexas. Módulos customizados para executar tarefas particulares para uma aplicação podem ser escritos e adicionados a paleta. Os nós do grafo são os módulos de visualização, as arestas são caminhos pelos quais os dados podem passar de um módulo para outro, como mostrado no exemplo da Figura 4.

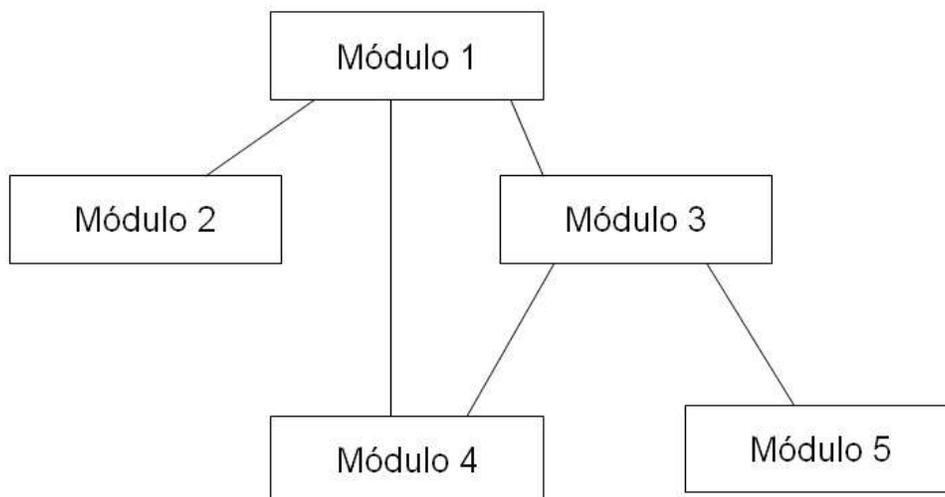


Figura 4: Em uma aplicação de visualização modular, fluxos de dados específicos podem ser definidos entre módulos

Construtores de aplicações são complexos de serem implementados. A complexidade vem do fato que cada módulo pode executar como um processo separado (nos termos de processo UNIX), e então o sistema de visualização deve suportar alguma forma de comunicação entre processos. Se um módulo é

um processo separado, então um usuário pode escrever, compilar e link-editar somente os módulos que ele está desenvolvendo sem ter que re-compilar toda a aplicação de visualização. Módulos podem também executar em máquinas remotas em ambientes computacionais heterogêneos. Existem vários métodos que podem ser usados para transmitir dados entre processos. Um método usado por vários sistemas de visualização é a utilização de sockets para fazer a comunicação entre os processos. O grande problema no uso de sockets é que os dados ficam duplicados, ou seja, tanto quem envia quanto quem recebe fica com uma cópia dos dados. É possível superar esse problema usando as facilidades de memória compartilhada dos sistemas operacionais (ex. UNIX), essas facilidades permitem a dois ou mais processos compartilharem um mesmo segmento de memória.

Construtores de aplicação podem ser um excelente caminho para prototipar uma aplicação de visualização. Eles são mais difíceis de usar do que os sistemas *turnkey* porque eles requerem mais estudo pelo usuário, que deve adquirir dois tipos de conhecimento: como construir uma aplicação e como escrever um módulo customizado, se necessário. Estes requerem muito mais tempo e recursos para usar efetivamente, e também não são adaptados para qualquer disciplina, mas são sistemas de visualização gerais.

### **3.3.5.3 Construtores de Aplicação utilizando Interface de Linguagem**

Um construtor de aplicação pode também usar interface baseada em linguagem para construir aplicações. Neste caso, o usuário constrói uma aplicação de visualização escrevendo um programa em um ambiente que suporta uma linguagem interpretada. Um ambiente de linguagem interpretada permite que programas possam ser escritos, testados, e depurados interativamente. Se a linguagem interpretada é uma linguagem de programação de propósito geral, que é, uma linguagem em que componentes computacionais podem ser escritos, então o construtor de aplicação vai ser muito mais flexível e extensível. Um exemplo de tal sistema é o *SuperGlue* (HULTQUIST; RAIBLE, 1992). *SuperGlue* está centrado em torno de um dialeto de Lisp chamado *Scheme*. Controle de fluxo de alto nível e programação de interface com usuário é feita em *Scheme* enquanto funções de computação intensiva funções gráficas são escritas numa linguagem compilada.

### 3.3.6 Sistema Gráfico

O componente gráfico inclui os mecanismos que exibem e manipulam as primitivas geométricas resultantes da execução de um módulo de visualização. Existem quatro componentes principais de sistema gráfico: Gerente de Dados Gráficos, view manager, user interface e bibliotecas de programação gráficas.

#### 3.3.6.1 Gerente de Base de Dados Gráficos

Quando um conjunto de primitivas geométricas, tais como um conjunto de polígonos representando uma *isosurface* dentro de um volume de elementos finitos, é criada pelo módulo de visualização, eles devem ser armazenados para serem exibidos na tela do computador. O Gerente de Base de Dados Gráficos é usado para organizar e manipular o armazenamento de objetos gráficos, tais como primitivas geométricas e seus atributos (cor, visibilidade, etc.). O Gerente de Base de Dados Gráficos também executa as operações de busca na base de dados, tais como localização de uma primitiva geométrica mais próxima de um dado ponto e a determinação de objetos dentro de uma dada área (para atualização seletiva da tela).

#### 3.3.6.2 Gerente de Visualização

O gerente de visualização determina onde e quando objetos devem ser exibidos.

## 3.4 *Data Explorer*

*Open Visualization Data Explorer* é um ambiente completo de visualização que oferece aos usuários a capacidade de aplicar técnicas avançadas de visualização e análise para os seus dados. Estas técnicas podem ser aplicadas para ajudar aos usuários a ganhar novos discernimentos dos dados de aplicações em uma grande variedade de áreas do conhecimento incluindo ciência, engenharia, medicina e negócios.

O *Data Explorer*, como é mais conhecido, prover um conjunto completo de ferramentas para manipular, transformar, processar, renderizar e animar

dados e para permitir que métodos de visualização e análise baseados, em qualquer combinação, de pontos, linhas, áreas, volumes, imagens ou primitivas geométricas. *Data Explorer* foi desenvolvido inicialmente nos centros de pesquisa da IBM, como um produto IBM. A IBM liberou o *Data Explorer* para o mundo do código aberto, e está participando ativamente na continuação do desenvolvimento do mesmo.

Na seção 5.2 vamos descrever a arquitetura do *Data Explorer*, na seção 3.4.2 será descrito o modelo de dados, na seção 3.4.3 o modelo de execução, na seção 3.4.4 os módulos e na seção 3.4.5 como estender as funcionalidade oferecidas pelo *Data Explorer*.

### 3.4.1 Arquitetura

*Data Explorer* foi projetado utilizando o modelo de arquitetura (também chamado de padrão arquitetural) cliente-servidor (BUSHMANN, 2001) incorporando componentes de sistema como TCP/IP, Sistema de janelas X (*X Window System*) e Motif.

Nesse modelo cliente servidor, a interface com o usuário é o cliente. Os módulos, e componentes de gerenciamento de dados, freqüentemente referenciados coletivamente como *executive*, formam o lado do servidor. A interface com o usuário cliente pode estar numa plataforma diferente da do servidor (*executive*), e o *executive* em plataformas diferentes simultaneamente. *Data Explorer* permite que o usuário comute entre os servidores rodando em plataformas de hardware diferente.

O *Data Explorer* foi projetado utilizando também o padrão arquitetural de camadas, como descrito em (BUSHMANN, 2001), sendo constituído de quatro camadas cada uma com a sua própria interface bem definida. Estas camadas são descritas na ordem que o usuário provavelmente as encontra:

**Interface gráfica com o usuário** foi construída utilizando os padrões *X Window* e *Motif*. Estas ferramentas organizam múltiplas janelas de aplicação que permitem ao usuário criar e controlar o processo de visualização facilmente e efetivamente. A interface com o usuário prover dois níveis de serviço. O primeiro nível para não programadores ou usuários com requisitos limitados que podem executar programas visuais previamente cria-

dos. O segundo nível para programadores que podem criar visualizações customizadas pelo uso da interface para conectar módulos de maneira flexível, e para criar novas combinações de módulos na forma de macros. A interface gráfica com o usuário do *Data Explorer* permite que o usuário crie ou modifique um programa visual para facilmente realizar amostragem, seleção, e transformação de dados durante a visualização. O usuário pode usar o *Visual Program Editor* (VPE) para criar novos cenários pela simples conexão de ícones de módulos na tela em alguma seqüência lógica. Segue uma breve descrição das janelas providas pelo *Data Explorer*, uma descrição detalhada dessas janelas está em (EXPLORER, 1997b):

**Visual Program Editor** permite ao usuário, criar e alterar programas visuais.

**Control Panel** permite ao usuário, controlar os valores dos parâmetros de entrada das ferramentas usadas num programa visual.

**Image Window** exibe uma imagem criada por um programa visual e permite interação direta com a imagem visualizada. A visualização de um objeto na janela *Image* pode ser modificada usando a opção “*View Control...*” do menu *Options*. A Figura 5 mostra as opções de controles da caixa de diálogo “*View Control...*”. As opções de controle são:

**Modo de Visualização** apresenta as opções de interação com a Imagem. Segue abaixo os modos de interação disponíveis: *Camera*, *Cursors*, *Pick*, *Pan/Zoom*, *Roam*, *Rotate*, *Zoom* e *Navigate*. O modo de visualização *Navigate*, permite que o usuário modifique a posição da câmera em todas as direções, como também é capaz de modificar a direção de visão. Este modo de visualização também permite ao usuário mover-se através de um objeto e vê-lo de dentro para fora. Quando o modo *Navigate* é usado a câmera é automaticamente modificada para a projeção perspectiva.

**Direção de Visão** As seguintes opções de direção são providas: None, Top, Bottom, Front, Back, Left, Right, Diagonal, Off Top, Off Bottom, Off Front, Off Back, Off Left, Off Right, Off Diagonal. Dado que uma visão *head-on* de um objeto tende a diminuir a quali-



Figura 5: Caixa de dialogo do Controle de Visualização

dade da imagem, são fornecidas opções *head-off* para enviesar ligeiramente a imagem.

**Projeção** O Data Explorer prover dois métodos de projeção geométricas. Estes métodos tornam possível mapear uma imagem tridimensional numa tela bidimensional. Os dois métodos são:

**Perspectiva** A projeção perspectiva sumula o sistema visual humano e o sistema da câmera fotográfica, resultando assim na renderização realista dos objetos.

**Ortográfica** Projeção ortográfica prover uma visão menos realista de um objeto do a projeção perspectiva. Contudo, a projeção ortográfica preserva a distância entre os objetos e as retas paralelas.

**Angulo de Visão** Se for utilizada a projeção perspectiva, é possível especificar também o ângulo da visão (em graus). O vértice do ângulo da visão fica situado na câmera, e os pontos de extremidade são os lados esquerdos e direitos da área da imagem. Assim, quanto maior o ângulo da visão, maior a área da imagem que pode caber na área da visão.

**Help** prover acesso ao manual do usuário e acesso a informações de

ajuda baseada em contexto.

**Executive** é o componente do *Data Explorer* que gerencia a execução dos módulos especificados num *script*. O *script* é gerado pela interface gráfica com o usuário para invocar funções de visualização em programas visuais. Os usuários podem também usar a linguagem de *script* para escrever seus próprios programas.

**Módulos** O *Data Explorer* prover um conjunto extensível de módulos de visualização interoperáveis. Os módulos usados por funções de visualização estão disponíveis:

- Como nós, através do uso em programas visuais.
- Como chamadas de funções, disponíveis da interface na linguagem de *script* e providas pela camada *executive*.
- Para aplicações integradas, como parte da interface da biblioteca de programação visual.

**Gerenciamento de dados** A camada de gerenciamento de dados é a porção da interface de programação que prover módulos com acesso ao modelo de dados, que será descrito na seção 3.4.2. Esta camada inclui serviços gerais de sistema como também rotinas para criação e gerenciamento do conjunto de dados. A camada de gerenciamento de dados também prover uma interface de programação de aplicações (API) para inclusão de novos módulos para o *Data Explorer* e para acessar a potência e flexibilidade do modelo de dados.

### 3.4.2 Modelo de Dados

A representação de dados é um problema significativo num sistema de visualização, por duas razões. Primeiro, cientistas têm uma ampla necessidade de representação de dados que possam ser importados para o sistema. Segundo, para facilitar a interoperabilidade dos módulos, que é a possibilidade dos módulos se conectarem de uma grande variedade de maneiras, para isso é importante que eles se comuniquem utilizando uma linguagem comum.

Assim o *Data Explorer* prover um modelo de dados comum para importar dados para o sistema e para uso na comunicação entre os módulos. O modelo

de dados concreto usado pelo *Data Explorer* é baseado num modelo abstrato de dados apresentado em Collons, Haber e Lucas (1992). O modelo de dados abstrato tem duas representações concretas no *Data Explorer*: Na primeira, os dados são passados entre módulos na forma de ponteiros para objetos em memória compartilhada. Segundo, os dados podem ser armazenados externamente em arquivos que incorporam o modelo de dados.

