PROGRAMAS PARA VISUALIZAÇÃO DE SIMULAÇÕES COMPUTACIONAIS DE DINÂMICA DE FLUIDOS (CFD) USANDO OPENDX

Marcelo Mariano da Silva

DISSERTAÇÃO SUBMETIDA AO PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA E TECNOLOGIA NUCLEARES DO INSTITUTO DE ENGENHARIA NUCLEAR DA COMISSÃO NACIONAL DE ENERGIA NUCLEAR COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA NUCLEAR - ÊNFASE PROFISSIONAL EM ENGENHARIA DE REATORES

Aprovada por:

Prof. Paulo Augusto Berquó de Sampaio(PPGIEN/CNEN)

Prof. Reynaldo Jacques Jospin (PPGIEN/CNEN)

Prof. Antônio José da Silva Neto (IPRJ/UERJ)

RIO DE JANEIRO, RJ - BRASIL
JANEIRO DE 2008

S I LV Silva, Marcelo Mariano da.

Programas para visualização de simulações computacionais de dinâmica de fluidos (CFD) usando o OpenDx / Marcelo Mariano da Silva. - Rio de Janeiro: CNEN/IEN, 2008.

x, 90f. : il. ; 31 cm.

Orientador: Paulo Augusto Berquó de Sampaio

Dissertação (Mestrado em Engenharia de Reatores) – Instituto de Engenharia Nuclear, PPGIEN, 2008.

1.Visualização Científica. 2. Elementos Finitos.
3. Fluidodinâmica Computacional. I.Programa de Pós-Graduação em Ciência e Tecnologia de Reatores. II.Título.

CDD
CDU

Aos meus irmãos, Alexandre e Ricardo, que sempre me inspiraram e me apoiaram em tudo que desejei fazer.

A minha mãe, pelo apoio em todos os momentos difíceis.

A todos meus amigos cariocas, que me ajudaram muito no ano que estive na cidade maravilhosa.

Ao Sr Jorge Loureiro e seu filho Hugo Attan pela ajuda nos meus primeiros meses no Rio, aos quais serei sempre grato.

Ao professor David Adjuto Botelho, que nos deixou recentemente e que muito me ajudou na minha passagem pelo IEN.

AGRADECIMENTOS

Ao meu orientador, Paulo Augusto Berquó de Sampaio por todo o empenho e atenção despendida para que eu absorvesse o máximo de conhecimento possível em meu curso.

Aos professores Reynaldo Jacques Jospin, Celso Marcelo Franklin Lapa, Maria Lourdes Moreira, Maria da Conceição Barbosa Vieira Soares e José Antônio Martins de Mello pelas conversas, pela motivação e pelo entusiasmo com que conduziam (e tenho certeza ainda conduzem) as atividades no dia a dia do IEN.

Ao coordenador do PPGIEN Cláudio Márcio do Nascimento Abreu Pereira, pelas orientações e parceria.

Ao colaborador Adino, pela consultoria e auxílio com os sistemas do IEN.

Aos colegas de curso, com quem interagia diariamente e com os quais aprendi muito: Milton, Newton, André e Reinaldo

Ao diretor do Instituto de Engenharia Nuclear, Julio Cezar Suita.

Resumo da dissertação apresentada ao PPGIEN/CNEN como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc).

PROGRAMAS PARA VISUALIZAÇÃO DE SIMULAÇÕES COMPUTACIONAIS DE DINÂMICA DE FLUIDOS (CFD) USANDO OPENDX

Marcelo Mariano da Silva

JANEIRO DE 2008

Orientador: Paulo Augusto Berquó de Sampaio

Programa de Pós-Graduação Em Ciência e Tecnologia Nucleares do IEN.

A busca de soluções de hardware e software de alto desempenho e baixo custo sempre norteia os trabalhos no Laboratório de Computação Paralela do IEN. Nesse contexto, esse trabalho discorre sobre a construção de programas de visualização de simulações computacionais de dinâmica de fluidos (CFD) utilizando o software aberto OpenDx. Os programas desenvolvidos são úteis para a geração de imagens e vídeos em 2 e 3 dimensões, são interativos, de fácil utilização pelo usuário e visam atender às necessidades dos pesquisadores de dinâmica dos fluidos. No presente trabalho é feita uma descrição detalhada da construção dos programas, das instruções de uso e do OpenDx como ferramenta de desenvolvimento. As descrições acompanham exemplos que ajudam o leitor a compreender o alcance das aplicações desses programas.

Abstract of the thesis presented to PPGIEN/CNEN as partial fulfillment of the requirements for the degree of Master of Science (M.Sc).

OPENDX PROGRAMS FOR VISUALIZATION OF COMPUTATIONAL FLUID DYNAMICS (CFD) SIMULATIONS

Marcelo Mariano da Silva

JANUARY 2008

Advisor: Paulo Augusto Berquó de Sampaio

School: Programa de Pós-Graduação Em Ciência e Tecnologia Nucleares do IEN.

The search for high performance and low cost hardware and software solutions always guides the developments performed at the IEN parallel computing laboratory. In this context, this dissertation about *the building of programs for visualization of computational fluid dynamics (CFD) simulations using the open source software OpenDx* was written. The programs developed are useful to produce videos and images in two or three dimensions. They are interactive, easily to use and were designed to serve fluid dynamics researchers. A detailed description about how this programs were developed and the complete instructions of how to use them was done. The use of OpenDx as development tool is also introduced. There are examples that help the reader to understand how programs can be useful for many applications.

Sumário

Lista de Figuras

Capítulo 1

Introdução

1.1 Visualização Científica

Visualização científica é a representação gráfica de grandes quantidades de dados científicos que, da forma tradicional como são gerados, não podem ser analisados ou criticados pelo cientista. Os dados científicos são então transformados em imagens que condensam todas as informações relevantes de forma que estas possam ser facilmente captadas pelo sistema visual e assimiladas pelo cérebro humano.

"*A visão é o sentido humano que possui maior capacidade de captação de informações por unidade de tempo. O sentido da visão é rápido e paralelo, sendo possível, inclusive, prestar atenção em um objeto de interesse especial, sem perder de vista (obviamente, com menos detalhes) o que está acontecendo ao redor.(...) As visualizações por si só trazem benefícios, uma vez que podem funcionar como uma extensão da memória humana e como um auxílio para o processo cognitivo. Por exemplo (...) desenhamos diagramas e organizamos informações espacialmente em uma folha de papel quando estamos estudando um problema que envolve diversas partes. As imagens nos ajudam a entender o problema e/ou a encontrar uma solução para o mesmo. Elas, inclusive, facilitam a memorização do objeto em estudo.*" [1].

Chris Pelkie, da Universidade de Cornell, escreveu:

"*The development of written language numbers and letters as abstract symbolic representations or descriptors of reality is a much younger cultural development than is the biological evolution of our million-year old visual system. We are therefore attuned to interpret input through visual perception in quite different ways than we are to the "higher brain" rational symbol interpretation we perform when analyzing written text. Vision gives us direct perception of motion rates and directions, edge detection, stereoscopic depth cues, light and shade, material surface properties, and so on. While all of these parameters can be codified into numbers (as in a computer graphics system), it is virtually impossible for the unaided human mind to turn a huge mass of numbers into images containing the rich detail of a computer-generated image.*" [2]

O uso de ferramentas gráficas para a análise e tomada de decisões não é novidade e há tempos faz parte do dia a dia de cientistas e de outros profissionais. Esse processo é conhecido como visualização de informações e consiste "*no uso de representações*

visuais (de dados abstratos)(...) e interativas para ampliar a cognição" [3]. Nesse caso, os dados abstratos podem não possuir representação geométrica.

A visualização científica trata-se fundamentalmente do mesmo processo de emprego de representações visuais e interativas que permitem o entendimento e aprendizado. Contudo, diferentemente da visualização de informações, as imagens, nesse caso, possuem representação geométrica e estão relacionadas a um domínio espacial.

O emprego da visualização científica é crescente devido a uma série de fatores, dentre os quais podemos citar:

1. permitir melhores análises e interpretação dos dados;
2. barateamento de processadores e placas gráficas ;
3. facilidade no armazenamento de grandes quantidades de dados;
4. disponibilidade de bibliotecas gráficas;

Os sistemas de visualização modernos, que constituem o estado da arte, apresentam as seguintes características:

1. **são interativos:** a interação entre o usuário com o conjunto de dados visualizados é simples. Modifica-se facilmente o programa e implementa-se ajustes que facilitam a modelagem dos dados de acordo com as necessidades do cientista/pesquisador.
2. **são modulares:** são de fácil programação, sem a necessidade de especialistas intermediando o desenvolvimento do aplicativo. Essa característica favorece o cientista, que tem a percepção crítica do fenômeno em simulação.
3. **são compartilháveis e adaptáveis a diferentes aplicações.**

1.1.1 Programas de Visualização Científica

Ferramentas de apresentação gráfica de resultados, oriundos de simulações ou experimentos, são utilizados por cientistas desde os primórdios da computação gráfica, por se tratar de uma linguagem sintética e facilmente assimilável [4]. Em CFD, *softwares* de visualização são recursos de particular importância para a análise de dados. Tão relevante quanto o método numérico empregado, a visualização é fundamental na interpretação e validação dos resultados [5].

Classificação das ferramentas de visualização em termos evolutivos

Em termos evolutivos, as ferramentas de visualização científica são assim classificadas [4]:

Bibliotecas Gráficas Genéricas (BGG): Eram bibliotecas criadas pelos fabricantes de equipamentos gráficos, terminais, impressoras etc, para gerar as primitivas gráficas de seus dispositivos. Essas bibliotecas eram utilizadas pelos pesquisadores para a visualização de seus dados. Os códigos escritos eram pouco portáteis, pouco interativos e muitas vezes as funções da biblioteca não atendiam a necessidade de elementos gráficos de aplicação.

Bibliotecas Gráficas Específicas (BGE): Essas bibliotecas traziam as primitivas gráficas para o tratamento de elementos gráficos, o que foi uma evolução. Contudo, eram desenvolvidas para aplicações específicas para áreas específicas. Os programas tinham de ser escritos e a interatividade era restrita.

Sistemas de Visualização Específicos (SVE): Para solucionar os problemas de pouca interatividade das BGEs, foram desenvolvidas as SVE's, que traziam recursos de interação. Todavia, as SVE's continuavam atendendo a aplicações específicas de determinadas áreas.

Sistemas de Visualização Genéricos (SVG): Os SVGs surgiram da necessidade de fazer das SVE's ferramentas mais gerais, aplicáveis a diferentes áreas e com mecanismos mais simples de implementação. Funções complexas puderam ser encapsuladas, facilitando sua utilização por leigos em programação.

Os SVGs podem ser divididos em classes:

1. sistemas de visualização genéricos por fluxo de dados (SVG_FDs);
2. sistemas de visualização genéricos orientados a objetos (SVG_OOs);
3. sistemas de visualização genéricos por mapeamento (SVG_Ms);

Sistemas de visualização genéricos por fluxo de dados (SVG_FDs) Conhecidos por sistemas *data flow*, são sistemas que promovem transformação dos dados a partir do arranjo de módulos de cálculo e visualização (unidades de transformação básica do sistema). Estão nessa classe os seguintes programas: *AVS , apE , Iris-Explorer, IBM-Data Explorer* e *Khoros*.