**Fields** Os objetos *Field* são os objetos fundamentais no modelo de dados do *Data Explorer*. Um *field* representa um mapeamento de algum domínio para algum espaço de dados. O domínio do mapeamento é especificado por um conjunto de posições e (geralmente) um conjunto de conexões que permitem a interpolação dos valores de dados para pontos pertencentes ao domínio. O mapeamento para todos os pontos no domínio é representado implicitamente especificando os dados que são dependentes dos pontos amostrados ou das conexões entre os pontos amostrados.

Esta simples abstração é suficiente para representar uma ampla variedade de dados. Por exemplo, é possível descrever dados tridimensionais volumétricos, cujo domínio é a região especificada pelas posições e cujo espaço de dados é o valor associado com cada posição. Superfícies bidimensionais imersas num espaço tridimensional podem ter um domínio que é o conjunto de posições na superfície, e um espaço de dados que é, por exemplo, o conjunto de valores na superfície.

Se os dados são dependentes das posições dadas, então o valor de qualquer ponto diferente dos que foram dados como entrada pode ser calculado utilizando interpolação dentro da conexão em que o ponto pertence. Se os dados são dependentes das conexões, então o valor dos dados é assumido ser constante dentro de cada conexão. Se não forem especificadas conexões, então não existe nenhuma informação implícita sobre o valor dos dados em posições diferentes das dadas. A informação num *Field* é representada por alguns componentes conhecidos. Cada componente tem um valor, que é um *Object*. Em geral, componentes são *Array Objects* (EXPLORER, 1997b). Por exemplo, o componente “*positions*” é um Array especificando o conjunto de pontos amostrados; o componente “*connections*” é um Array especificando um meio para interpolar entre os valores; e o componente “*data*” é um Array especificando um conjunto de

valores.

Um *Field* pode também conter vários outros componentes adicionalmente, tais como *colors*, *opacities*, *tangents*, *normals*, e *data statistics*, estes componentes podem ser usados na *renderização* de um *field* ou para geração de transformações adicionais e estão descritos em Explorer (1997b). Os *Fields* podem ser agrupados em *Groups* que podem ser reunidos com outros *fields* e *groups*. Provendo a habilidade de compartilhar componentes entre *fields* e usando esse mecanismo de agregação é possível facilmente e compactamente representar tanto dados multivariados sobre o mesmo espaço quanto dados univariados em vários espaços.

**Compartilhamento de dados** Todos os dados que são passados entre módulos do sistema é passado na forma ponteiros para objetos. Um objeto é uma região da memória global que contém um identificador de tipo, informações independentes do tipo como um contador de referencias (reference count), e informações dependentes do tipo. O acesso aos objetos é realizado através de um conjunto rotinas de acesso para ocultar a informação. A eficiência é garantida com o encapsulamento os dados em *array objects*, que consiste de um cabeçário de objeto (contendo contador de referência e informação de tipo) que referencia um simples array de dados. Assim, para operar nos dados, o módulo chama uma rotina para obter um ponteiro para os dados e depois disso o acesso aos dados é tão eficiente quanto um simples array.

Uma consequência importante desse modelo de dados é que o compartilhamento de dados entre objetos é possível e na verdade encorajado. Isso é ilustrado na Figura 6. Esta figura ilustra dois *fields* que compartilham posições tridimensionais e conexões tetraédricas, mas cada um dos quais tem dados separados. Por exemplo, este compartilhamento permite aos membros de uma serie no tempo (representado por um grupo), definida numa grade fixa e representada por dois *fields*, compartilhar as posições e conexões enquanto cada uma tem dados diferentes. Além do que, compartilhamento é vital para uma implementação eficiente do modelo de programação baseado no fluxo de dados, em que um módulo não pode modificar os dados de entrada. No exemplo da Figura 6, o primeiro *Field* pode representar a entrada para um módulo (ex. um campo vetorial), enquanto o segundo campo pode representar a saída de um módulo que

computa o comprimento de cada vetor. O módulo construiu um *Field* com um componente de dados separado representando o resultado calculado, mas não teve que copiar os componentes do *Field* que permaneceram a mesma (posições e conexões), porque eles podem ser compartilhados entre os campos de entrada e saída.

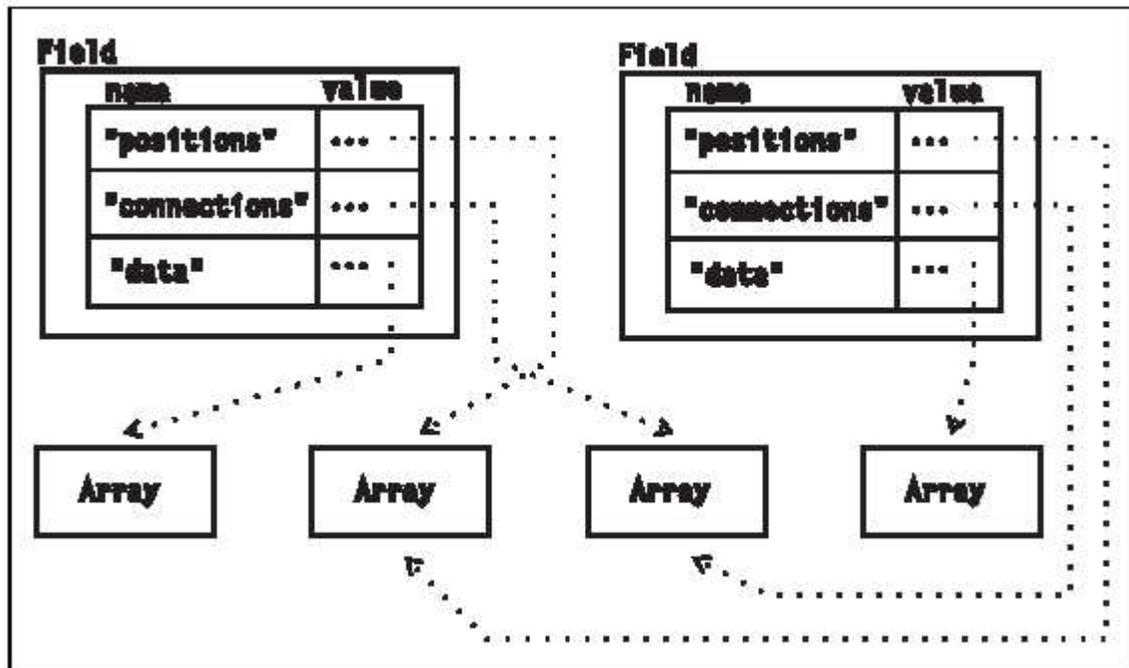


Figura 6: Componentes Compartilhados entre Campos Diferentes

**Paralelismo** Os dados são preparados para paralelização através de particionamento explícito. O processo de particionamento divide um *field* de entrada num número de *fields* auto-contidos, que são reunidos num *field* composto. Cada elemento de interpolação do campo de entrada é atribuído para um e somente um dos campos de saída. Fazendo isso, a união dos campos resultantes cobre exatamente o mesmo espaço que o campo de entrada. O processo de particionamento pode ser controlado especificando o número de partições de saídas desejadas e o número mínimo de elementos necessários numa partição. Se estes parâmetros de particionamento não forem especificados, o processo de particionamento escolhe valores apropriados para a arquitetura da máquina sendo utilizada.

### 3.4.3 Modelo de Execução

O modelo de execução do *Data Explorer* é baseado nos conceitos de fluxo de dados. Contudo, são providas características que estendem o conceito de fluxo de dados para permitir a criação de programas visuais que não poderiam ser criados usando simplesmente o modelo de fluxo de dados, essas características serão descritas na seção 3.4.3.1. Por exemplo, existem ferramentas que permitem guardar resultados parciais de um programa visual para ser usado em execuções subseqüentes. *Data Explorer* também prover várias ferramentas para controlar o fluxo de execução de programas visuais. A maioria destas ferramentas são análogas à construções encontradas em linguagens de programação. Por exemplo, são providas ferramentas que executam a função dos comandos IF, CASE e laços FOR.

#### 3.4.3.1 Fluxo de Dados

Em implementações realmente baseadas em fluxo de dados, todos os módulos são funções pura (i.e., as saídas são completamente definidas pelas entradas). Portanto, os processos não guardam estado. Quando as entradas de um módulo são recebidas, ele é executado, e quando termina de executar distribui os resultados para os módulos esperando, seguindo o fluxo dos dados.

The Collect module waits for inputs from the Isosurface and MapToPlane modules. Import would send its results to the waiting Isosurface and MapToPlane modules.

No exemplo da Figura 7, a primeira instancia (da esquerda para a direita) do módulo *Display* espera por dados de entrada dos módulos *AutoCamera* e *Transform*, já a segunda instancia do módulo *Display* espera por dados de entrada dos módulos *Isosurface* e *AutoCamera*. O módulo *Import* envia seus resultados ao módulo *Isosurface* que está a espera desses resultados, para processar enviar para os módulos que estão esperando pelo resultado. Na realidade, este modelo de execução é inteiramente dirigido por dados (data-driven) e de cima para baixo (top-down): a execução dos módulos depende somente na passagem dos dados pelo sistema. Este simples modelo de execução parece um mecanismo natural para a execução de programas visuais, mas

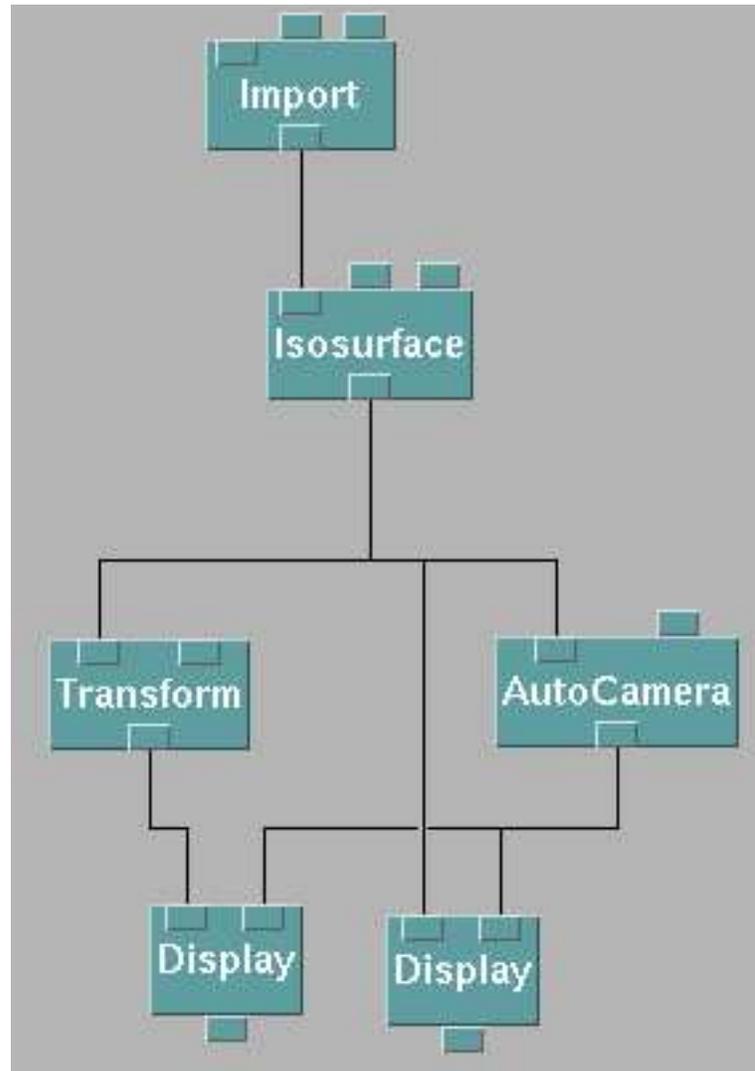


Figura 7: Um exemplo de programa visual equivalente ao *script* da seção 3.3.2.1

uma análise detalhada revela que os problemas do mundo real são mais complexos. Com o propósito de funcionar eficientemente, é vital que o sistema evite processamento desnecessário. Usualmente, existem duas razões para que os módulos presentes em um programa visual não necessitem ser executado quando chega a vez de cada um deles:

1. seus resultados não são realmente necessários para o resultado do programa
2. suas entradas não mudaram desde a última execução do programa (i.e. o resultado vai ser o mesmo)

As saídas de um programa de visualização ocorrem em módulos que tem efeitos colaterais. Eles produzem resultados fora do programa visual, tais como exibição de imagens no monitor ou criação de arquivos de saída. A menos que o resultado de um módulo afete um módulo que produz efeitos colaterais, esse módulo não precisa ser executado.

Eliminando os módulos que não são ancestrais de módulos com efeitos colaterais é feito no *Data Explorer* pelo pré-processamento do programa antes que a avaliação do programa baseado em fluxo de dados seja iniciada. Isto é feito percorrendo o grafo *bottom-up*, iniciando em cada módulo que é conhecido ter efeitos colaterais e marcando cada módulo que é encontrado. Uma vez que isto tenha terminado, os módulos que não tenham sido marcados não precisam ser executados. Note que a ordem exata que os módulos vão ser executados não pode ser controlada pelo usuário; por exemplo, módulos em ramos paralelos podem executar em qualquer ordem; só é garantido que um módulo que depende dos resultados de um ou mais módulos vai esperar que eles terminem de executar para começar a executar.

#### **3.4.4 Módulos**

O conjunto de módulos providos pelo *Data Explorer* são projetados com um alto grau de interoperabilidade. Interoperabilidade é simplesmente a propriedade que os módulos podem ser conectados em uma grande variedade de maneiras para obter efeitos diferentes. Esta propriedade surgiu de dois objetivos de projeto de módulos: fazer os módulos suficientemente primitivos, e fazê-los tão poderosos quanto possível.

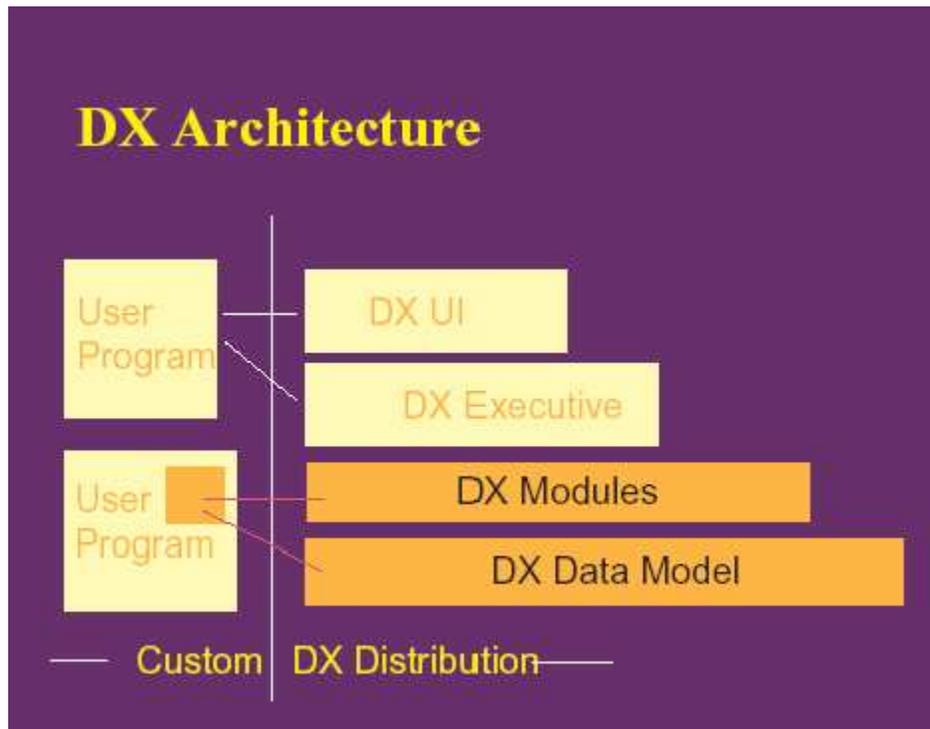


Figura 8: Arquitetura do *Data Explorer*

### 3.4.5 Estendendo as funcionalidades do OpenDX

As bibliotecas do *Data Explorer* permitem ao desenvolvedor estender as funcionalidades do *Data Explorer* de várias maneiras. Dependendo da tarefa que se deseja realizar, as maneiras de utilizar as bibliotecas vão variar. A figura 8 mostra a arquitetura do *Data Explorer*.

A interface com o usuário e o *executive* são os dois programas usados para criar e executar programas visuais, respectivamente. Subjacente ao *Data Explorer Executive* está uma coleção de módulos, que usa o modelo de dados do *Data Explorer* para manipular todos os objetos de um programa. *Data Explorer* Se você quer escrever um módulo para usar no *Visual Program Editor*, ou na linguagem de script, será necessário usar as bibliotecas `libDXlite.a`. Esta biblioteca contém todas as rotinas do modelo de dados que permitem consultar, manipular, e criar objetos.

### 3.4.5.1 Adicionando Funcionalidade ao *Visual Program Editor* ou a Linguagem de *Script*

Suponha que você quer realizar uma tarefa no *Visual Program Editor* ou com a linguagem de *script*, que nenhum módulo do *Data Explorer* realiza. Primeiramente deve ser investigado se os módulos existentes podem ser usados juntos em uma macro para realizar a tarefa. Por exemplo, o módulo *Compute*, junto com vários módulos de estruturação, tal como *Mark*, *Unmark*, e *Replace*, pode executar uma variedade de tarefas de transformação de dados, estes módulos estão descritos em (EXPLORER, 1997c). Quando você cria uma *macro*, você pode definir os parâmetros de entrada e saída. Se, contudo, ou porque usando macros não é tão eficiente quanto você quer, ou se a funcionalidade não possa ser provida usando os módulos existentes, você pode criar o seu próprio módulo no *Data Explorer*.

Quando você escreve um módulo para usar no *Data Explorer*, o *executive* continua sendo o processo que contém a função “*main*”. Seu módulo é simplesmente incorporado ao *Data Explorer* como um módulo:

***Inboard*** São compilados junto com o *Data Explorer*. Ou seja, a versão do *Data Explorer* que você usará será diferente da distribuição padrão.

***Outboard*** Estes módulos executam em processo separados. A link-edição de um módulo *outboard* é rápida, pois não envolve a criação de uma versão completa do *Data Explorer* (como a compilação dos módulos *inboard* o faz). Assim um módulo *outboard* é também mais fácil de depurar porque pode ser link-editado novamente mais facilmente. Contudo, os módulos *outboard* são tipicamente menos eficiente do que outros módulos, especialmente se quantidades significativas de dados precisam ser transferidos: dados são transferidos para e a partir de um módulo *outboard* via sockets ao invés de ponteiros para memória compartilhada que os módulos *inboard* e runtime-loadable usam.

***Carregável em tempo de Execução*** Podem ser carregados quando o *Data Explorer* é iniciado ou em qualquer momento depois da inicialização, e eles não requerem uma versão separada do *Data Explorer*, como os módulos *inboard* precisam. Assim, esses módulos tem a vantagem da portabilidade sem a desvantagem do overhead da transferência de dados

associada com os módulos *outboard*.

#### **3.4.5.2 Escrevendo um Programa *Stand-alone* usando o Modelo de Dados do *Data Explorer***

Você também pode querer escrever um programa *stand-alone* que usa o modelo de dados do *Data Explorer*. Neste caso, seu programa contém a função “*main*” e simplesmente link-edita as rotinas do modelo de dados do *Data Explorer*. Essas rotinas estão listadas no apêndice B do (EXPLORER, 1997a). Graficamente, esta maneira de estender o *Data Explorer* é representada pelo “User Program” de baixo na Figura 8, que embute as rotinas do modelo de dados do *Data Explorer* no programa do usuário.

#### **3.4.5.3 Escrevendo um Programa *Stand-alone* usando os Módulos do *Data Explorer***

Neste caso, seu programa contém a função “*main*” e é link-editado com a biblioteca *libDXcallm.a*, e usa o *DXCallModule* para fazer chamadas aos módulos do *Data Explorer*. Desta maneira é possível construir sistemas de visualização completos, por exemplo, a partir da importação de dados para a exibição de uma isosuperfície a partir do seu programa. Todavia, você não teria acesso as funcionalidades do *executive*, incluindo gerenciamento de cache, e controle de ordem de execução. Você vai, adicionalmente, ser responsável por remover objetos após terminar de usá-los.

#### **3.4.5.4 Controlando o *Executive* ou a Interface com o Usuário a partir de um Programa Separado**

Neste caso, você poderia escrever a sua própria interface, provendo um “look and feel” customizado, e enviar scripts do *Data Explorer* para o *executive* processar os dados e calcular imagens. Assim, você iria ter acesso a todas as funcionalidades providas pelo *executive* (gerenciamento de cache, controle de ordem de execução, e gerenciamento de objetos). Você poderia também controlar diretamente a interface com o usuário do *Data Explorer* a partir de um programa separado, carregando e executando programas visuais. Por exemplo, você poderia iniciar o *Data Explorer* com um programa de visualização quando a simulação terminar, com parâmetros dentro programa visual pre-

definidos. Graficamente, ambos os caso estão representados pelo “User Program” de cima na Figura 8.

## 4 *Método do Elemento Finito*

No campo da engenharia foi constatado, através da análise de muitos problemas complexos, que a formulação matemática dos mesmos é muito trabalhosa e, geralmente, não é possível resolvê-los por métodos analíticos. Nessas situações é possível recorrer ao uso de técnicas numéricas. O Método de Elemento Finito (MEF) é uma ferramenta muito poderosa na solução numérica de uma larga classe de problemas da física, da engenharia etc. O MEF recebe como entrada um problema (definição) exato, discretiza o problema e monta um sistema de equações algébricas para ser resolvido, cuja solução permite a obtenção de uma aproximação para a solução do problema exato.

### 4.1 O Problema Exato

O Método do Elemento Finito (MEF) foi inicialmente desenvolvido para resolver problemas a valores no contorno. Estes problemas são definidos de forma exata, no domínio geométrico contínuo, através da lei de comportamento para um campo vetorial que descreve o fenômeno físico. A lei de comportamento é dada por uma equação diferencial parcial, por exemplo, na forma

$$L\phi = f, \tag{4.1}$$

onde  $\phi$  é o campo vetorial que descreve o comportamento do fenômeno,  $L$  é o operador diferencial relacionado, e  $f$  é a função fonte. A função  $\phi$  está definida no domínio geométrico (no  $\mathbb{R}^n$ , por exemplo)  $S$ . Este campo vetorial ainda está sujeito a condições de contorno de vários tipos. Tudo isto pode ser colocado em uma única formulação chamada de forma fraca exata.

## 4.2 Discretização do Problema

A primeira noção importante do MEF é a partição (não trivial) do domínio geométrico. Usualmente, a partição de um domínio  $S$  é o conjunto de partes de  $S$ . Normalmente, cada parte possui uma forma simples (triângulos, quadriláteros (em 2-D) ou tetraedros, hexaedros (em 3-D), por exemplo). Cada parte é chamada de "Elemento Finito". O domínio original é aproximado pela montagem desses elementos numa malha geométrica. Os vértices dos elementos finitos compõem o conjunto de "Nós geométricos" ou "Pontos Nodais geométricos" da malha. O conceito de nó pode ser abstraído para designar informações empregadas na aproximação geométrica definida sobre a malha.

A definição da malha é fundamental para a construção dos conjuntos de funções empregados na aproximação da solução exata do problema. Estas funções são conhecidas como funções de forma. Funções afins são geralmente usadas como função de forma, devido a sua simplicidade. Funções de forma de mais alta ordem podem ser usadas para obter melhor precisão. As funções de forma são importantes para a aplicação do MEF e têm sido muito estudadas.

O campo vetorial que descreve o comportamento espacial do fenômeno é aproximado pela combinação linear de um conjunto de funções de forma definidas sobre os nós da malha de elementos finitos. Na expressão abaixo,  $\phi_i$  é o coeficiente da  $i$ -ésima função de forma, ou o valor nodal do nó  $i$ . A solução aproximada da equação (4.1) é dada pela equação abaixo:

$$\phi = \sum_{i=1}^n \phi_i N_i, \quad (4.2)$$

onde  $n$  é número total de nós da malha, e  $\phi_i$  será calculado na resolução do sistema (4.3), e  $N_i$  é a função de forma do nó  $i$ .

A partir da forma fraca exata, é definida a forma fraca discreta, envolvendo o campo vetorial discreto. Esta lei de comportamento discreto é equivalente a um sistema de equações algébricas, por exemplo, do tipo:

$$[K]\{\phi\} = \{b\}, \quad (4.3)$$

onde  $[K]$  é uma matriz de tamanho  $n \times n$ , e  $n$  é o número total de graus de

liberdade nodal vezes o número de nós da malha de elementos finitos,  $\phi$  é um vetor de tamanho  $n \times 1$ , formado pelos valores nodais  $\{\phi_i\}_{i=1}^m$ , sendo  $m$  o número de nós da malha;  $b$  é um vetor de tamanho  $n \times 1$  calculado baseado na função fonte  $f$ . Os elementos  $K_{ij}$  da matriz  $[K]$  e o elemento  $b_i$  do vetor  $b$  são calculados a partir das funções de forma dos nós e da lei de comportamento discreta (na chamada forma fraca discreta). Um procedimento com ordem de complexidade linear no número de elementos pode ser usado para montar a matriz do sistema algébrico.

Algumas referências para o MEF são (ZIENKIEWICZ; TAYLOR; TAYLOR, 2000; BATHE, 1995; SZABO; BABUSKA, 1991).

## 5 *Visualização no Plexus*

Como discutido na seção 2.1, não é possível implementar um sistema de visualização (*turnkey*) que resolva todos os problemas de todos os domínios de forma satisfatória. Por esta razão, definimos que o Plexus será a plataforma que irá prover as funcionalidades para guardar os dados relevantes da simulação no formato desejado pelo usuário, e prover funcionalidades para visualizar o resultado da simulação baseado apenas no tipo do dado, ou seja, para dados escalares, vetoriais, tensoriais com domínio em uma malha de elementos finitos.

Neste trabalho fizemos um estudo de como prover características de visualização ao Plexus, resultando em uma proposta de arquitetura de visualização para o mesmo, descrita na seção 5.1. E na seção 5.2 encontra-se a especificação da arquitetura de visualização para o *Data Explorer*, incluindo a implementação de um protótipo de programa de visualização de dados.