Sistemas de visualização genéricos orientados a objetos (SVG_OOs) Os sistemas orientados a objetos utilizam-se de modelos de dados definidos em classes que podem ser construídas pelo programador. Os recursos de herança e polimorfismo oferecidos por esse tipo de linguagem (orientado a objetos) potencializam sua utilização. A arquitetura orientada para objetos permite a geração de código flexível, de fácil manutenção, extensível e reutilizável. Exemplo: *VTK (Visualization Toolkit)*

Sistemas de visualização genéricos por mapeamento (SVG_Ms) Destinam-se à edição, compilação, depuração e execução de algoritmos de visualização científicos. Os sistemas de mapeamento utilizam-se de uma função de exibição que mapeia os elementos de um modelo de dados e cria elementos de um modelo de exibição. Desta forma, esses sistemas podem exibir os dados internos de um algoritmo sem acionar rotinas gráficas que poderiam exigir um tipo específico de dado. Esse conceito torna a ferramenta interativa, uma vez que essa disponibiliza estruturas de referência editadas em janela de texto, que regulam o mapeamento da função de exibição permitindo execução por passos, combinação de vários dados, geração de animações.

Classificações dos Visualizadores

Baseando-se nos recursos que são oferecidos para configuração da interface, das funcionalidades estruturais, funções gráficas e gerenciamento de dados, os programas de visualização podem ser separados em dois grupos [6]:

Sistemas tipo "usuário-final" (*turnkey*) São sistemas que não oferecem recursos para a configuração da interface e de sua estrutura. Os *softwares* comerciais se enquadram nessa classificação. São utilizados por especialistas numéricos, treinados para o uso da ferramenta. Esses programas podem ser incapazes de modelar determinados problemas e não é possível reconfigurá-los para tanto [7]. Cita-se como exemplos: *Flow Analysis Software Toolkit - FAST* (NASA,comercial), *FILDVIEW* (Intelligent Light, comercial), *XDATASLICE* (University of Illinois, gratuito).

Sistemas tipo "Geradores de aplicação" (*Application Builders*) Permitem a configuração do sistema de visualização. Podem utilizar de códigos pré-programados em bibliotecas ou de códigos implementados pelo próprio usuário, o que confere flexibilidade às suas aplicações. São exemplos de *Application Builders* : *VTK, Íris Explorer, IBM Data Explorer, AVS, apE* e *Khoros.*

Descrição dos Principais Visualizadores

Segue abaixo uma relação dos principais visualizadores utilizados no meio científico e suas principais características:

VTK O VTK é uma biblioteca de algoritmos para computação gráfica que também pode ser utilizado como um *Application Builder.* É um SVG_OO que foi desenvolvido com código aberto e pode ser utilizado para a visualização e processamento de imagens. A linguagem utilizada em seu desenvolvimento é a C++ mas pode ser utilizado a partir de linguagem Java e Tcl/Tk . O VTK serve de base a dezenas de pós-processadores customizados que são comercializados. É uma ferramenta versátil com boa portabilidade. Muitas implementações em CFD têm utilizado o VTK.

IRIS Explorer O IRIS é um software comercial de visualização, versátil, que opera em Unix, Linux e Windows. É um SVG_FD que utiliza de bibliotecas gráficas *Open Inventor*, *ImageVision* e *OpenGL* além das desenvolvidas pela NAG (*Numerical Algorithms Group*) proprietária dos direitos de uso. O IRIS possui um *Map Editor* (Editor de Mapas) no qual são construídos os mapas (programas) pelo arranjo de módulos armazenados num outro módulo, a LIBRARIAN. Os módulos são construídos em linguagem C ou FORTRAN. Caso seja necessário customizar uma aplicação, o IRIS oferece um *Module Builder* para a construção de módulos que não estão disponíveis no LIBRARIAN. Outro utilitário disponível é o *DataScribe* que serve para conversão de dados do IRIS para outros formatos.

AVS O *Application Visualization System* (AVS) é um pacote de visualização e processamento de imagens que, da mesma forma que o IRIS, utiliza-se de módulos que são conectados para a construção de uma aplicação (é um SVG_FD). Permite a produção de animações e geração de malhas. É portátil e expansível. O AVS é um programa comercial dividido em quatro módulos [3]:

- **Image Viewer** - ferramenta de manipulação e visualização de imagens;
- **Graph Viewer** - ferramenta de processamento gráfico;
- **Geometry Viewer** - promove a geração dos dados na tela a partir da descrição computacional. Possui uma interface para a manipulação dos objetos, que podem ser movidos, rotacionados ou dimensionados;

- **Network Editor** - interface para programação visual, onde os módulos são conectados e os aplicativos construídos;

apE O apE (*animation production Environment*) é um sistema muito semelhante ao AVS, desenvolvido na Universidade Estatual de Ohio. Atualmente esse *software* é comercializado pela *TaraVisual*. Possui módulos que são conectados por *pipelines* e ferramentas de interação com imagens [1].

Khoros Comercializado atualmente pela *Khoral*, foi criado na Universidade do Novo México para ser executado em sistemas Unix e era inicialmente distribuído gratuitamente. É um SVG_FD que possui um ambiente de programação visual , semelhante ao AVS. É portátil e extensível [1].

IBM -Data Explorer OpenDX O OpenDx é um SVG_FD essencialmente de visualização muito utilizado no meio científico. Foi criado em 1991 pela IBM e seu código tornou-se aberto em 1999 sendo, a partir de então, aperfeiçoado por desenvolvedores de todo o mundo. O OpenDX possui um *Visual Program Editor* (VPE) no qual módulos polimórficos (que encapsulam funções programadas) são conectados através de *pipelines* através de suas entradas e saídas. O resultado é um *script* pelo qual os dados fluem e as imagens são geradas. Módulos customizados podem ser desenvolvidos em C++, o que torna a ferramenta flexível e justifica sua aplicação em diferentes áreas de pesquisa científica que empregam *softwares* de visualização. O OpenDX possui inúmeras ferramentas de interação com as imagens, diretos (*zoom*, rotação) e indiretos (*dials*, *switches*, botões, *sliders*), essenciais no estudo visual dos resultados obtidos. Além de interação com as imagens, essa ferramenta permite a geração de animações em formato *mpeg* que podem ser utilizadas para melhor avaliação de resultados experimentais ou simulacionais. É portátil, extensível e prático para a construção de aplicativos de visualização. O formato padrão dos arquivos processados no OpenDx é o data explorer file (*.dx*), mas esse programa também importa arquivos nos formatos HDF, CDF e *Spreadsheet* [6] [8].

1.2 Objetivo do trabalho

A filosofia adotada no Laboratório de Computação Paralela do IEN (LCP/IEN) é a de buscar soluções de *hardware* e de *software*, com alto desempenho e baixo custo de aquisição e manutenção, voltados para o tratamento de problemas da engenharia, em especial da engenharia nuclear. O objetivo do presente trabalho é desenvolver programas para visualização de simulações computacionais de dinâmica de fluidos (CFD), utilizando *software* aberto, que possam servir tanto aos trabalhos que encontram-se em andamento no LCP/IEN quanto em outros centros de pesquisa.

Os programas de visualização foram desenvolvidos para gerar imagens de simulações que empregam o método de elementos finitos. Neste caso, a simulação dinâmica de escoamento de fluidos se dá através da discretização de um domínio contínuo sobre o qual se pretende calcular o campo de uma grandeza (vetorial ou escalar). Na discretização por elementos finitos, o domínio é dividido em pequenas células (elementos), geralmente triangulares (simulação em 2 dimensões) ou tetraédricos (simulações em 3 dimensões) formando uma malha ou grade. Os cálculos sobre os pontos da malha (nós) ou sobre as células são feitos por algoritmos que utilizam as equações de *Navier-Stokes*. Entre os nós da malha, nos quais a grandeza é efetivamente calculada, o intervalo do

domínio é contínuo e por isso se faz necessário o uso de técnicas de interpolação para se determinar o campo da grandeza simulada (vetorial/escalar).

As simulações em CFD que utilizam o método de elementos finitos são divididas em três etapas, que são descritas na seqüência:

1. **Pré-processamento**

   Nessa etapa, são determinados os domínios e a geometria do problema que será simulado. As condições de contorno e as condições iniciais são introduzidas. É nessa etapa que há a geração da primeira malha com a discretização do domínio. No LCP/IEN, até o momento, o *software* GID vem sendo utilizado no pré-processamento. O GID é um *software* proprietário que fornece uma interface gráfica para modelagem geométrica, facilitando a entrada de dados em programas de simulação numérica.

2. **Processamento**

   O processamento é feito utilizando-se de um código computacional desenvolvido por *De Sampaio* [17] [18] [19] em linguagem *Fortran 90*. O processador emprega as equações de *Navier-Stokes* no domínio discretizado, sobre os nós e elementos da malha, computando a evolução, passo a passo, de grandezas de interesse em mecânica dos fluidos.

3. **Pós-processamento**

   Processados os valores na etapa anterior, os resultados são impressos em arquivos para serem utilizados por um pós-processador, que deverá gerar as imagens para a análise do pesquisador. Os pós-processadores utilizados no momento são o GID e os programas de visualização desenvolvidos com a ferramenta OpenDX (*software* livre).

A necessidade de geração de imagens mais robustas, de recursos para a edição, manipulação e transformação das imagens geradas e principalmente de produção de animações e vídeos a partir dos resultados do processamento foi o que norteou e fundamentou o desenvolvimento dos programas de visualização (pós-processadores) desenvolvidos com o OpenDX.

No início de 2005, o emprego do OpenDX como ferramenta para a construção de pós-processadores foi introduzido no IEN por *Guimarães* e *De Sampaio* [20], que construíram os primeiros programas.

A necessidade de tornar os pós-processadores construídos no OpenDx mais robustos e fáceis de utilizar, sob a ótica do usuário, culminou no desenvolvimento desse trabalho, cuja contribuição foi o aprimoramento desses programas, promovendo-se a correção de problemas de programação, melhorando-se a interação do usuário com as diferentes funcionalidades oferecidas, introduzindo-se recursos de manipulação/edição de imagens e construindo-se mecanismo para geração de arquivos de animação automaticamente e concomitantemente com a geração das imagens. Foi construída também, dentro do processador, uma subrotina em Fortran 90 para a impressão direta dos arquivos utilizados pelo pós-processador para a geração de imagens. Todas essas melhorias tornaram os programas de visualização construídos no OpenDX pós-processadores de poderosa e fácil utilização que contribuem para a melhoria das análises nos trabalhos desenvolvidos em CFD.

1.2.1 Escolha do OpenDX para construção do pós-processador

O uso do OpenDx para a construção de pós-processadores que atendessem a necessidade de geração de imagens e animações a partir dos resultados das simulações numéricas em CFD foi uma decisão acertada, uma vez que comparativamente às demais ferramentas disponíveis, o OpenDX é o único que possui todas as características desejadas: trata-se de uma ferramenta *open source*, portábil, interativa e extensível. Além disso, essa ferramenta oferece recursos para implementação de diversas funcionalidades graças a um rico conjunto de rotinas disponíveis ao programador. A possibilidade de gerar animações e de implementar dispositivos de interação e controle da simulação são diferenciais adicionais dessa ferramenta.

Sendo o OpenDx então a melhor opção, melhorias e novas funcionalidades foram agregadas aos programas já existentes e em uso no LCP/IEN, resultando nas novas versões de visualizadores. Foram costruídos dois visualizadores: VISUALIZADOR_2D_NS para geração de imagens em 2 dimensões e o VISUALIZADOR_3D_NS para geração de imagens em 3 dimensões.