### 5.1 **Arquitetura de Visualização para o Plexus**

Os fenômenos físicos presentes em simulações do Plexus são representados como fenômenos computacionais, descrito na seção 2.2. O Plexus ainda não especifica a persistência dos dados gerados durante uma simulação. Assim precisamos especificar as seguintes funcionalidades necessárias para a análise de resultados de simulações: a entidade responsável por guardar e armazenar em memória secundária os dados de interesse e a entidade responsável por interagir com o sistema de visualização.

A entidade responsável por guardar e armazenar os dados é um fenômeno, chamado *Extractor*, que é especificado durante o projeto de um simulador no Plexus, onde são informados que fenômenos da simulação são relevantes para

a análise dos resultados e conseqüentemente precisam ser visualizados. Por *guardar*, queremos dizer coletar os dados dos fenômenos da simulação que precisam ser analisados após a simulação, e por *armazenar* queremos dizer transformar e escrever os dados para o formato especificado em memória secundária. O fenômeno *Extractor* é muito importante, pois permite aos usuários que precisem armazenar os dados em outro formato não suportado pelo Plexus que o façam implementando o seu próprio fenômeno *Extractor*.

Um outro fenômeno chamado *Viewer* é a entidade responsável por interagir com o sistema de visualização a ser utilizado na análise dos resultados. A interação do *Viewer* com o sistema de visualização é feita através de chamadas de sistema para *rodar* o sistema de visualização passando os parâmetros adequados. O *Viewer* pode também ser usado para implementar uma interface gráfica para que o usuário especifique a dos resultados.

O o sistema Plexus e os fenômenos *Viewer* e *Extractor* se relacionam de acordo com o padrão de projeto *Bridge* (GAMMA, 1995).

A Figura 9 (página 47) mostra um diagrama escrito em UML (*Unified Modeling Language*) (GROUP, 2003), apresentando uma descrição para a arquitetura de visualização evidenciando as entidades mais importantes no contexto desse trabalho. O componente *Plexus* representa o sistema Plexus como um todo, descrito em mais detalhes na seção 2.2. As classes abstratas *FenomenaExtractor* e *FenomenaViewer* representam as interfaces que os componentes precisam para fornecer o serviço de *Extractor* e *Viewer*.

O diagrama da Figura 9 também mostra que o sistema de visualização não precisa necessariamente estar na mesma máquina que o Plexus.

Com essa abordagem o Plexus não fica dependente de um sistema de visualização específico, e abre a perspectiva de poder criar versões do Plexus para sistemas de visualizações disponíveis no mercado. A seção 5.2, especifica a implementação dessa arquitetura utilizando o sistema de visualização *Data Explorer*.

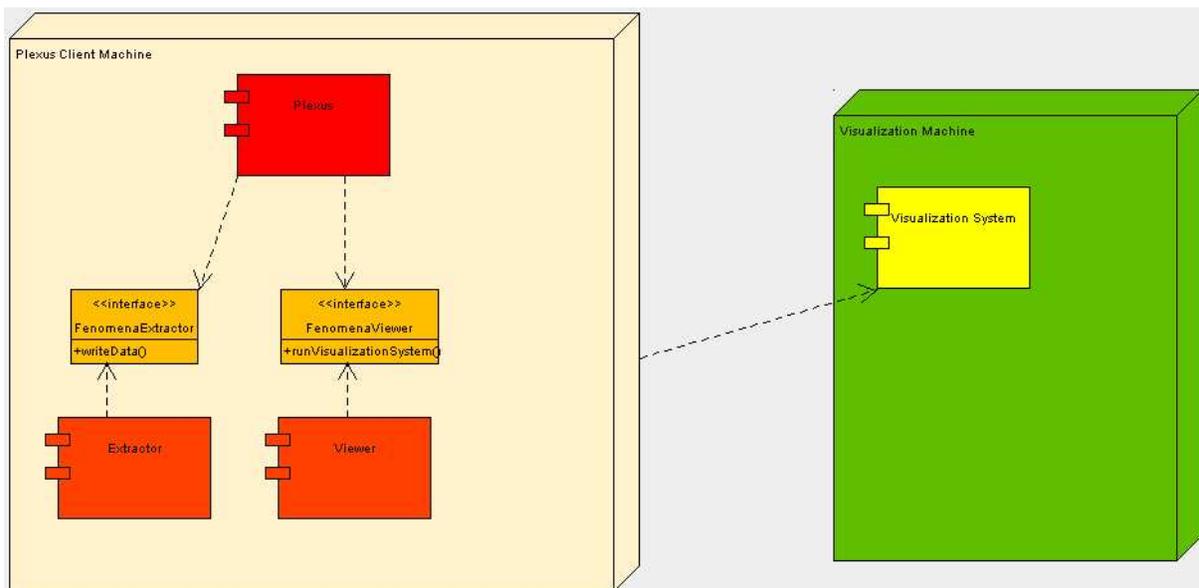


Figura 9: Diagrama UML de Componentes e *Deployment* da Arquitetura de Visualização do Plexus

## 5.2 Projeto dos Componentes da Arquitetura de Visualização do Plexus para o *Data Explorer*

O *Data Explorer*, como um sistema de visualização que é, possibilita a implementação do *Viewer* da arquitetura de visualização do Plexus apresentada na seção anterior. A seção 5.2.1, descreve a especificação do componente *Extractor* da arquitetura de visualização do Plexus, e a seção 5.2.2 apresenta a especificação do componente *Viewer*.

### 5.2.1 Especificação do Componente *Extractor*

A descrição da coleta dos dados dos fenômenos de uma simulação pelo *Extractor* não será apresentada nesse trabalho e será colocada como trabalho futuro, vamos apenas descrever como armazenar os dados resultantes de uma simulação para fins de visualização.

O Plexus possui um modelo de dados interno próprio (SANTOS; LENCASTRE; ALMEIDA, 2001), onde são armazenados os dados das entidades da simulação entre outros dados importantes para o sistema. Os dados resultantes de simulações do Plexus, relevantes no contexto desse trabalho, estão descritos na seção 2.1 de forma abstrata, onde apenas são mostrados os tipos de dados

e não as estruturas de dados usadas.

Como estamos usando o *Data Explorer* para visualizar os resultados da simulação, então para que isso seja possível precisamos transformar os dados do Plexus para o modelo de dados do *Data Explorer*, descrito na seção 3.4.2.

Como descrito na seção 2.1 os dados são malhas de elementos finitos, onde em cada nó está associado um dado escalar, vetorial ou tensorial. De acordo com a seção 3.4.2, o modelo de dados do *Data Explorer* suporta malhas de elementos finitos com dados escalares, vetoriais, e tensoriais. Assim sendo, esses dados do Plexus têm uma representação direta no modelo de dados do *Data Explorer*.

Os dados de fenômenos dependentes do tempo (dinâmicos) são uma sequência de malhas descrevendo o fenômeno ao longo do tempo. Caso o fenômeno seja dinâmico ele será mapeado num objeto *Series* do *Data Explorer*, que tem um conjunto de membros, onde cada membro é uma associação entre um passo no tempo e um *Field* representando o fenômeno simulado naquele instante de tempo.

### **5.2.2 Especificação do Componente Viewer**

O *Viewer* deve prover uma interface gráfica para possibilitar a especificação de parâmetros de visualização para o *Data Explorer*. O *Viewer* deve iniciar uma instância do *Data Explorer* na máquina cliente onde devem ser visualizados os resultados da simulação, passando como parâmetro um programa visual com uma interface gráfica para o usuário visualizar e interagir com os dados. Esse controle do *Data Explorer* por outra aplicação está descrita na seção 3.4.5.4, onde são descritos as possibilidades de extensão do *Data Explorer*.

### **5.2.3 PlexusViewer**

Esta seção descreve um programa de visualização implementado usando a ferramenta de criação de programas visuais do *Data Explorer* chamada *Visual Program Editor*, apresentada na seção 3.4. Este programa foi desenvolvido com o objetivo de conhecer e tornar familiares as ferramentas de criação de

aplicações de visualização providas pelo *Data Explorer* e vislumbrar as possibilidades de geração de programas de visualização a partir de simulações executadas no Plexus.

A interface principal do *PlexusViewer* é um painel de controle, mostrado na Figura 10, onde o usuário informa o arquivo de dados a ser processado e escolhe dentre as opções disponíveis de técnicas de visualização a opção desejada.

As técnicas de visualização foram organizadas seguindo a classificação descrita na seção 3.1.6.1. As técnicas de visualização implementadas foram:

- Técnicas de Visualização de Malhas: Vértices e Conexões
- Técnicas de Visualização de Campos Escalares: Interpolação, Gradiente, e *Contouring*
- Técnicas de Visualização de Campos Vetoriais: *Glyphs* e *Streamlines*

A descrição do funcionamento do *PlexusViewer* será apresentada em cenários de execução baseados no tipo dos dados de entrada. A seção 5.2.3.1 descreve um cenário em que o dado de entrada é um objeto *Field*, a seção 5.2.3.2 descreve o cenário em que o dado é objeto *Group*, a seção 5.2.3.3 apresenta o cenário em que o dado de entrada é um objeto *Series*, e a seção 5.2.3.4 descreve o cenário em que o dado de entrada nem é um *Field*, um *Group*, ou um *Series Group*.

Os cenários das seções 5.2.3.2, 5.2.3.3 e 5.2.3.4 só devem ser lidos após a leitura do cenário da seção 5.2.3.1.

### **5.2.3.1 Cenário: *Field***

Esta seção descreve um cenário de execução em que o usuário passa como parâmetro de entrada um arquivo de dados contendo um objeto *Field*. A descrição do funcionamento do *PlexusViewer*, neste cenário, será desenvolvida de trás para frente, ou seja, do efeito para o usuário (a imagem) para os dados, pois é assim que o *Data Explorer* funciona. O modelo de execução de programas visuais está descrito na seção 3.4.3.

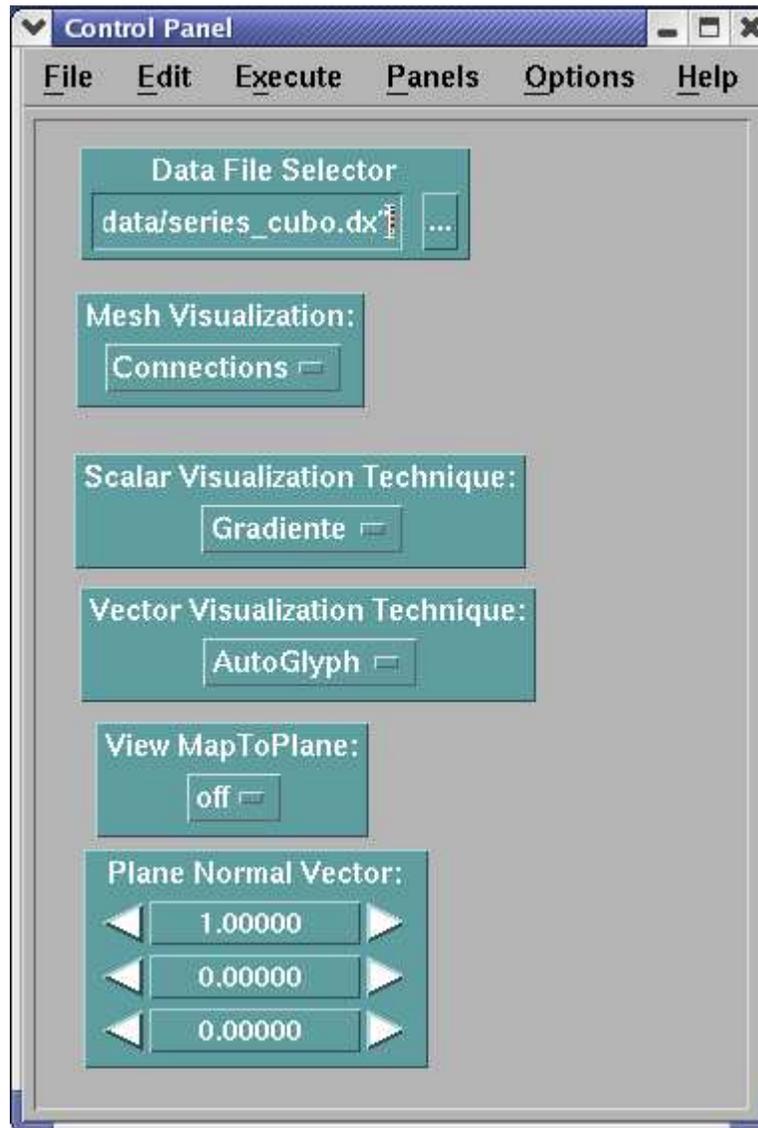


Figura 10: Painel de Controle do *PlexusViewer*

A Figura 11 mostra a página do programa que exibe o resultado da visualização. O módulo *Image* é responsável por criar uma janela chamada *Image* e exibir na mesma o resultado da visualização. A janela *Image* possui uma grande quantidade de controles de visualização, tais como navegação, rotação, zoom entre outros que estão descritos na seção 5.2.

A Figura 11 também mostra que o módulo *Image* depende do resultado do módulo *Collect*, que depende de *image\_data* e *Route*, já o módulo *Route* depende dos módulos *MapPlaneSelector* e *MapPlane*. Essa relação de dependência significa que se um módulo *x* depende de um módulo *y* então o módulo *x* precisa esperar pelo resultado da execução *y* para poder executar.

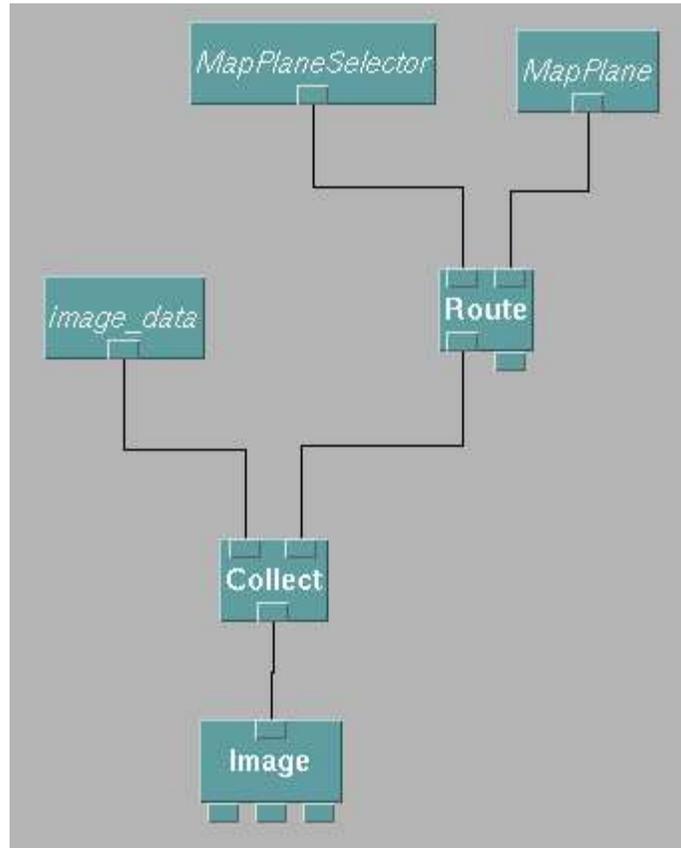


Figura 11: Resultado da Visualização

Assim para que a visualização seja exibida e necessário que os módulos *image\_data*, *MapPlaneSelector* e *MapPlane* sejam executados.

A Figura 12 mostra a página do programa onde é calculado o valor *image\_data*. O módulo *image\_data* é usado para armazenar os dados da imagem a ser visualizada para ser usado em outra página do programa. Neste caso *image\_data* depende do módulo *Switch*, que depende do *switch\_index*, do *invalid\_data\_processed*, do *procesed\_group*, do *processed\_series*, e do *field\_decoration*. O *Data Explorer*, por motivos de eficiência, implementa técnicas para evitar ter de avaliar todos parâmetros de entrada de um *Switch*, estas técnicas estão descritas em (EXPLORER, 1997b). A técnica usada nesse caso espera que a dependência do seletor do *Switch*, neste caso *switch\_index*, seja satisfeita para em seguida avaliar a opção escolhida dentre as possíveis. Como estamos num cenário em que o usuário passa um arquivo de dados contendo um *Field* então o módulo *field\_decoration* também precisa ser executado.

A Figura 13 mostra a página do programa, onde, dentre outras coisas, cal-

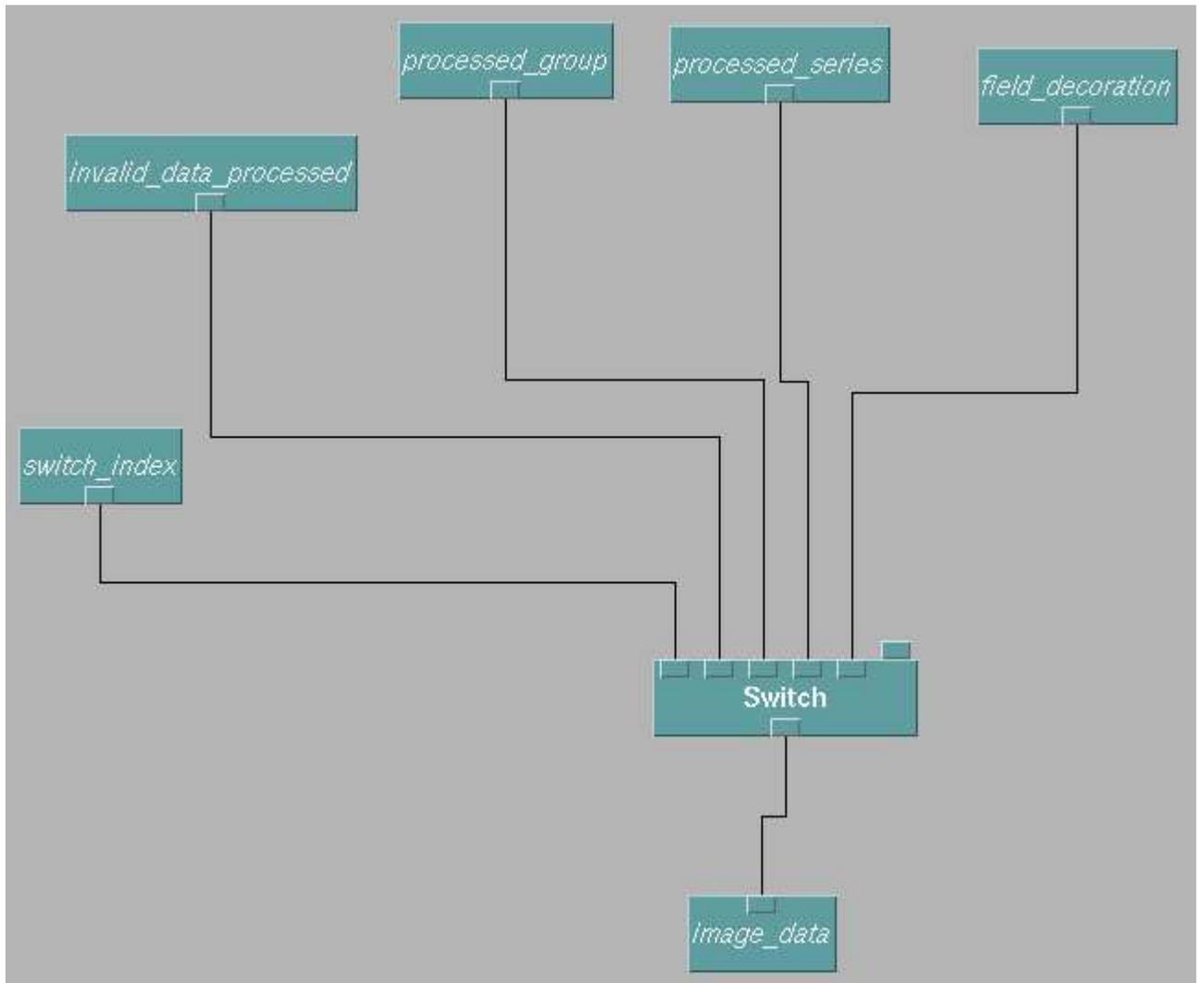


Figura 12: Seleção de Resultado da Visualização

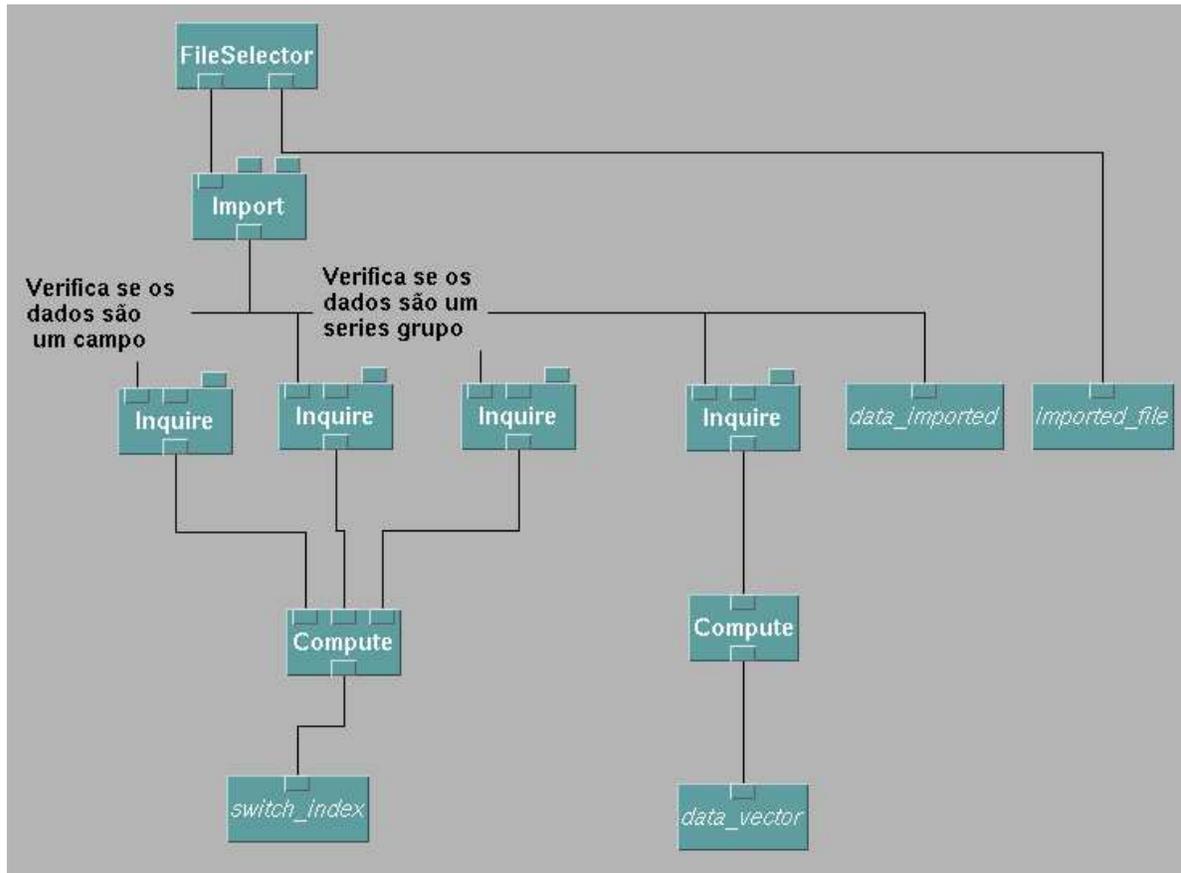


Figura 13: Leitura de Dados

cula o módulo *switch\_index*, que depende do módulo *Compute* o qual depende de três instâncias do módulo *Inquire*, esse módulo retorna 0 ou 1 e é usado para verificar se um objeto tem uma determinada característica ou não. Cada instância do módulo *Inquire* depende do módulo *Import*, que depende do módulo *FileSelector*.

A instância do módulo *Compute* mostrada na Figura 13 computa a seguinte expressão:  $4field + 3group + 2series$ , onde *field*, *group* e *series* são os valores retornados por cada instância do módulo *Inquire* da direita para a esquerda na Figura 13.

Nessa mesma página do programa também são calculados os módulos: *data\_vector*, *data\_imported*, e *imported\_file*. O módulo *data\_vector* representa se os dados importados são vetoriais ou não, o módulo *data\_imported* representa o objeto importado do arquivo de dados, e o módulo *imported\_file* representa o nome do arquivo de dados importado.