As versões anteriores já possuíam os seguintes recursos:

- Permitiam a visualização através de mapas de cores (variação de cores em função do gradiente do campo escalar) de variáveis importantes nas análises de mecânica de fluidos : temperatura, pressão, módulo da velocidade, e módulo do fluxo de calor. Nas imagens em três dimensões, permitiam a visualização do campo escalar dentro do volume (campo escalar translúcido - "efeito gelatina").

- Permitiam a visualização dos campos vetoriais de variáveis importantes como velocidade e fluxo de calor. Nesse caso, os vetores ainda eram coloridos de acordo com o gradiente de intensidade dos mesmos, como acontece com os módulos nos mapas de cores.

- Permitiam ao usuário a visualização da malha de elementos finitos que foi aplicada e o domínio espacial da simulação.

- Traziam opções ao usuário de combinações de diferentes modos de visualização: apresentação concomitante da variável de simulação (mapa de cores) e da malha; do domínio espacial e malha, etc.

As novas versões possuem funcionalidades adicionais. As melhorias e contribuições introduzidas são relacionadas a seguir:

- Os arquivos numéricos para a geração das imagens são processados e gerados automaticamente pelo processador e disponibilizados em diretório especificado para acesso do pós-processador.

- Permitem ao usuário especificar em que diretório o programa de visualização deverá buscar os arquivos para posterior exibição de imagens.

- Permitem que o usuário utilize o programa sem ter de editar seu código fonte.

- Favorecem a interação do usuário com a imagem, permitindo ampliações e reduções (*zoom in* e *zoom out*), rotações (*rotate*) ou deslocamentos (*move*).

- Possuem a funcionalidade de geração de linhas de corrente. O usuário pode determinar os pontos de origem dessas linhas (linhas de corrente são linhas que descrevem a trajetória de partículas arrastadas pelo escoamento do fluido).

- Fornecem ao usuário a ferramenta para a geração automática das animações e dos vídeos para exibição das simulações em qualquer vídeo *player*.
- Os programas tornaram-se flexíveis à introdução de novas variáveis e combinações de diferentes modos de visualização.
- Foram criados controles de execução nos programas.
- Foram elaboradas interfaces mais amigáveis para interação do usuário com a ferramenta.

1.3 Organização do trabalho

Nesse capítulo foi feita uma apresentação desse trabalho, seus objetivos e o contexto em que está inserido seu desenvolvimento. No capítulo 2, busca-se apresentar o OpenDx como ferramenta para elaboração de programas. Descrevem-se detalhadamente seus recursos, desde a estrutura dos dados de entrada (*input*), passando pelas formas possíveis de construção dos programas até a produção da imagem final para visualização. No capítulo 3 é feita uma apresentação da lógica de construção dos programas de visualização científica que são o propósito desse trabalho: VISUALIZADOR_2D_NS para geração de imagens em 2 dimensões e o VISUALIZADOR_3D_NS para geração de imagens em 3 dimensões. Instruções sobre o uso dos programas são dadas no capítulo 4. As conclusões, observações e apontamentos sobre melhorias que podem ser implementadas numa segunda versão desses programas são feitos no capítulo 5.

Capítulo 2

OpenDX (IBM Data Explorer)

O OpenDX é um programa de visualização científica que foi idealizado pela *International Business Machines Corporation - IBM* no final dos anos 80 e desenvolvido no *T.J. Watson Research Center*, no estado de Nova York. Pesquisadores do *The Cornell Theory Center* também participaram dos aperfeiçoamentos desse *software* que passou a ser comercializado em 1991. Em 1999 o OpenDX tornou-se um *software* livre (open source) e assim seu desenvolvimento e aperfeiçoamento passou a ser feito por inúmeros voluntários de todas as partes do planeta. Novas versões e customizações são desenvolvidas e compartilhadas diariamente entre os usuários dessa ferramenta.

2.1 Características

O OpenDx é um *software* de visualização científica utilizado em vários ramos da ciência. Essa ferramenta não traz módulos de computação matemática específicos (para aplicações específicas, como por exemplo para CFD), embora esses módulos possam ser desenvolvidos e incorporados aos programas. Dessa forma, ele funciona como um visualizador e um "construtor" gráfico dos dados processados por programas numéricos.

O OpenDX é um SVG_FD, ou seja um sistema *data-flow*. Nesses sistemas, os dados atravessam uma série de módulos (nós) ligados seqüencialmente ou paralelamente onde sofrem processamentos e alterações. Cada módulo recebe os dados, atua sobre estes segundo a subrotina que encapsula e repassa esse dados para módulo seguinte, até que um *output* seja determinado.

Existem muitos módulos que podem ser utilizados para visualização e construção de imagens, em duas ou três dimensões. Pode-se, por exemplo, construir/visualizar superfícies de nível, gradientes através de uma dispersão de cores ou traçar percursos de partículas em campos de velocidade. Campos vetoriais são graficamente representados por vetores orientados de tamanhos e cores proporcionais a suas intensidades. Há ainda subrotinas que permitem a manipulação da imagens, oferecendo recursos de ampliação (*zoom*), rotação (*rotate*) e deslocamento (*move*) da imagem.

O OpenDx pode ser utilizado de três formas diferentes, de acordo com os objetivos e necessidades do usuário:

1. Como um *software* final, para simples utilização por usuários que o desconhecem e só estão interessados em gerar visualizações e animações.

2. Como um *application builder* gráfico, em que o usuário se utiliza de uma interface gráfica para desenvolver uma lógica e criar uma aplicativo para a geração de visualizações e animações de acordo com suas necessidades. Nesse caso, o usuário faz um encadeamento de módulos que encapsulam subrotinas através de uma interface gráfica que facilita esse trabalho. As conexões desses módulos resultam num aplicativo customizado para uma determinada aplicação.

3. Como um *application builder* não gráfico. Nesse caso, o usuário pode criar os módulos para serem utilizados na interface gráfica de programação ou programar diretamente em linha de comando, acessando as bibliotecas gráficas e as subrotinas.

2.2 Funcionamento

De uma forma sintética, o funcionamento do OpenDX pode ser representado pelo diagrama na figura 2.1 :

Figura 2.1: Esquema sintético de funcionamento do OpenDX

Três etapas podem se identificadas:

1. **Entrada de dados :** os dados de *input* seguem as convenções do *Data Explorer (Data Model)*;

2. **Programa de visualização (*Visualization Program*):** programa que gera e processa as imagens a partir dos dados de entrada. Esse programa pode ser desenvolvido na interface gráfica (VPE - *Visual Program Editor*), utilizando-se os módulos disponíveis ou customizados - estes últimos produzidos através do *Module Builder* (aplicativo para construção de módulos). Caso deseje, o usuário pode desenvolver o programa através de *scripting language*. Nesse caso não há interface gráfica e a programação se dá de forma tradicional.

3. **Imagem (apresentação do resultado)** : A imagem gerada é disponibilizada numa janela, que pode ser a *Image Window* ou o *Display*. Um ou outro pode ser escolhido como *output* da imagem durante a elaboração do programa de visualização. O *Image Window* traz uma série de recursos implementados para a manipulação da imagem. Contudo, dependendo da lógica empregada na elaboração do fluxo de dados no programa de visualização, pode ser que esses recursos não funcionem a contento. Quando isso acontece estes recursos de interação precisam ser construídos a partir de módulos disponíveis e nestes casos é indicada a utilização do *Display*. Dependendo das interações possíveis com a imagem

ou das funcionalidades implementadas no programa de visualização (possibilidade de geração de animações por exemplo), alguns painéis de controle (*Control panels*) ficam disponíveis para que o usuário interaja com a imagem através de botões, barras de rolagem ou *dials* .

A seguir, com o objetivo de aprofundar um pouco mais o entendimento do funcionamento do OpenDX, detalha-se um pouco mais cada uma das etapas da geração de uma imagem no programa.

2.2.1 Entrada de Dados

A entrada de dados segue as convenções do *Data Model* do *IBM Data Explorer*. O *Data Model* do OpenDX associa dados com pontos ou com conexões entre esses pontos, que são determinados sobre o domínio dos dados. Os dados podem ser:

1. Reais ou complexos;
2. Escalares, vetoriais ou tensoriais;
3. Inteiro, flutuante, byte;

Os pontos e/ou suas conexões definem ou não uma estrutura de dados sobre o domínio, formando ou não uma malha. Quando existe a malha, essa pode ser:

1. Regular e ortogonal;
2. Regular e curvilínea;
3. Irregular (conexões de triângulos, quadriláteros, tetraedros de diferentes tamanhos);

Quando não existe malha, a distribuição dos pontos pelo domínio é desestruturada (sem conexões entre os pontos).

Os dados são armazenados na forma de objetos para que sejam utilizados pelos módulos. Os objetos são estruturas de dados armazenados na memória que guardam o tipo de objeto e de dependências desse objeto.

Principais tipos de objetos

Objeto tipo *Field*: Relaciona o dado com a posição ou uma conexão entre duas posições. Especifica, desta forma, o dado que é dependente de um conjunto de posições ou das conexões entre essas posições. Entre um ponto e outro, os dados são obtidos por interpolação. Já os dados relacionados às conexões, são considerados constantes. Se não houver conexões entre os pontos (ausência de malha/distribuição desestruturada), o domínio passa a ser discreto e só são associados dados para os pontos. Um objeto *Field* é composto por um conjunto de componentes, cada qual definido por um objeto, que geralmente é do tipo *Array*.

Na figura 2.2 temos uma representação visual de um objeto *Field* que possui quatro componentes. Cada componente é uma objeto do tipo *Array* que armazena uma estrutura de dados. Os componentes de um objeto *Field* possuem atributos associados a eles. Por exemplo, o atributo *dep* estabelece uma relação entre uma grandeza escalar ou vetorial e um "ente" geométrico, como um ponto ou conexão entre pontos. No exemplo da figura 2.2, os valores de *dado de temperatura* estão relacionados (têm

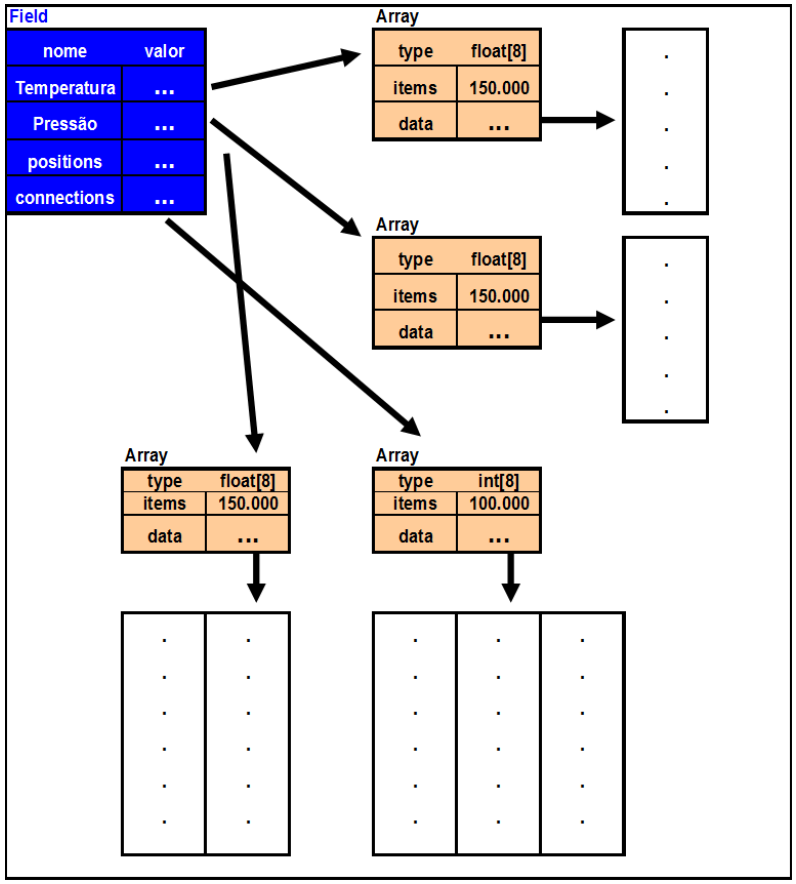

Figura 2.2: Representação visual de um objeto *Field* que possui quatro componentes

"dependência") aos pontos, denominados como *positions*. Um ponto é definido através de duas coordenadas e de um "nome", que é um número único que permitem sua identificação. O atributo *ref*, serve para agrupar ou estabelecer relações entre elementos geométricos, sem necessariamente relacioná-los à alguma grandeza. Por exemplo, quando é preciso agrupar pontos para se definir um triângulo numa malha de elementos finitos bidimensional, faz-se uma "referênciação" ao conjunto de pontos (através de seus "nomes") que reunidos formam o triângulo. Outro atributo é *element type*, que estabelece como devem ser as conexões entre os pontos: triangulares, quadradas, tetraédricas etc.

Mesmos componentes podem ser compartilhados por diferentes objetos do tipo *Field*. Desta forma, esses objetos (*Field*) podem compartilhar os mesmos componentes de posições e conexões embora atribuam aos mesmos dados diferentes. Isso é possível porque os componentes são objetos *Arrays* que encerram neles os dados (figura 2.3).

Uma relação dos componentes que podem fazer parte de um objeto *Field* está na página 19 da referência [9]. Uma relação dos atributos dos componentes está na página 25 da referência [9].

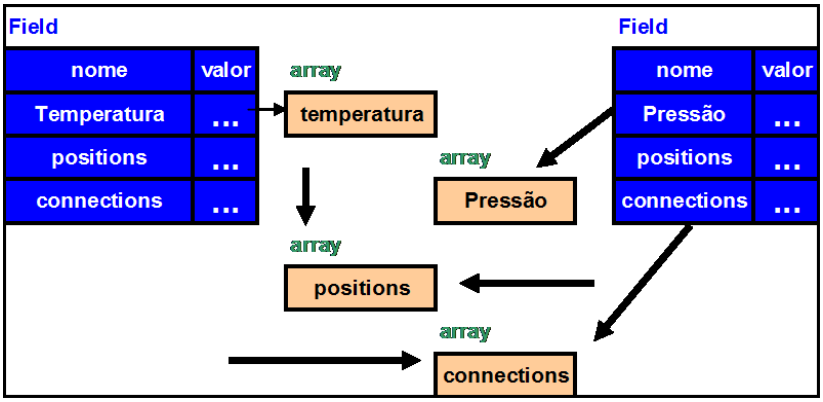

Figura 2.3: Componentes sendo compartilhados por diferentes objetos do tipo Field.

Objeto tipo Array: Objetos que reservam os dados, as posições (coordenadas), as conexões e outras informações relativas a outros componentes, os *arrays* possuem parâmetros que devem ser definidos. Esse parâmetros estão relacionados a seguir:

Type - Tipo de dado que será acumulado no *array* : *double, float, int, uint, short, ushort, byte, ubyte ou string*. O tipo *uint* por exemplo é o inteiro sem sinal (*unsigned int*).

Category - Especifica se o dado será Real ou Complexo.

Rank - Especifica como será o arquivo de dados.
Rank 0 define um arquivo com uma *string* de valores, que são organizados em uma coluna, com um valor por linha (linha a linha). Arquivos parametrizados como *Rank 0* são utilizados para representação de grandezas escalares.
Rank 1 define um arquivo com mais de uma *string* de valores, que são distribuídos em colunas. Nesse caso, uma linha conterá dois ou mais valores que agrupados constituem um vetor de dados. Esses vetores de dados armazenam coordenadas de pontos, pontos de um elemento da malha (por exemplo os pontos que identificam os vértices de um triângulo numa malha triangular) ou o tamanho das componentes de um vetor físico na representação de grandezas vetoriais. Portanto, a ordem dos valores em cada linha (vetor de dados) é importante;
Rank 2 define um arquivo com estrutura matricial, para representação de matrizes e tensores;
Higher Ranks define estruturas para a representação de tensores de ordens superiores.

Shape - Define as dimensões da estrutura.
Se *Rank 0*, *Shape* não existirá, porque uma única *string* de valores será gravada no arquivo;
Se *Rank 1*, *Shape* definirá o número de dimensões (*strings*) da estrutura de dados, ou em outras palavras, o número de componentes do vetor de dados (= 2, 3 ou 4).
Se *Rank 2*, shape definirá a estrutura da matriz ou tensor de dados. Por exemplo *Shape 3x3* definirá uma matriz ou tensor de 3 linhas por 3 colunas.

Objeto tipo *Group*: Um objeto *Group* agrupa objetos tipo *Field* e/ou outros objetos tipo *Group*. Cada objeto que faz parte do *Group* é denominado membro. Na figura 2.4 é apresentado um exemplo de *Group*.

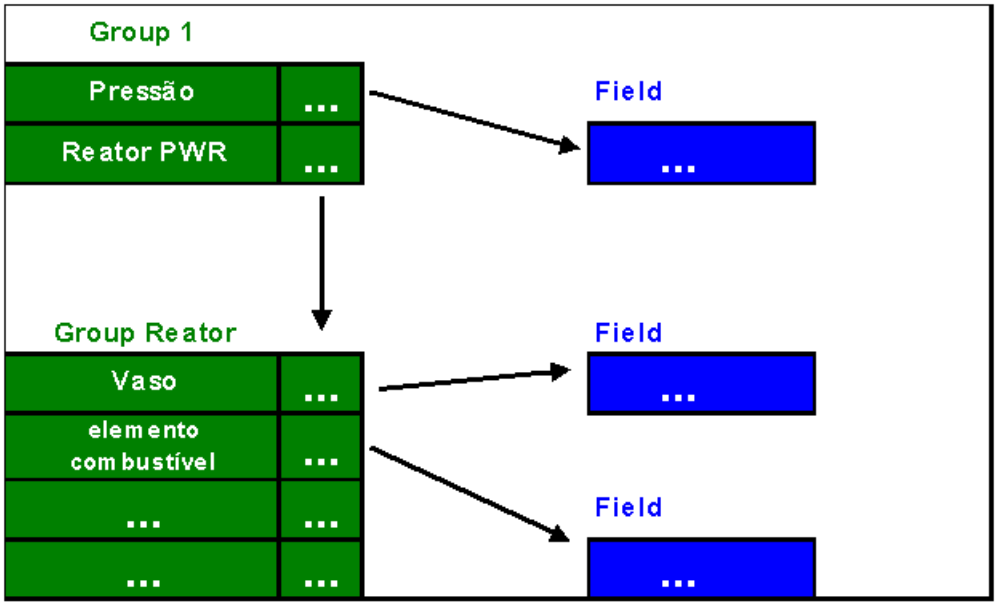

Figura 2.4: Exemplo de Grupo.

O *Group 1* do exemplo da figura 2.4 possui dois membros: um *Field*, cujo nome é Pressão e outro *Group*, cujo nome é *Reator PWR*. O *Field* pressão acumula os valores de pressão ao longo do volume do reator. O membro *Reator PWR*, que é um grupo, acumula a descrição geométrica do reator nuclear. Esse grupo possui vários *Fields* como membros, cada um com a descrição geométrica de uma parte do reator (vaso, elementos combustíveis etc). Essa constitui uma forma de organização dos dados que facilita a produção da imagem da reator como um todo e a visualização do campo de pressão ao logo de sua estrutura.

Geração de arquivos para pós-processamento com OpenDX

Como foi explicado no Capítulo 1 (Seção 1.2), o processador deve imprimir os arquivos que serão utilizados pelo pós-processador, no caso os programas desenvolvidos no OpenDX, para a geração de imagens e animações. Esses arquivos possuem um formato definido que deve ser considerado pelo programa do processador que o imprimirá.

Para a geração de imagens por programas no OpenDx são necessários no mínimo 3 arquivos:

1. **arquivo de coordenadas (X,Y se bidimensional ou X,Y,Z se tridimensional)**: são as coordenadas dos pontos que serão os vértices dos elementos geométricos que particionarão o domínio . A figura 2.5 é um exemplo de arquivos de coordenadas processados por programas no OpenDX em 2D. A figura 2.6 é outro exemplo, em 3D.

20.00000	15.00000
19.80000	15.00000
20.00000	14.79954
19.82403	14.69668
19.60000	15.00000
20.00000	14.59908
19.62557	14.83093
19.82574	14.49668
19.65169	14.59520
19.40000	15.00000
20.00000	14.39862
19.47767	14.69367
19.65276	14.39489
19.30553	14.79924
19.82709	14.29651
19.47932	14.49387
19.20000	15.00000
20.00000	14.19816
19.30580	14.59177
19.47979	14.29510

Figura 2.5: Arquivo de coordenadas em 2D.

2. **arquivo de elementos**: o domínio é dividido em elementos geométricos, e cada vértice desses elementos recebe um número. Esse números, armazenados e ordenados, permitem a identificação de cada elemento. Por exemplo, podemos dividir um domínio bidimensional em vários triângulos. Cada vértice é numerado de tal forma que uma trinca ordenada permite sempre que se identifique cada um dos triângulos.

   Na figura 2.7, um domínio foi dividido em 3 triângulos, A, B e C. Cada um deles pode ser identificado pela reunião dos números que representam seus vértices (por convenção, a numeração segue o sentido horário):

   Triângulo A = 1,3,4

   Triângulo B = 1,5,2

   Triângulo C = 2,5,4

   Logo, um arquivo de elementos trará os grupos numéricos que permitam identificar cada elemento geométrico. A figura 2.8 é um exemplo de arquivo em 2D para elementos geométricos do tipo triangular. A figura 2.9 é um exemplo de arquivo 3D para elementos do tipo tetraédrico.

3. **arquivo de dados:** são os arquivos com os dados das variáveis (vetorial ou escalar) considerados na simulação.

   Esses arquivos, diferentemente dos arquivos de coordenadas e de elementos, devem possuir um cabeçalho, um corpo e um rodapé, cada qual armazenando um conjunto específico de informações. Para facilitar a compreensão de como esses arquivos são organizados, é apresentado um exemplo na figura 2.10.

   Em relação ao exemplo da figura 2.10:

   O arquivo possui três objetos (*objects*) que são relacionados para a construção do *Field* de dados.

-0.09777	0.49035	20.00000
-0.19156	0.46185	20.00000
0.00000	0.50000	20.00000
-0.15375	0.47577	19.91268
-0.11619	0.38239	20.00000
0.00000	0.50000	19.90000
-0.08005	0.49355	19.85000
0.03911	0.39799	20.00000
-0.10413	0.37020	19.85731
-0.23923	0.43905	19.88523
-0.27795	0.41563	20.00000
0.09732	0.49044	20.00000
-0.16023	0.47363	19.80750
0.00000	0.50000	19.80000
0.08454	0.49280	19.86170
-0.25368	0.30887	20.00000
-0.08005	0.49355	19.75000
-0.31725	0.38646	19.91340
-0.12347	0.23070	20.00000
0.10226	0.37588	19.86356

Figura 2.6: Arquivo de coordenadas em 3D.

Figura 2.7: Partição do domínio em triângulos.