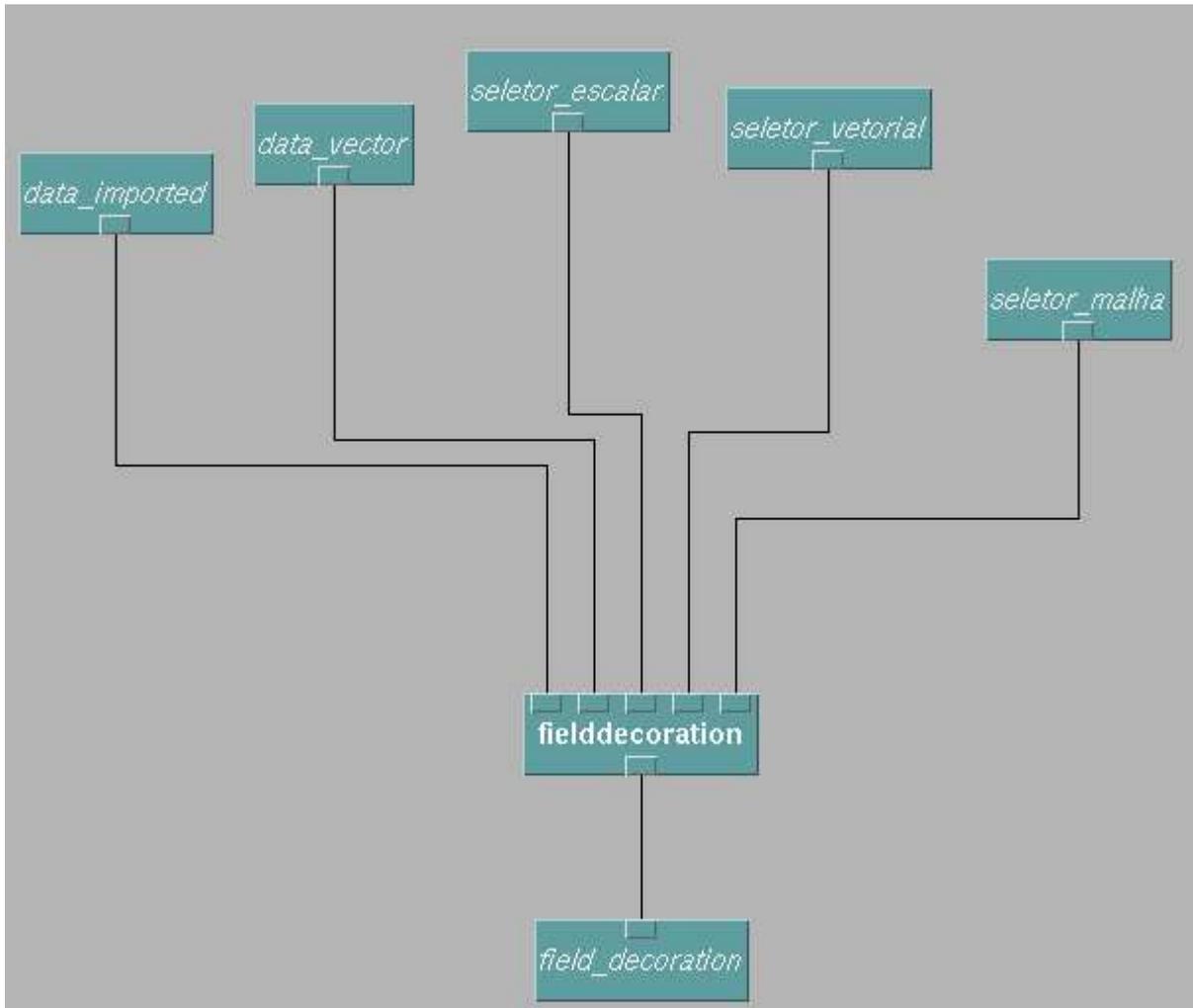


Figura 14: Processamento de Fields

A Figura 14 mostra a página do *PlexusViewer* responsável por calcular o módulo *field\_decoration*. O módulo *field\_decoration* depende do módulo *fielddecoration*, que depende dos módulos: *data\_imported*, *data\_vector*, *seletor\_escalar*, *seletor\_vetorial* e *seletor\_malha*.

Como *fielddecoration* não é um módulo pré-definido e sim uma *macro* criada para processar objetos *Field*, então não há nenhuma otimização para a avaliação dos módulos dos quais ele depende. A *macro* *fielddecoration* foi criada, pois o código de “decoração” de um objeto *Field* também é utilizado em outras páginas do programa.

A Figura 15 mostra a página do *PlexusViewer*, onde os módulos de interação com o usuário estão localizados. Nesta página podemos ver que os módulos: *seletor\_escalar*, *seletor\_vetorial*, *seletor\_malha*, e *MapPlaneSelector* dependem

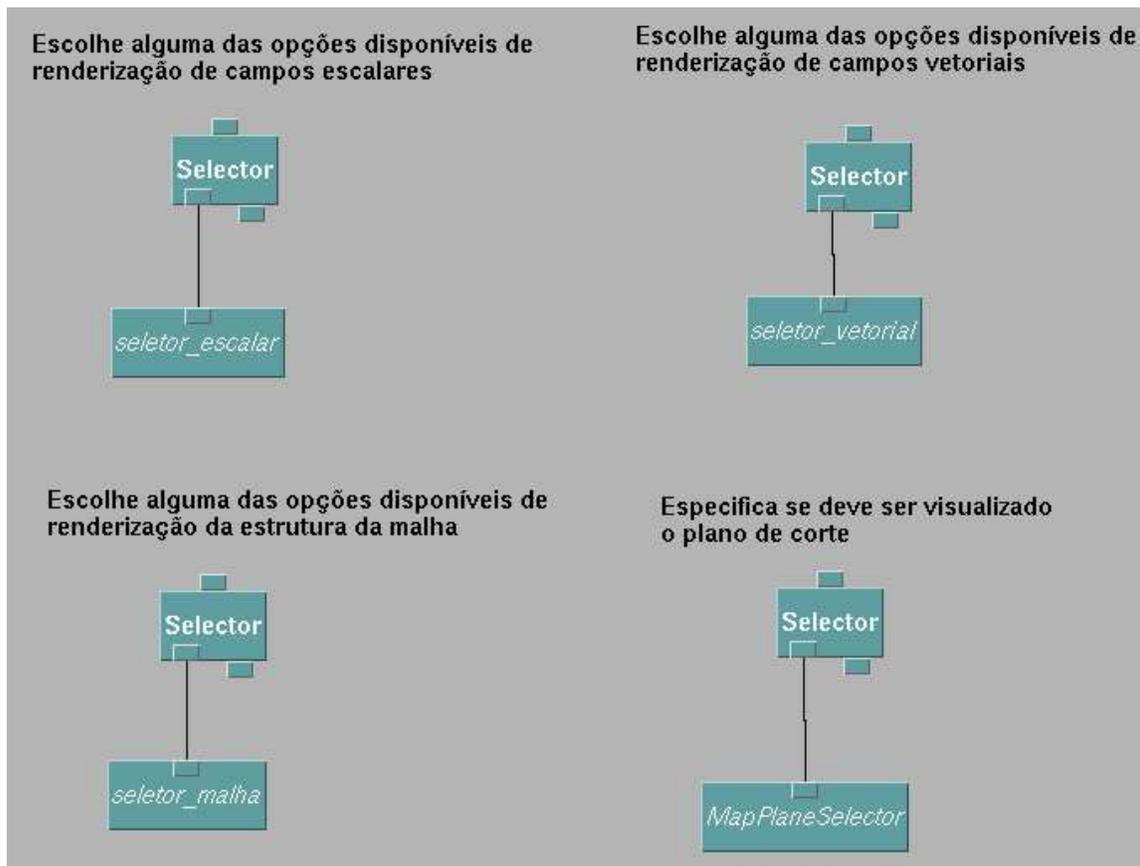


Figura 15: Entrada de Dados de Controle da Visualização

de instâncias do módulo *Selector*, que são elementos de interface com o usuário representados na Figura 10 como listas.

A Figura 16 mostra a página do *PlexusViewer* especificada para gerar um plano de corte. O módulo *MapPlane* depende do módulo *AutoColor*, que depende do módulo *MapToPlane*. Já o módulo *MapToPlane* depende dos módulos *data\_imported*, *poke*, e *Vector*.

O instância módulo *AutoColor*, da Figura 16, é responsável por colorir o *Field* retornado pelo módulo *MapToPlane*. A instância de *MapToPlane* recebe como entrada um *Field* de dados representado por *data\_imported*, um ponto usado para definir o plano em que vão ser mapeados os dados representado por *poke*, e um vetor que junto como o ponto determinam um plano perpendicular ao vetor e que contem o ponto, representdado na Figura 16 pelo elemento de interface *Vector*, que está no painel de controle da Figura 10.

O único módulo que ainda não foi apresentado, e que é preciso executar, foi o *poke*, ele está definido na página do *PlexusViewer* mostrado na Figura 17.

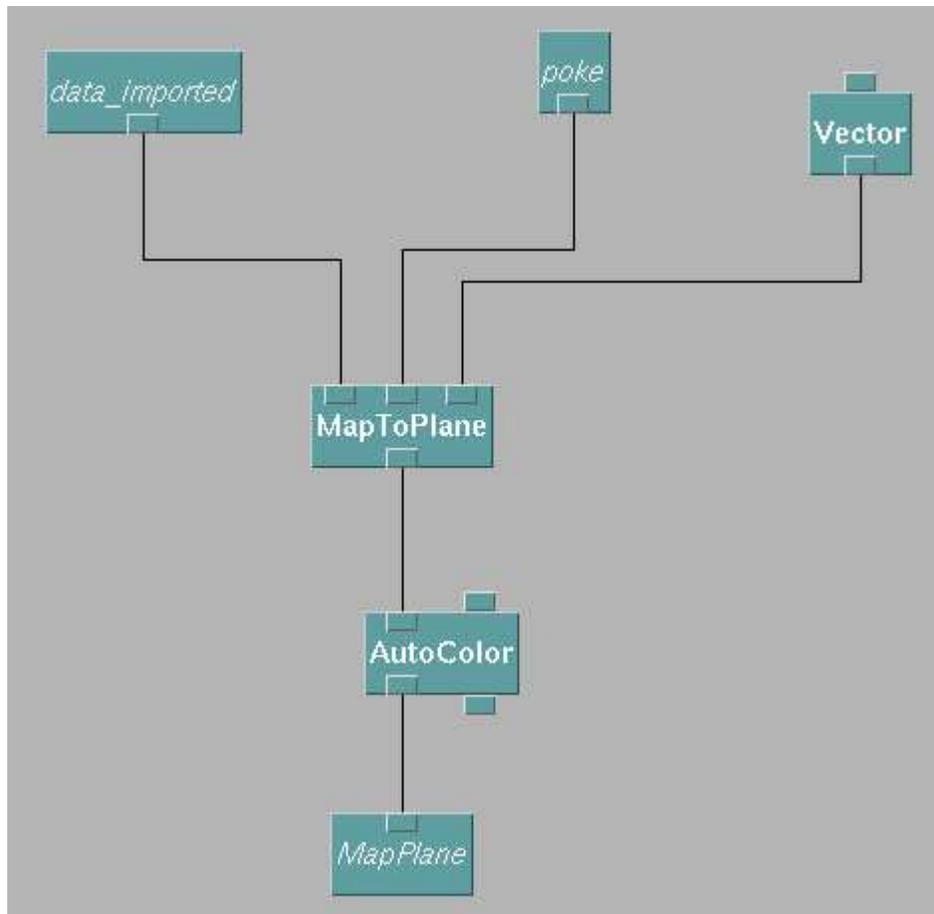


Figura 16: Visualização de Planos de Corte

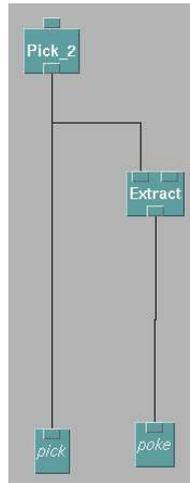


Figura 17: Pick

O módulo *poke* depende de uma instância do módulo *Extract*, que depende do módulo *Pick\_2*. *Pick\_2* é um elemento de interface usado para que o usuário marque pontos na janela *Image* para que seja usado nos programas visuais. O módulo *Pick\_2* é executado sempre que um usuário ativa o modo de visualização “*Pick*” e clica em algum pixel da janela *Image*.

### 5.2.3.2 Cenário: *Group*

Esta seção descreve um cenário de execução em que o usuário passa como parâmetro de entrada um arquivo de dados contendo um objeto *Group*.

A descrição do funcionamento do *PlexusViewer*, neste cenário, irá explicar a diferença entre o cenário da seção 5.2.3.1 e o desta seção. A ponto de divergência entre o cenário para o *Field* se dá no cálculo do módulo *image\_data*, pois o cenário do *Field* o módulo *switch\_index* foi avaliado e escolheu o módulo *field\_decoration*, mas que nesse cenário selecionará *processed\_group*.

A Figura 18 mostra a página do *PlexusViewer*, que descreve como é calculado o módulo *processed\_group*. Como podemos ver o *processed\_group* só depende do módulo *AutoColor*, que só depende do *imported\_data*.

### 5.2.3.3 Cenário: *Series Group*

Esta seção descreve um cenário de execução em que o usuário passa como parâmetro de entrada um arquivo de dados contendo um objeto *Group*.

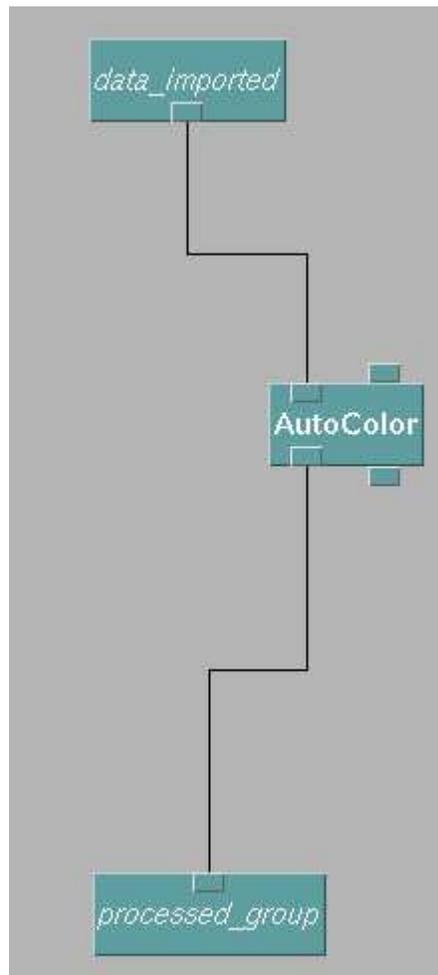


Figura 18: Processamento de Grupos Genéricos

A descrição do funcionamento do *PlexusViewer*, neste cenário, irá explicar a diferença entre o cenário da seção 5.2.3.1 e o desta seção. A ponto de divergência entre o cenário para o *Field* se dá no cálculo do módulo *image\_data*, pois o cenário do *Field* o módulo *switch\_index* foi avaliado e escolheu o módulo *field\_decoration*, mas que nesse cenário selecionará *processed\_series*.