O *cabeçalho* possui dois objetos, o *Object 1* de coordenadas dos pontos e o *Object 2* dos elementos geométricos (células) da malha de simulação.

Object 1 é a maneira que o objeto que contém as coordenadas é referenciado. Sempre que "*value 1*" for relacionado, se estará fazendo uma referência ao objeto 1, nesse caso ao arquivo de coordenadas. Da mesma forma, "*value 2*" faz referência ao *Object 2*, que relaciona os pontos que definem cada elemento geométrico da malha de simulação.

O *Object 3*, que não pertence ao cabeçalho, é relacionado no corpo do arquivo, designado por *Dados*. Esse objeto traz os valores da grandeza que será simulada. Sempre que "*value 3*" for relacionado, se estará fazendo uma referência ao objeto 3.

Os três objetos são *Class Array* , o que indica que são objetos do tipo *array*.

Os objetos 1 e 3 possuem dados do tipo flutuante (*type float*). O objeto 2 é *type int*, porque seus dados são do tipo inteiro.

11746	11734	11759
12343	12359	12357
12098	12081	12109
11881	11858	11891
11624	11610	11642
12236	12223	12247
11683	11670	11697
12301	12285	12309
12024	12007	12034
11814	11797	11824
11575	11594	11560
12173	12151	12183
11716	11699	11736
12331	12314	12337
12061	12042	12068
11847	11827	11856
11596	11583	11606
12204	12188	12214
11656	11638	11661
12271	12256	12280

Figura 2.8: Arquivo em 2D para elementos geométricos do tipo triangular.

Rank determina o tipo de estrutura de dados que será empregado e *shape* define a dimensão da estrutura de dados (consulte item 2.2.1, *Principais tipos de objetos*, *Objetos tipo array*). Nos objetos 1 e 2 da fig 2.10, *rank=1*, ou seja, os dados serão dispostos em mais de uma *string* de dados (*strings* são dispostas em colunas de dados). Cada linha conterá dois ou mais valores que constituirão um vetor de dados. Essa quantidade, de dois ou mais, é definido pelo parâmero *shape*. Logo o objeto 1 possuirá 2 strings ou dois dados em cada um de seus vetores de dados (dispostos em cada linha). Da mesma forma, o objeto 2 possuirá três *strings* pois seu *shape* é 3 ou em outras palavras, cada vetor de dados possuirá 3 dados (são 3 dados em cada linha).

O objeto 3 possui *Rank*=0, o que quer dizer que só conterá uma *string* (coluna) de dados. Se *rank* é 0 *shape* é 1 por definição, por isso não é necessário declará-lo no arquivo.

Items é a quantidade de dados que serão relacionados em cada objeto.

Nos arquivos de extensão *dx* utilizados pelo OpenDx, os dados devem ser relacionados às posições ou conexões, para que os programas visuais possam processá-los. Essa referenciação se dá através da declaração de atributos dos dados.

Os objetos 1 e 3 usam atributos de "dependência" expresso pela linha

attribute "dep" string "positions".

No objeto 1, esse atributo estabelece uma relação *de um para um* entre vetores de dados de coordenadas e os pontos. Os pontos, designados por *positions*, passam a ser vinculados às suas coordenadas. Já no objeto 3, esse atributo estabelece uma relação entre um dado(valor) e um ponto, também numa correspondência de um para um.

O objeto 2 usa um atributo de "referência", expresso pela linha

2599	2572	2545	2566
2545	2572	2516	2566
10646	10625	10678	10629
10091	10070	10038	10052
10075	10058	10018	10055
10176	10147	10124	10151
10131	10110	10158	10126
4102	4076	4046	4084
10131	10102	10078	10126
2495	2471	2527	2487
6350	6376	6404	6382
3972	3943	3996	3960
10147	10119	10093	10151
6374	6402	6429	6392
4093	4118	4146	4107
4046	4076	4024	4084
4146	4118	4173	4107
6350	6323	6376	6382
10734	10736	10729	10677
6374	6348	6402	6392

Figura 2.9: Arquivo em 3D, para elementos do tipo tetraédrico.

attribute "ref" string "positions"

Nesse caso, o atributo estabelece uma relação entre os pontos, designados por números que os nomeiam. Não há uma correspondência *de um para um* entre valores e pontos, mas só um agrupamento dos pontos em conjuntos.

O objeto 2 possui ainda outro atributo que determina como cada agrupamento de pontos deve ser considerado na formação de um elemento de malha.

attribute "element type" string "triangles"

No exemplo, os elementos são definidos como triângulos ou seja, o agrupamento dos pontos deverá formar uma célula triangular.

No rodapé, fechando o arquivo, cria-se o objeto Field, reunindo-se os demais objetos.

A declaração,

object "irregular positions irregular connections file" class field

define a criação de um objeto *field* com conexões irregulares entre pontos irregularmente distribuídos pelo domínio, ao passo que as declarações,

component "positions" value 1

component "connections" value 2

component "data" value 3

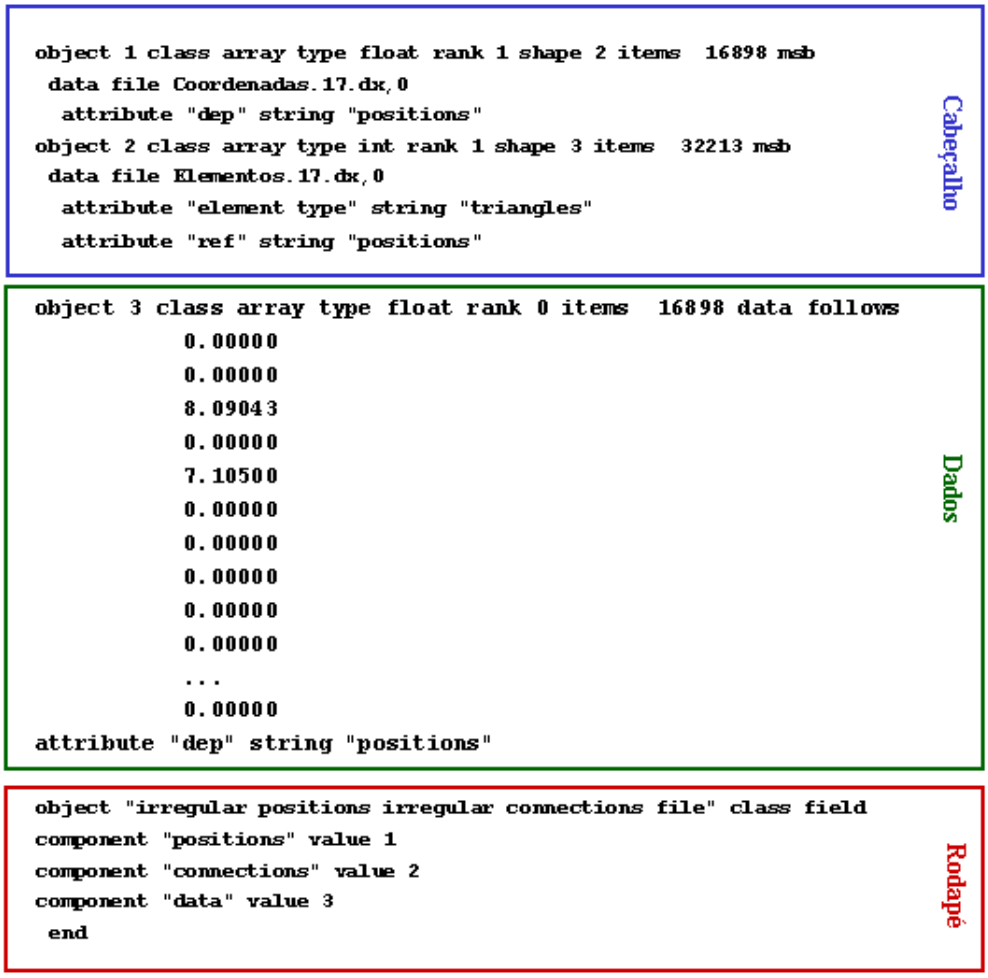

Figura 2.10: Exemplo de arquivo de dado escalar em domínio bimensional.

estabelecem o relacionamento entre os objetos 1 ,2 e 3 para a consolidação do *field* de dados.

Outros exemplos estão relacionados no apêndice A , como referência para elaboração de arquivos de extensão *dx* que possam ser utilizados em programas visuais construídos com o OpenDX.

2.2.2 Programa de Visualização (Visualization Program)

No OpenDX, um programa de visualização é o resultado da reunião e conexão de módulos, que encapsulam subrotinas, segundo uma lógica para a geração de uma imagem como *output*. Os programas de visualização são construídos no VPE (*Visual Program Editor*), apresentado na figura 2.11.

Os módulos podem ser selecionados à esquerda e inseridos no *canvas*. Os *inputs* e *outputs* de dados são representados graficamente por "indentações" na caixa do módulo.

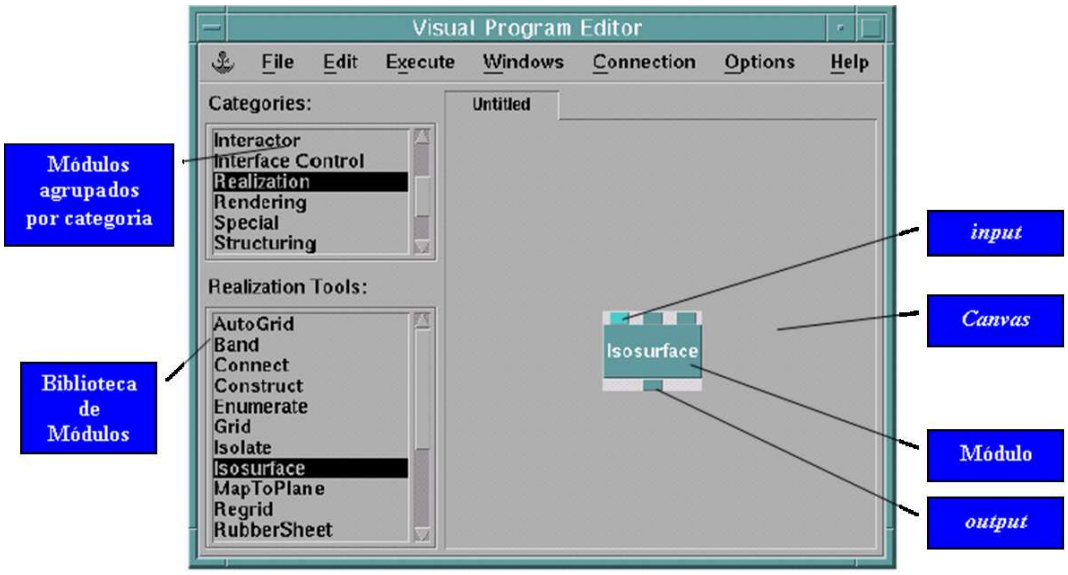

Figura 2.11: Visual Program Editor

Os *inputs* servem para a entrada dos dados que serão processados pela subrotina. Uma vez processados, os dados saem pelo *output* e podem assim alimentar o *input* de outro módulo. Cada indentação (*input/output*) recebe ou repassa somente um tipo de dado.

A quantidade de *inputs* ou *outputs* pode ser ampliada para atender à necessidade do programador/usuário. Basta selecionar o módulo com um *click* do *mouse* e acessar *Edit* na barra da janela do VPE, onde se encontra essa opção.

Uma vez dispostos no *canvas*, os módulos podem ser conectados segundo uma lógica para a construção do programa de visualização. Os *outputs* dos módulos são conectados ao *input* de outros, conforme mostra a figura 2.12.

Um recurso importante que pode ser utilizado durante a construção de um programa de visualização é a *Caixa de Configuração* que pode ser aberta clicando-se sobre a caixa do módulo que se deseja editar. Essa *Caixa de Configuração* apresenta os tipos de variáveis que são relacionadas no *input* e *output* de cada módulo. Na figura 2.13 é apresentada a *Caixa de Configuração* do módulo ISOSURFACE.

O módulo possui um nome que está designado no campo *Notation*. Esse nome pode ser editado diretamente caso se deseje alterá-lo. Logo abaixo estão relacionadas as variáveis de *input*, onde são descritos o nome da variável e o tipo. Cada uma delas está relacionada com uma das "indentações" de (*input*) que aparecem na parte superior da caixa do módulo. O campo *value*, do lado direito, serve para se atribuir diretamente valores a essas variáveis. Caso essa opção seja feita, essas variáveis tornar-se-ão constantes matemáticas dentro da subrotina. Se por exemplo o módulo ISOSURFACE receber dados no seu *input data* advindos do *output* de outro módulo, a exibição dessa variável na *Caixa de Configuração* fica desabilitada e torna-se impossível inserir valores no campo *value*. Esse campo só se tornará habilitado novamente, quando houver a desconexão entre *output* do outro módulo e o *input* do módulo ISOSURFACE. Quando a exibição da variável está desabilitada, o campo *source* passa a exibir o nome do módulo que está conectado àquela variável.

Caso não sejam utilizados os *inputs* ou *outputs* de um módulo durante a construção

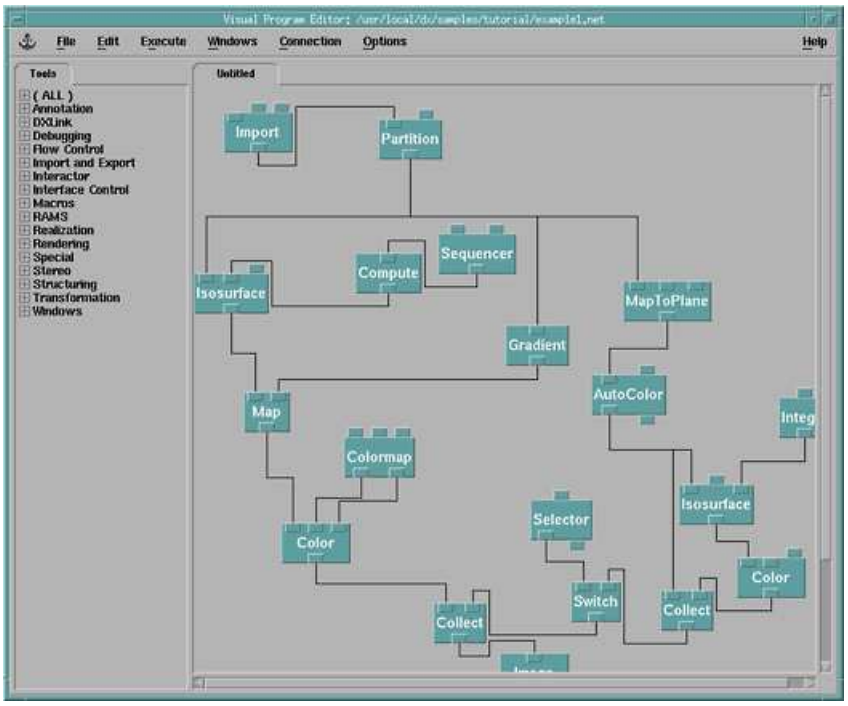

Figura 2.12: Programa Visual no VPE

de um aplicativo, as variáveis relacionadas assumem o valor de *default* do módulo. Se for necessário alterar o valor de *default* por outro, basta inserir diretamente o valor desejado no campo *value*.

Nos botões abaixo da *Caixa de Configuração*, encontram-se mais algumas opções.

- **Ok :** Salva as alterações e fecha a *Caixa de Configurações*;
- **Apply:** Salva as alterações, mas não fecha a *Caixa de Configurações*;
- **Expand:** Faz aparecer variáveis que foram opcionalmente ocultadas. Essa opção é dada ao usuário que pode acionar o botão *Hide*, ao lado da descrição do tipo de variável;
- **Collapse:** Faz exatamente o oposto do que faz o botão *Expand*;
- **Description:** Traz uma descrição dos parâmetros de *input* e *output* do módulo;
- **Restore:** Restaura as condições de *default* do módulo;
- **Cancel:** Cancela as operações e fecha a *Caixa de Configurações*;

2.2.3 Module Builder

O OpenDx permite que o usuário produza seus próprios módulos. Para isso oferece como recurso uma interface gráfica que facilita a criação dos arquivos necessários para a implementação de um novo módulo, o *Module Builder*. Para se construir um módulo, são necessários três arquivos:

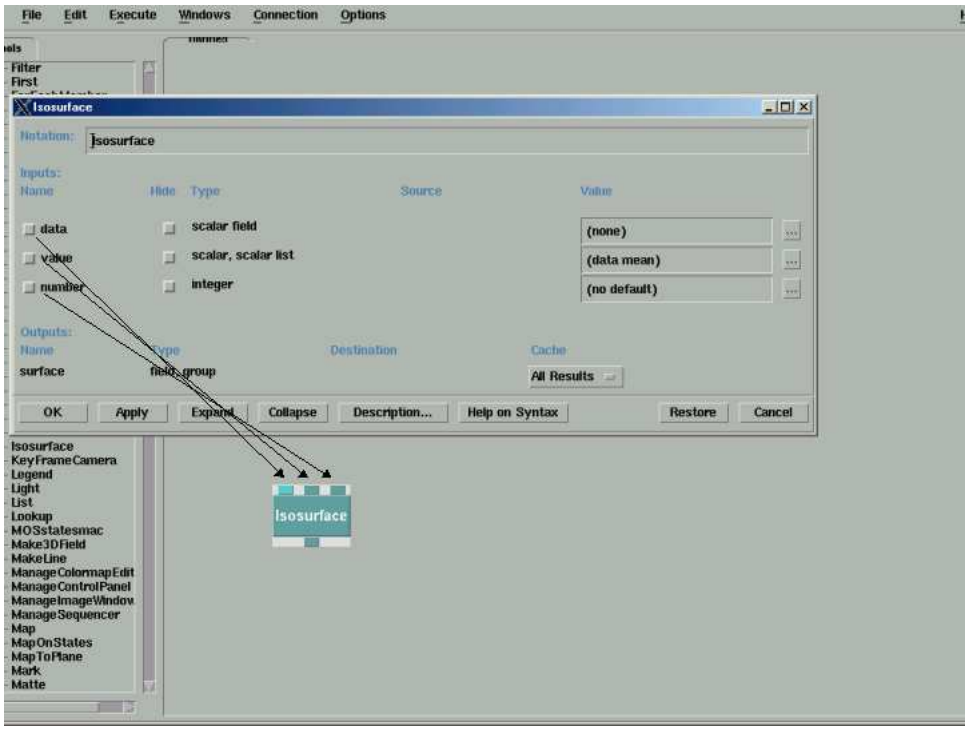

Figura 2.13: Caixa de Configuração do Módulo

1. **arquivo de descrição do módulo**

   O arquivo de descrição possui extensão *mdf* e contém informações sobre o módulo a ser construído, como por exemplo seu nome, a categoria à qual pertencerá, um descrição do mesmo, seus *inputs* e seus *outputs*. Esse arquivo é gerado automaticamente quando se utiliza do *Module Builder*.

2. **arquivo com o código C**

   Esse arquivo traz a funcionalidade do módulo.

3. **arquivo makefile**

   O arquivo *.make* é o arquivo de extensão C compilado.

A construção de um módulo segue os seguintes passos:

- deve-se definir quais serão seus *inputs* e *outputs*;
- deve-se criar um arquivo de descrição do módulo;
- deve-se escrever o código que será executado pelo módulo;
- deve-se compilar o módulo;
- deve-se ativar o OpenDx para incorporar o módulo;

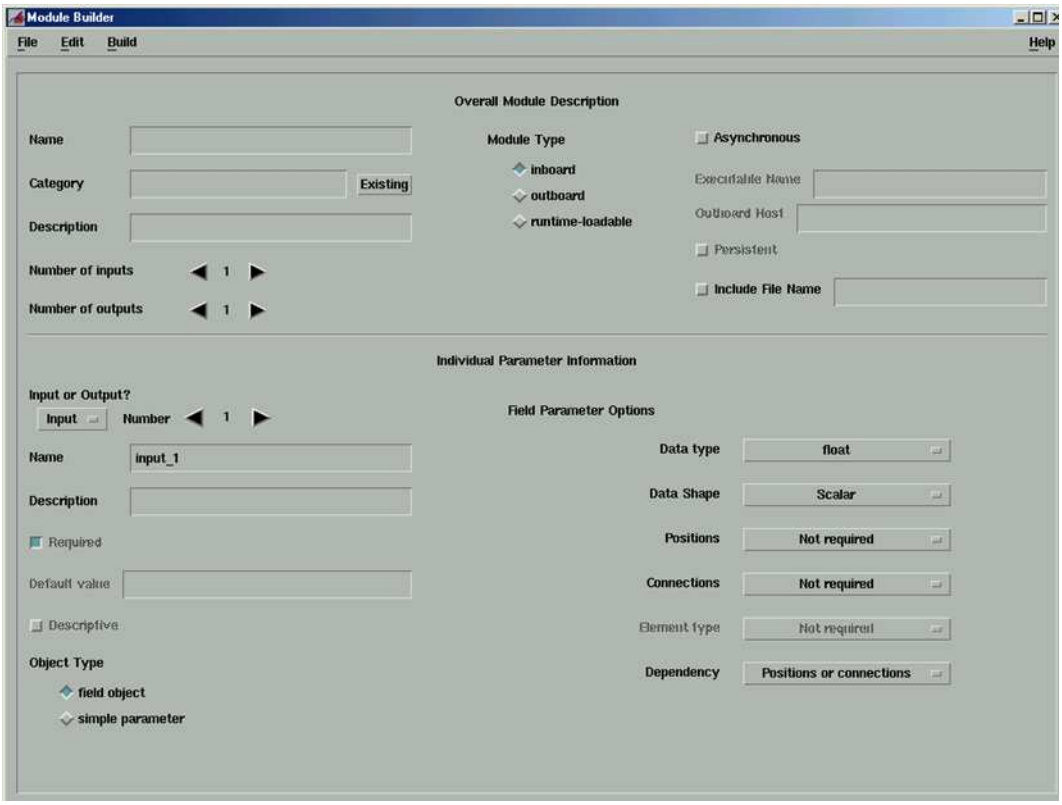

Figura 2.14: Module Builder

O *Module Builder* á apresentado na figura 2.14. Ele é dividido em duas partes, sobre as quais é feita uma descrição.