A Figura 18 mostra a página do *PlexusViewer*, que descreve como é calculado o módulo *processed\_series*. Como podemos notar que *processed\_series* só depende de *fielddecoration*, que depende de uma instância do módulo *Select* e de outros módulos descritos no processamento de *Fields* do cenário 5.2.3.1 mostrados na Figura 14. Assim temos analisar a avaliação do módulo *Select*, que é usado para selecionar um dos *Fields* do módulo *imported\_data*, que é um objeto *Series*, indexado pelo módulo *Sequence*. O módulo *Sequence* é um elemento de interface que representa um *player* (tocador) de animações.

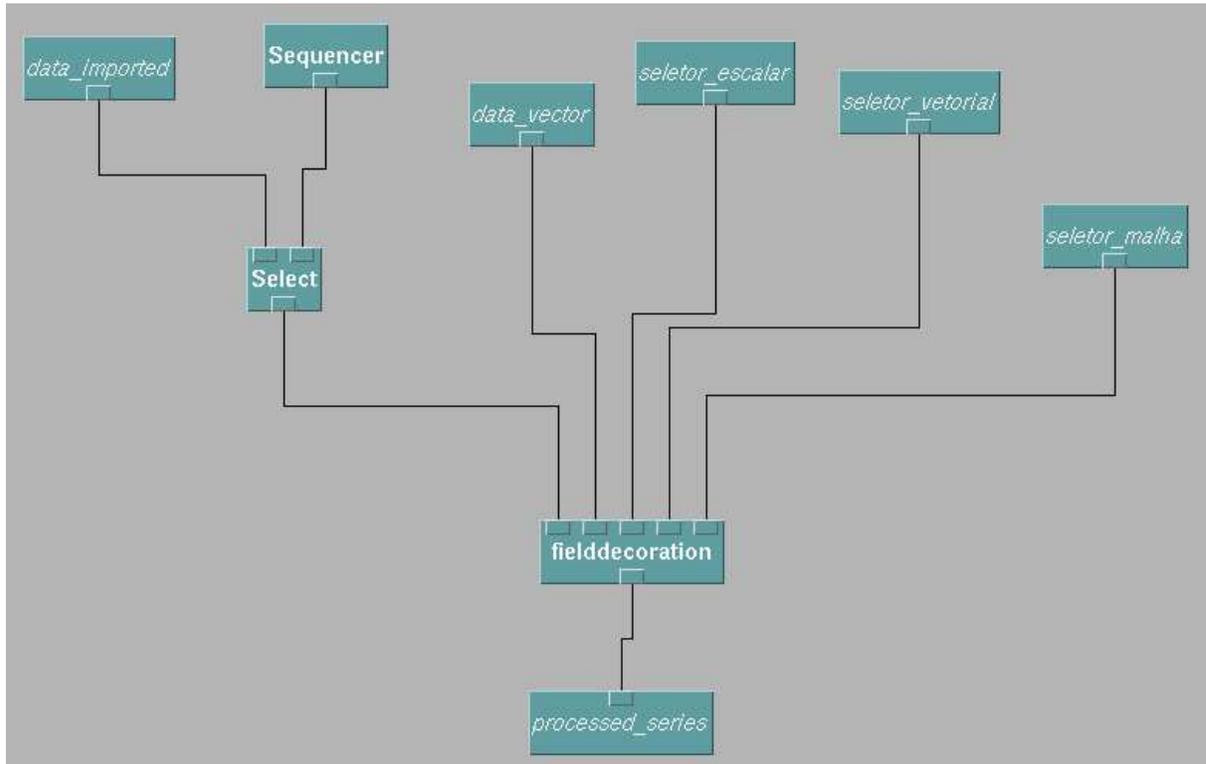


Figura 19: Processamento de Series Group

#### 5.2.3.4 Cenário: Dados Inválidos

Esta seção descreve um cenário de execução em que o usuário passa como parâmetro de entrada um arquivo de dados que não contem objetos do seguintes tipos: *Field*, *Group*, e *Series*.

A descrição do funcionamento do *PlexusViewer*, neste cenário, irá explicar a diferença entre o cenário da seção 5.2.3.1 e o desta seção. A ponto de divergência entre o cenário para o *Field* se dá no calculo do módulo *image\_data*, pois o cenário do *Field* o módulo *switch\_index* foi avaliado e escolheu o módulo *field\_decoration*, mas que nesse cenário selecionará *invalid\_data\_processed*.

A Figura 20 mostra a página do *PlexusViewer*, que descreve como é calculado o módulo *invalid\_data\_processed*. Como podemos ver o *invalid\_data\_processed* só depende de uma instância do módulo *Caption*, que só depende do módulo *Format*, que só depende do módulo *imported\_file*, que representa o nome do arquivo de dados.

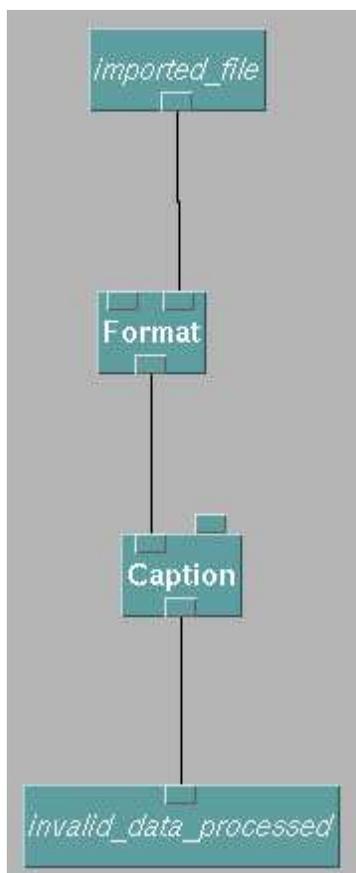


Figura 20: Mensagem de erro para os dados inválidos

## 5.2.4 Estudo de caso: Simulações de Fenômenos em Órgãos Humanos

Esta seção mostra o uso da arquitetura de visualização do Plexus instanciada para o *Data Explorer*, na visualização dos resultados da simulação de fenômenos biomecânicos no sistema cardiovascular humano, desenvolvida no Departamento de Engenharia Mecânica da UFPE, coordenada pelo professor Felix C. G. Santos.

O processo de preparação da simulação foi composto das seguintes etapas, tendo como dado imagens médicas na forma de imagens de seções transversais dos órgãos a serem reconstruídos:

**Segmentação** é o processo de identificação das estruturas presentes em cada fatia. O resultado desta etapa é a representação das seções por figuras poligonais planas. Nesta etapa são incorporados conhecimentos sobre as materiais identificados por este processo.

**Reconstrução Geométrica** A partir das informações coletadas na etapa anterior, tipicamente figuras poligonais planas representando cada fatia, é construída uma forma geométrica descrita através de uma malha de elementos tetraédricos.

**Geração de Malhas** A partir da descrição da geometria na forma de uma malha, esta é condicionada, isto é, modificada para evitar mal-condicionamento da solução numérica.

Os sistemas envolvidos nas etapas desse processo estão integrados e modularizados, permitindo que a etapa de segmentação seja refinada sem que seja necessário a modificação das outras etapas do processo.

A Figura 21 mostra o resultado da visualização da malha de elementos finitos, utilizando a técnica de visualização do contorno da malha, ou seja, da superfície externa da malha.

A Figura 22 (página 62) mostra o resultado da visualização da malha de elementos finitos, utilizando a técnica de visualização que mostra as conexões entre os nós da malha, ou seja, visualiza as arestas dos elementos finitos.

A Figura 23 (página 63) mostra o resultado da visualização da transferência

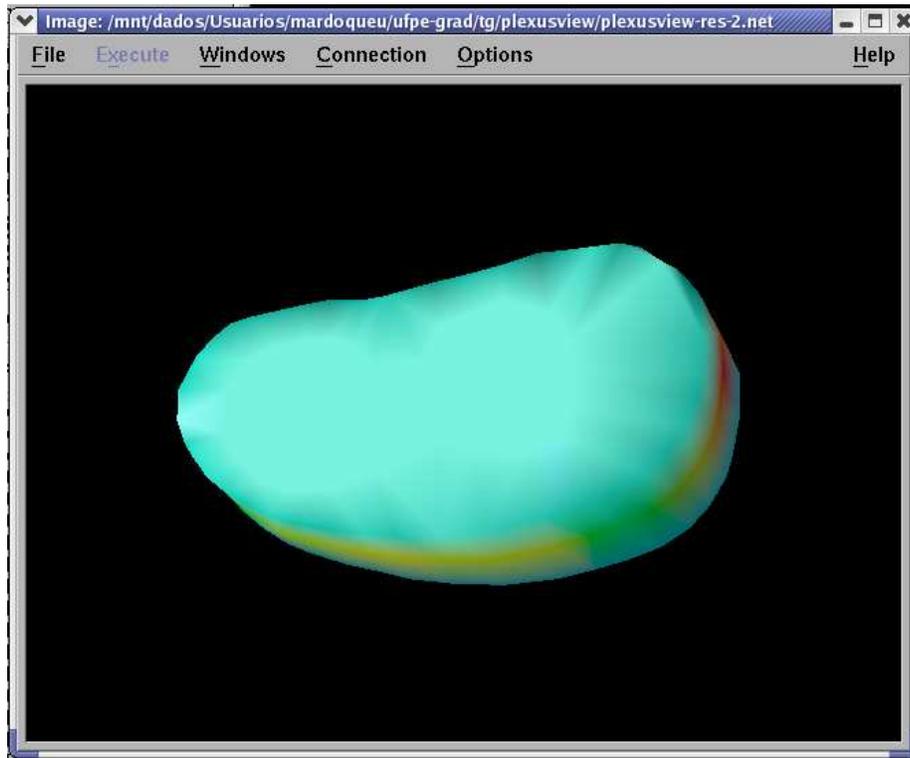


Figura 21: Resultado da Aplicação da Técnica de Visualização do Contorno

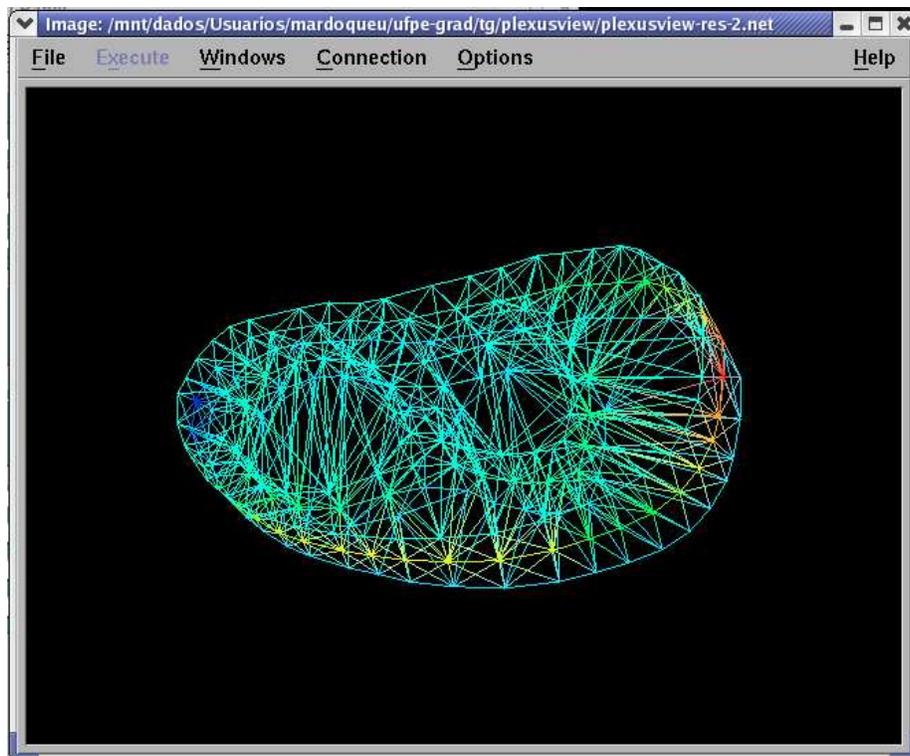


Figura 22: Resultado da Aplicação da Técnica de Visualização de Malhas pelas Arestas

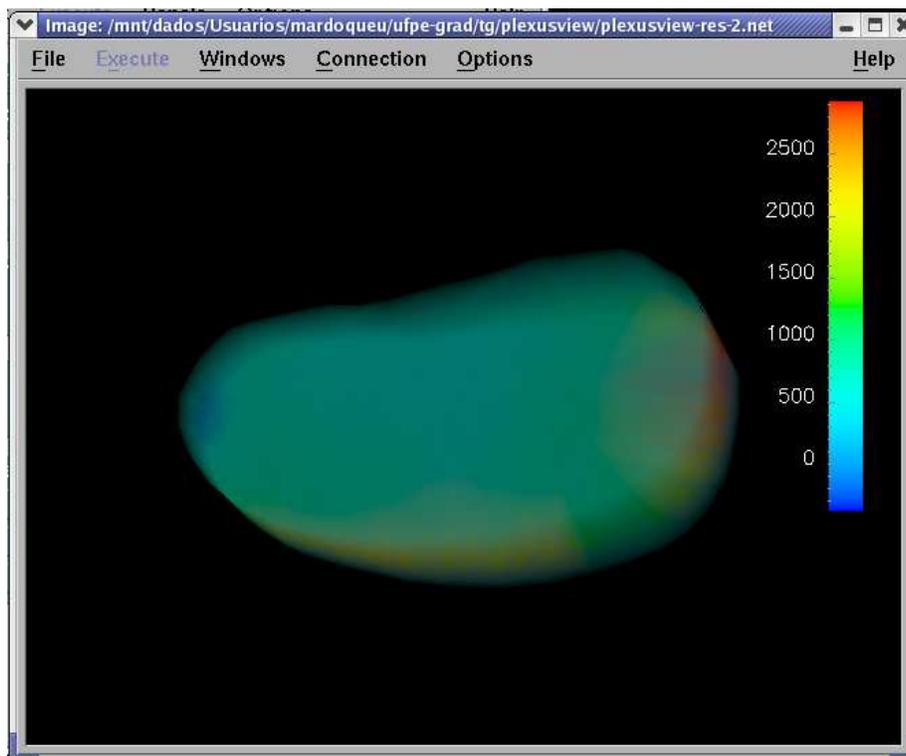


Figura 23: Resultado da Aplicação da Técnica de Visualização Gradiente

de calor entre a fatia da base e a do topo, utilizando a técnica de visualização do gradiente.