Parte 1 - Overall Module Description

Essa parte é designada para a descrição do módulo que será criado. Conta com os seguintes campos (figura 2.14):

1. ***Name*** - nome do módulo;
2. ***Category*** - a categoria à qual esse módulo pertencerá;
3. ***Description*** - é a descrição do módulo que será visível na *Caixa de Configuração* do módulo;
4. ***Number of inputs/outputs*** - determina quantos serão os *inputs* e *outputs* do módulo;
5. ***Module type*** - aqui o programador determina se o módulo será *inboard*, *outboard* ou *runtime-loadable*;
    - *Módulo inboard:* o módulo é compilado dentro do arquivo executável do *Data Explorer* e passa a fazer parte dessa nova versão do OpenDx que é gerada;

- *Módulo outboard:* o módulo é compilado mas não é incorporado ao arquivo executável do *Data Explorer*, funcionando em processo separado. O arquivo *outboard* tem a desvantagem de ser menos eficiente, porque os objetos de dados precisam ser transferidos ao módulo via *sockets*, diferentemente dos módulos *inboard* e *runtime-loadable*, que utilizam apontadores em memória compartilhada;
- *Módulo Runtime-loadable:* nesse caso, o módulo compilado também não é incorporado ao arquivo executável do *Data Explorer*, mas pode ser carregado sempre que necessário, pois fica a disposição como numa biblioteca de módulos;

6. ***Executable name*** - caso seja feita a opção por um módulo *outboard*, esse campo torna-se de preenchimento obrigatório. Nele deve ser informado o nome que será dado ao módulo executável;

7. ***Outboard Host*** - especifica o nome da máquina *host* na qual o módulo *outboard* será executado. O *default* é o *host* local;

8. ***Persistent*** - determina que o módulo *outboard* não será encerrado após produzir o *output*;

9. ***Include File Name*** - opção para inclusão da rotina customizada (arquivo C) diretamente no módulo. Quando o *Module Builder* é preenchido, ele gera uma porção de código conhecido por *Work Routine* dentro do qual deve ser incluído um arquivo de extensão C (código customizado). Essa inserção pode ser feita diretamente abrindo-se o arquivo do módulo e copiando-se o código em linguagem C ou referenciando-se nesse campo (*include file name*) o nome do arquivo C desenvolvido separadamente num editor;

Parte 2 - Individual Parameter Information

1. ***Botão Input/Output*** - esse botão define sobre qual tipo de variável (se de *input* ou *output*) serão atribuídos os demais parâmetros (figura 2.14);

2. ***Number:*** um a um, deve ser modificado até que o número máximo de *inputs* ou *outputs* seja alcançado;

   Por exemplo: Foi definido que o módulo deverá possuir 2 variáveis de *input* e uma de *output*. Iniciando-se pelas variáveis de *input*, faz-se a seleção no botão *input/output* para *input* e deixando-se declarado o campo *Number* com o número "1", faz-se toda descrição dessa primeira variável de *input*. Terminada essa primeira descrição, altera-se *Number* para "2" e preenche-se todas os parâmetros dessa segunda variável de *input*. O passo seguinte então é preencher os parâmetros da variável de *output*, o que deve ser feito após se escolher a opção *output* no botão *input/output*.

3. ***Required*** - o botão deve ser acionado sempre que a variável de *input* for essencial no processamento do módulo;

4. ***Default value*** - valor de *default* que a variável deve assumir . Essa opção é válida quando o botão *required* estiver desativado;

5. ***Object Type*** - define-se a variável é um parâmetro simples ou um objeto *Field*; Dependendo da opção selecionada, opções de parametrização são oferecidas no ***Field Parameter Options*** (parametrizações dos objetos *Field*, figura 2.14) ou no ***Simple Parameter Options*** (parametrizações dos parâmetros simples figura 2.15)

Opções do Field Parameter Options

- ***Data Type:*** se dados são do tipo *int, float, double* etc;
- ***Data Shape:*** se variáveis são escalares, vetor de 1 dimensão, vetor de 2 dimensões etc;
- ***Positions:*** se são *regular* (regulares), *irregular* (irregulares) ou *not required* (não há informações sobre posições);
- ***Connections:*** se são *regular* (regulares), *irregular* (irregulares) ou *not required* (não há informações sobre conexões);
- ***Element Type:*** quando existem conexões, se estas são quadradas, triangulares, cúbicas ou tetraédricas;
- ***Dependency:*** se há dependência dos dados em relação às posições, às conexões ou em relação a ambos;

Opções do Simple Parameter Options

Entre as opções (figura 2.15), o programador poderá determinar que os dados sejam:

- Inteiros ou lista de inteiros;
- Vetor de inteiros ou lista de vetores de inteiros;
- Escalares ou lista de escalares;
- Vetores ou lista de vetores;
- Flag;
- String;

O novo módulo deverá ser salvo, acionando-se o menu *File*, *Save as*. No menu *Build*, deverá se acionada as opções para a geração dos arquivos *.mdf*, *.c* e *.make*.
No terminal, os seguintes comandos devem ser digitados:

make -f nomearquivo.make

e para a geração de um módulo *inboard*:

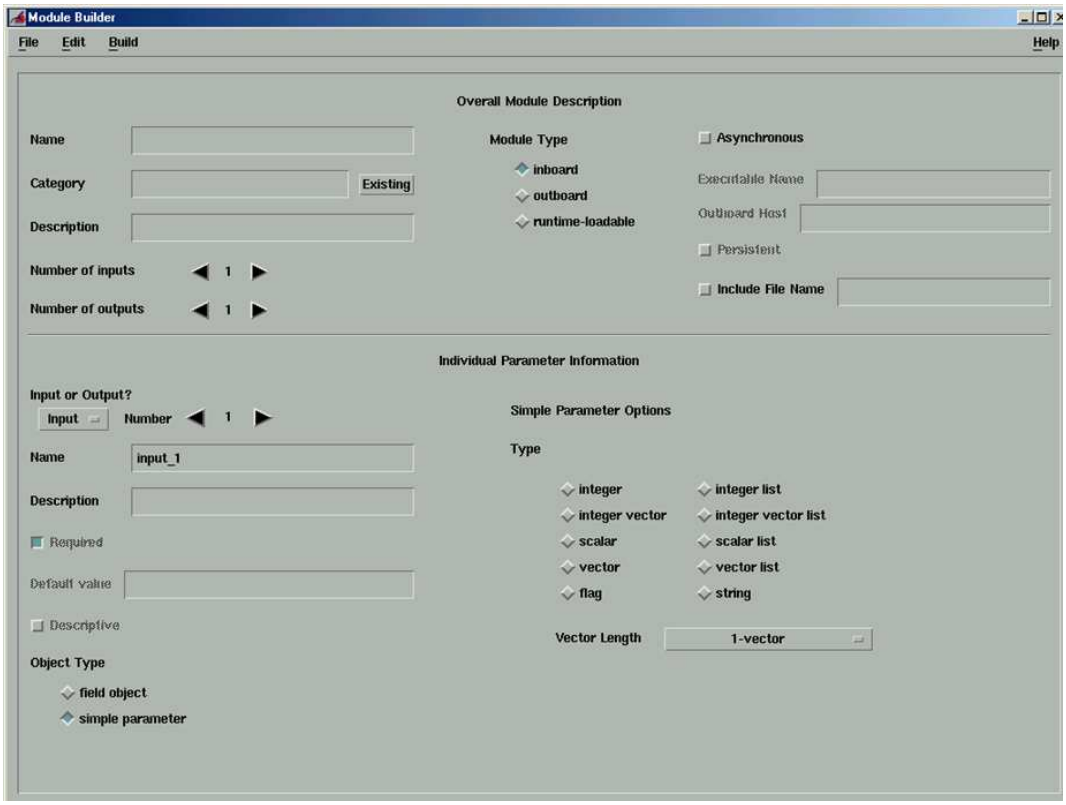

Figura 2.15: Parametrização do Simple Parameter Options

dx -mdf nomearquivo.mdf -exec dxexec

ou caso deseje-se que o módulo seja *outboard* ou *runtime-loadable*, o comando é:

dx -mdf nomearquivo.mdf

2.2.4 Scripting Language

O OpenDX oferece uma interface não gráfica para os programadores que desejam desenvolver seus programas de forma tradicional.

O acionamento do modo *script* pode ser feito no *prompt* através do comando:

dx -script

Uma vez no modo *script*, o usuário pode digitar comandos diretamente embora seja mais comum criar um *script* apartado em um editor e somente submetê-lo à execução no modo *script* através do comando:

include 'nome do script'

Quando um arquivo gerado no *Visual Program Editor* (VPE) é salvo, gerando um arquivo de extensão *.net*, gera-se na verdade um arquivo escrito em *script language* que é gravado no disco. Ou seja, a interface gráfica é uma espécie de facilitador na produção do programa visual que, quando é salvo, é automaticamente "traduzido" para a linguagem *script* do programa. Contudo, há módulos funcionais cujos recursos só estão disponíveis quando estes são empregados no VPE, como por exemplo os módulos das categorias de interação (*Interactors*) e os módulos COLOR, EDITOR, IMAGE, PICK, PROBE, RECEIVER, e TRANSMITTER.

O *script language* do OpenDX possui uma semântica própria que deve ser conhecida pelo programador que opta por desenvolver seus programas visuais através da forma tradicional.

Informações detalhadas sobre como declarar variáveis, como construir expressões e sentenças, de como invocar macros e módulos, de como fazer as parametrizações de argumentos e atributos, podem ser encontradas na referência bibliográfica [9].

No apêndice B é apresentado um exemplo de *script language* e seu equivalente construído diretamente no VPE (figura 2.16). O objetivo é simplesmente demonstrar de forma sucinta, como um código de *script* é construído quando se deseja utilizá-lo preferencialmente ao VPE.

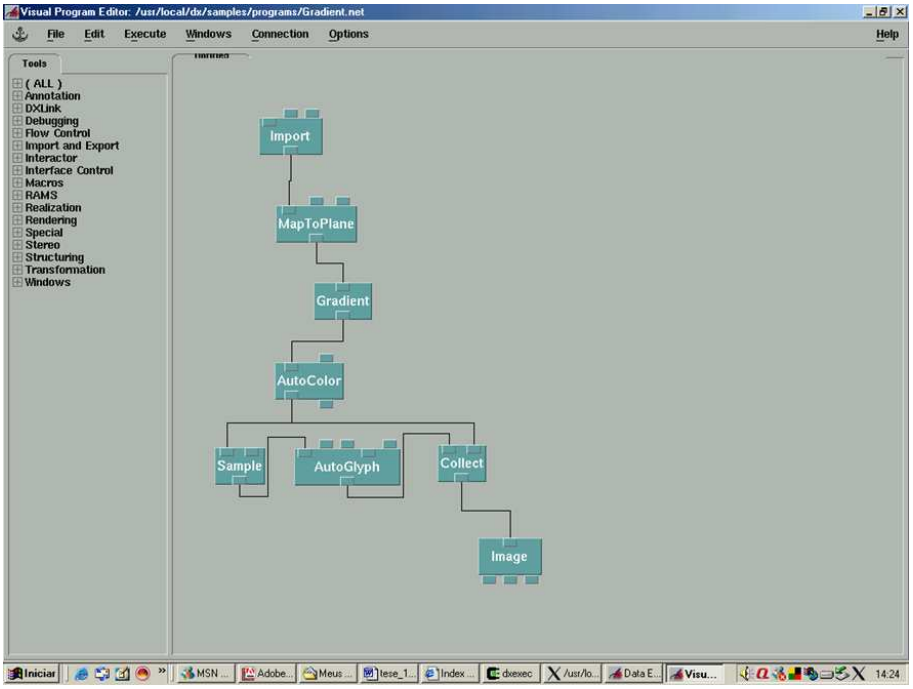

Figura 2.16: Programa construído através da concatenação dos módulos no VPE

O exemplo de programa da figura 2.16 foi extraído da referência [2] e serve para a visualização do gradiente de uma variável escalar num plano. A imagem que ele gera é mostrada na figura 2.17. As cores apresentam a variação da magnitude do gradiente e os vetores indicam a direção, sentido e intensidade de uma grandeza vetorial associada.