A Figura 24 (página 64) mostra o resultado da visualização da iso-superfície contendo todos os pontos com temperatura igual à temperatura média dos nós da malha.

Embora os resultados mostrados nas figuras 23 e 24 sejam visualmente menos precisos do que os valores que os geraram, é praticamente impossível para qualquer usuário extrair a partir de meros valores a informação que pode ser compreendida a partir da visualização. Este exemplo ilustra o poder da visualização restando, ainda, a interação com as informações.

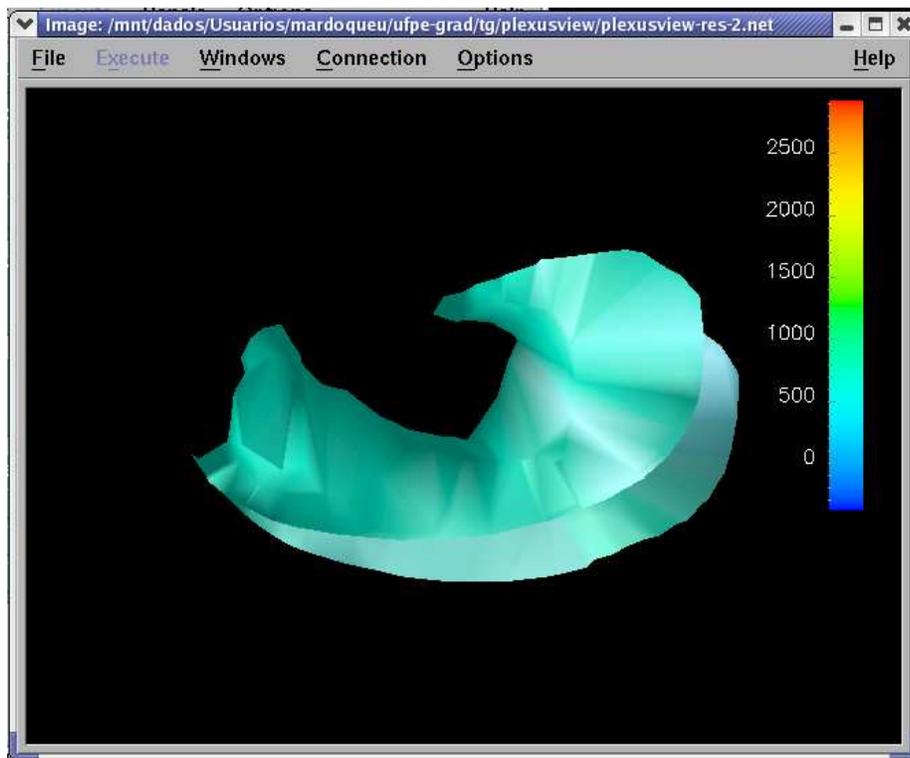


Figura 24: Resultado da Aplicação da Técnica de Visualização *Contouring*

## 6 *Contribuições e Trabalhos Futuros*

Este capítulo apresenta as conclusões deste trabalho. As conclusões estão organizadas em contribuições e em trabalhos futuros descrevendo os próximos passos no desenvolvimento de visualizações no Plexus. As contribuições estão descritas na seção 6.1, e os trabalhos futuros na seção 6.2.

### 6.1 *Contribuições*

Pelo fato de não haver documentação em português, uma contribuição deste trabalho é a conceitualização das tecnologias de visualização. Esta conceitualização consistiu da descrição dos principais conceitos da visualização de dados e de visualização científica, e incluiu a conceitualização das ferramentas para visualização com a descrição pormenorizada dos sistemas de visualização.

A proposta de uma arquitetura de visualização para o Plexus, na seção 5.1, consistindo da especificação das entidades envolvidas e o relacionamento entre as entidades usando UML. O projeto dos componentes da arquitetura de visualização do Plexus, onde o sistema de visualização é o *Data Explorer*, descrito na seção 3.4, consistindo da especificação dos componentes *Extractor* e *Viewer*, e com a implementação de um protótipo de programa de visualização a ser usado como ferramenta de auxílio a análise dos resultados de simulações, está descrita na seção 5.2.

## 6.2 Trabalhos Futuros

Esta seção apresenta algumas funcionalidades que podem ser implementadas, para dar continuidade ao trabalho já desenvolvido com o objetivo de produzir um ambiente de visualização e análise de resultados de simulação para o Plexus. Segue abaixo uma breve descrição de algumas funcionalidades:

**Visualização de Métricas de Qualidade de Malhas** Consiste da visualização de métricas que estão fundamentadas em critérios de avaliação da qualidade de malhas de elementos finitos. Estes critérios são baseados exclusivamente em características geométricas de cada elemento finito da malha. Os aspectos teóricos e uma interpretação geométrica de cada critério estão descritos em (FREY; BOROUCHAKI, 1998).

A implementação dessa técnica de visualização de qualidade de malhas implica em criar um módulo no *Data Explorer*, que recebe como parâmetro de entrada um *Field* representando a malha de elementos finitos e retorna um outro *Field* onde a malha é a mesma do campo de entrada, mas os dados são dependentes das conexões e o valor dos dados associados com cada elemento é o valor da métrica a ser calculada. O problema é que o cálculo das métricas depende da estrutura das malhas, e os módulos pré-definidos não permitem acesso a estrutura da malha de um *Field*. A solução consiste em escrever um módulo para o *Data Explorer* que calcule o valor da métrica para cada elemento da malha acessando a estrutura da malha através da interface de programação descrita no guia de programação do *Data Explorer* (EXPLORER, 1997a).

**Visualização Interativa de Malhas** Esta funcionalidade consiste da visualização interativa de malhas com o objetivo de avaliar o comportamento de fenômenos no interior de uma geometria tridimensional. Segue abaixo algumas funcionalidades que precisam ser implementadas para permitir a interação desejada:

**Cortes não Destrutivos** consiste em definir planos de corte para definir um limiar para separar os elementos que vão ser removidos da imagem visualizada dos que vão ficar. Os elementos que estiverem somente no semi-espaço do vetor normal do plano de corte, ou seja nenhum ponto está no outro semi-espaço.

**Cortes não Destrutivos com Navegação** esta funcionalidade consiste em navegar numa malha de elementos, onde a cada deslocamento da câmara um outro plano de corte é definido. O grande desafio está em implementar essa técnica de maneira eficiente, dado que em aplicações reais as malhas são muito grandes o causa lentidão no processo de visualização.

**Implementação da Arquitetura de Visualização do Plexus, para o *Data Explorer*** essa atividade consiste em implementar o projeto especificado na seção 5.2, com objetivo validar a arquitetura de visualização proposta para o Plexus.

# Referências Bibliográficas

- AVS. *Advanced Visualization System*. 2002. Disponível em: <<http://www.avs.com>>. Acesso em: 3 de Janeiro de 2002.
- BATHE, K.-J. *Finite Element Procedures*. 2nd ed. [S.l.]: Prentice Hall, 1995.
- BRITTAIN, D. L. Design of an end-user data visualization system. In: *Proceedings of Visualization 90*. IEEE Computer Society Press: IEEE, 1995. p. 323-328.
- BUSHMANN, F. et al. *PATTERN-ORIENTED SOFTWARE ARCHITECTURE A System of Patterns*. 6th ed. [S.l.]: WILEY, 2001.
- COLLONS, N. S.; HABER, R. B.; LUCAS, B. A data model for scientific visualization with provisions for regular and irregular grids. In: *Visualization '91, Proceedings*. [S.l.]: IEEE, 1992. (Visualization), p. 298 -305.
- EXPLORER, I. V. D. *Programmer's Reference*. Seventh. Thomas J. Watson Research Center/Hawthorne, maio 1997.
- EXPLORER, I. V. D. *User's Guide*. Seventh. Thomas J. Watson Research Center/Hawthorne, maio 1997.
- EXPLORER, I. V. D. *User's Reference*. Fourth. Thomas J. Watson Research Center/Hawthorne, maio 1997.
- FRERY, A. C.; KELNER, J. *Realidade Virtual e Multimídia*. 2003. Disponível em: <<http://www.cin.ufpe.br/if124>>. Acesso em: 6 de março de 2003.
- FREY, P. J.; BOROUCHAKI, H. Geometric evaluation of finite element surface meshes. In: *Finite Elements in Analysis and Design 31*. [S.l.: s.n.], 1998. p. 33-53.
- GALLAGHER, R. S. *Computer Visualization Graphic Techniques for Scientific and Engineering Analysis*. [S.l.]: CRC Press, 1995.
- GAMMA, E. et al. *Design Patterns Elements of Reusable Object-Oriented Software*. United States of America: [s.n.], 1995.
- GLOBUS, A. *A Software Model for Visualization of Time Dependent 3-D Computational Fluid Dynamics Results*. [S.l.], nov. 1992.
- GROUP, O. M. *UML Resource Page*. 2003. Disponível em: <<http://www.omg.org/uml/>>. Acesso em: 6 de março de 2003.

- GROUP, T. N. A. *Iris Explorer*. 2002. Disponível em: <<http://www.nag.co.uk/>>. Acesso em: 3 de Janeiro de 2002.
- HULTQUIST, J.; RAIBLE, E. Superglue: A programming environment for scientific visualization. In: *Technical Report RNR-92-014*. [S.l.]: NASA Ames Research Center, NAS Systems Division, 1992. (Applied Technical Branch).
- IBM. *Open Data Explorer*. 2002. Disponível em: <<http://www.opendx.org>>. Acesso em: 3 de Janeiro de 2002.
- IEEE. *IEEE Symposium on Information Visualization*. 2003. Disponível em: <<http://www.infvis.org/>>. Acesso em: 3 de janeiro de 2003.
- KUHL, D. F.; WEATHERLY, D. R.; DAHMANN, D. J. *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*. [S.l.]: Prentice-Hall, Inc., 1999.
- OPENGL. *OpenGL*. 2002. Disponível em: <<http://www.opengl.org>>. Acesso em: 3 de Janeiro de 2002.
- SANTOS, F. C. G.; LENCASTRE, M.; ALMEIDA, I. Data and process management in a FEM simulation environment for coupled multi-physics phenomena. In: *Fifth International Symposium On Computer Methods In Biomechanics and Biomedical Engineering*. Rome-Italy: [s.n.], 2001.
- SANTOS, F. C. G.; LENCASTRE, M.; ALMEIDA, I. An approach for FEM simulation development. In: *International Workshop on Computational Codes: the Technological Aspects of Mathematics advances in computing and software development for Differential Equations*. Università di BARI - Itália: [s.n.], 2002.
- SANTOS, F. C. G.; LENCASTRE, M.; ALMEIDA, I. FEM simulation environment for coupled multi-physics phenomena. In: *Simulation and Planning In High Autonomy Systems - AIS2002*. Lisboa-Portugal: [s.n.], 2002.
- SANTOS, F. C. G.; LENCASTRE, M.; ALMEIDA, I. Fem simulator based on skeletons for coupled phenomena (fem - simulator skeleton). In: *The Second Latin American Conference on Pattern Languages of Programming SugarloafPLOP'2002 Conference*. Itaipava, Rio Janeiro, Brasil: [s.n.], 2002.
- SANTOS, F. C. G.; VIEIRA, M.; LENCASTRE, M. Workflow for simulators based on finite element method. In: *International Conference on Computational Science 2003*. Saint Petersburg, Russian Federation: [s.n.], 2003.
- SCHROEDER, W.; KENMARTIN; LORENSEN, B. *The Visualization Toolkit*. 2nd ed. [S.l.]: Prentice-Hall, Inc., 1998.
- SHNEIDERMAN, B. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. 3 ed. [S.l.]: Addison Wesley, 1998.
- SILICON GRAPHICS, I. *Open Inventor, An SGI Technology*. 2002. Disponível em: <<http://www.sgi.com/software/inventor>>. Acesso em: 3 de Janeiro de 2002.

SZABO, B. A.; BABUSKA, I. *Finite Element Analysis*. New York: Wiley Interscience, 1991.

ZIENKIEWICZ, O. C.; TAYLOR, R. L.; TAYLOR, R. L. *Finite Element Method: Volume 1, The Basis*. 5th ed. [S.l.]: Butterworth-Heinemann, 2000. (Finite Element Method, v. 1).