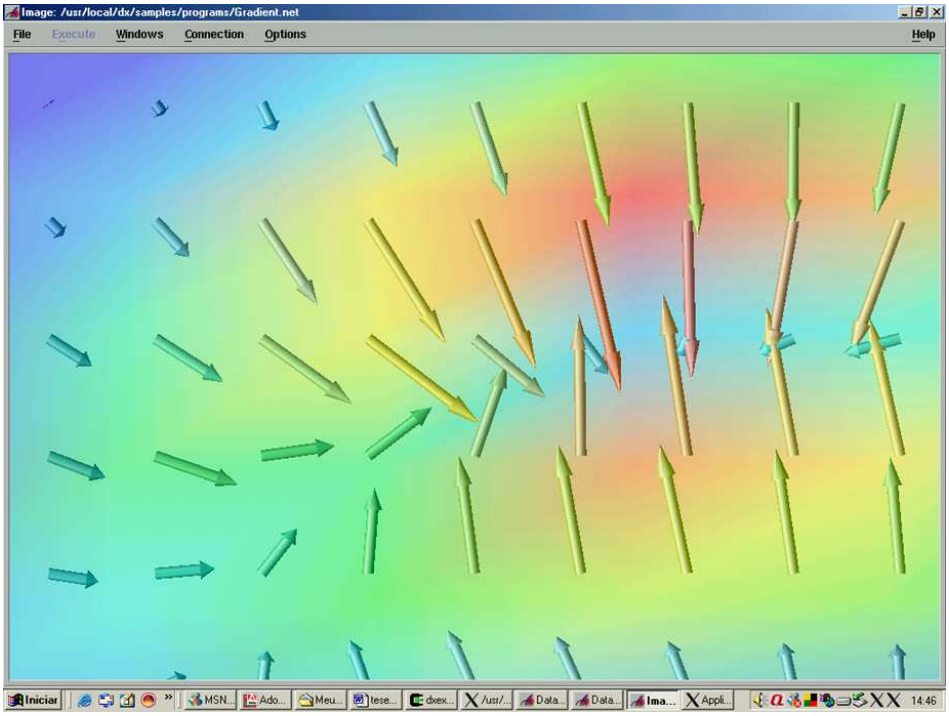

Figura 2.17: *Output* do programa desenvolvido e processado no VPE - imagem de um gradiente.

O mesmo programa construído pela concatenação dos módulos pode ser elaborado na forma de *script*, que se encontra no apêndice B. O imagem gerada através desse *script* é apresentada na figura 2.18.

No *script* os mesmos módulos são chamados e a parametrização de seus argumentos e atributos é feita essencialmente em linha de comando. Um *script* pode ser executado no modo *script* do OpenDX, quando submetido através do comando *include*,

dx -script (aciona o modo script do OpenDX)

include nome_do_arquivo.net

O *output* que resulta da submissão do *script* é o mesmo obtido através do VPE, contudo as funcionalidades do módulo IMAGE, como por exemplo *zoom* (*in* e *out*), *rotate* (rotação), *move* (movimentação/deslocamento) entre outras não podem ser utilizadas por se tratar de recursos exclusivos do VPE.

Comparando-se a geração de imagens através de um programa visual desenvolvido no VPE e no *Script Mode* do OpenDx pode-se concluir que o emprego deste último torna a elaboração de um programa visual complicada nos casos em que muitos módulos precisam ser utilizados. É por isso que geralmente o *Script Mode* é utilizado somente para a identificação de problemas (*bugs*) nos programas.

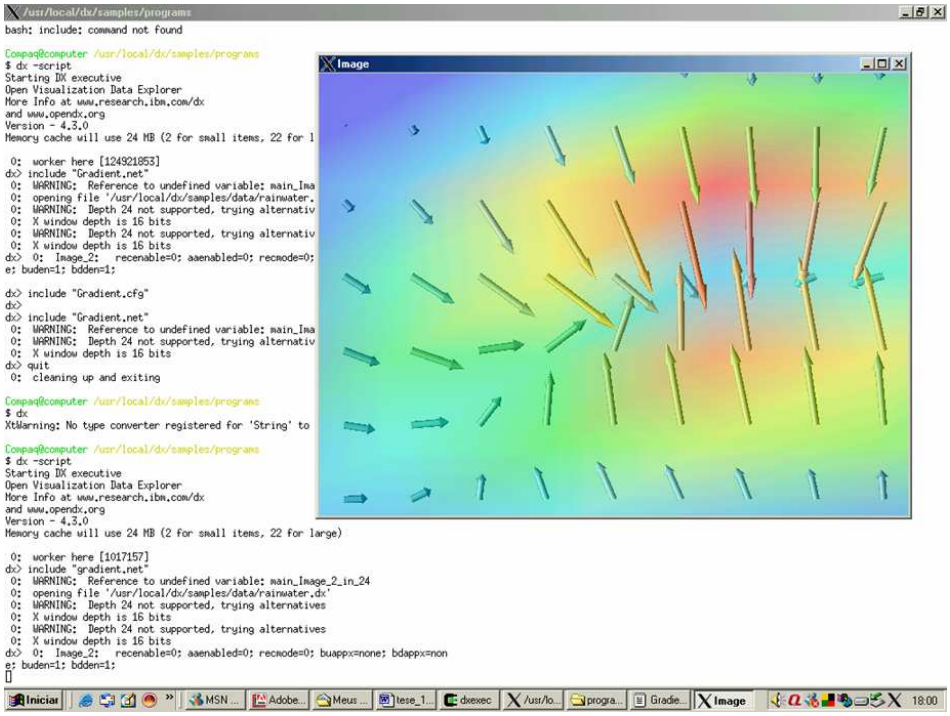

Figura 2.18: Output idêntico ao do programa desenvolvido e processado no VPE, mas dessa vez oriundo da submissão de um *script* ao *Script Mode* do OpenDX. (visível na janela ao lado). Note que a janela de *output* (com a imagem) não traz uma barra com opções na sua porção superior.

2.2.5 Imagem - Apresentação do Resultado

Uma vez processados os dados de entrada pelo programa visual, o resultado desse processamento (a imagem propriamente dita) precisa ser disponibilizado para o usuário. Essa exibição é feita através do *Image Window*, funcionalidade associada ao módulo IMAGE , ou através de uma janela de *display* (funcionalidade associada ao módulo DISPLAY).

O *Image Window* permite que uma série de recursos de interação sejam utilizados pelo usuário, como *zoom* e navegador de imagem por exemplo. São ferramentas importantes para os usuários que necessitam editar as imagens em suas atividades.

Contudo, dependendo de como o programa visual é elaborado, os recursos disponibilizados no *Image Window* podem não funcionar apropriadamente. Nesse caso se recomenda a utilização de outro módulo de exibição de imagem: o DISPLAY. A janela de exibição do DISPLAY não traz recursos de interação com a imagem, mas sua utilização associada a outros módulos permite que o programador recrie os principais recursos de interação do *Image Window*.

A exibição da imagem é o resultado do processamento dos dados de *input* pelo programa de visualização. Dependendo de como é estruturado, o programa de visualização pode oferecer ao usuário a possibilidade de alterar o fluxo de dados em processamento ou dos dados de *input* para a obtenção de diferentes imagens que mostram diferentes abordagens do problema. Nas figuras 2.19, 2.20 e 2.21, através de um seletor, o usuário

pode escolher uma entre as seguintes imagens obtidas de um modelo de molécula: (i) a magnitude do gradiente do campo elétrico (figura 2.19) ,(ii) o quadrado da densidade eletrônica (figura 2.20) ou (iii) a raiz quadrada da densidade eletrônica (figura 2.21). Esse exemplo foi extraído da referência [2], e acompanha o pacote OpenDX.

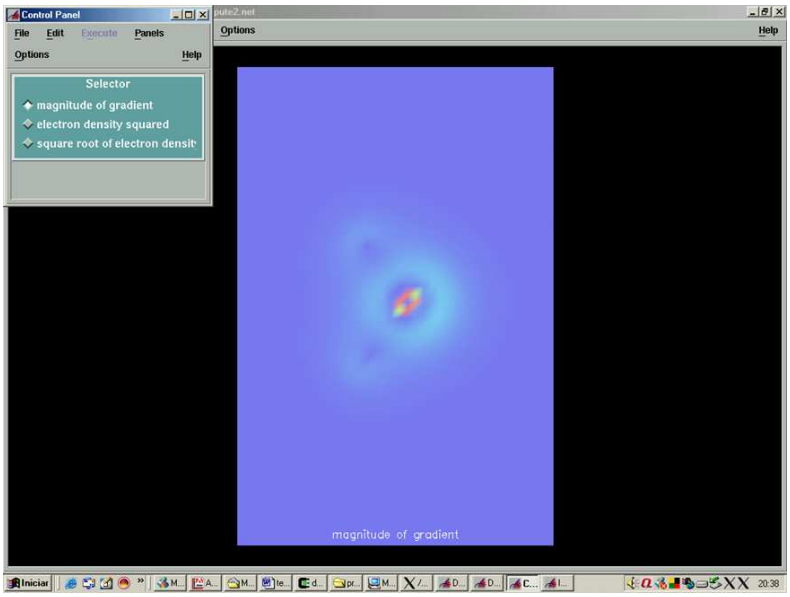

Figura 2.19: Magnitude do gradiente de um campo elétrico da molécula

O seletor, nesse exemplo da molécula, é um exemplo de *Control Panel*, um dispositivo que serve para o controle de exibição de imagens e atua no controle dos dados de *input* do programa visual ou no fluxo de dados, regulando o que chega ao módulo IMAGE (ou DISPLAY) para exibição. O *Control Panel* disponibiliza botões, seletores, *dials*, *sliders* e *interactors* (que permitem a inserção de uma *string*, um número ou um vetor como parâmetros na execução do programa visual) para que seja possível o controle sobre o fluxo de dados e sobre os dados de *input* conforme necessidade do usuário. A decisão de quais dispositivos do *Control Panel* utilizar fica a cargo do programador que desenvolve o aplicativo para atender um conjunto de necessidades dos usuários.

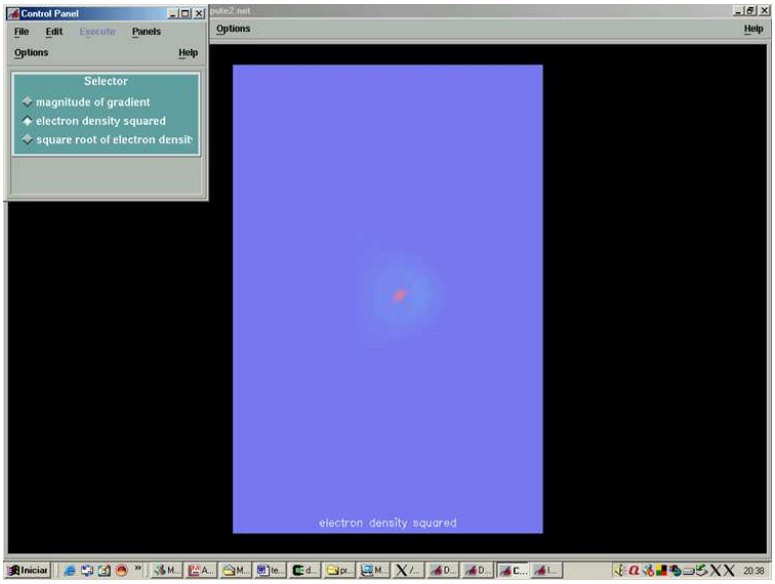

Figura 2.20: Quadrado da densidade eletrônica da molécula

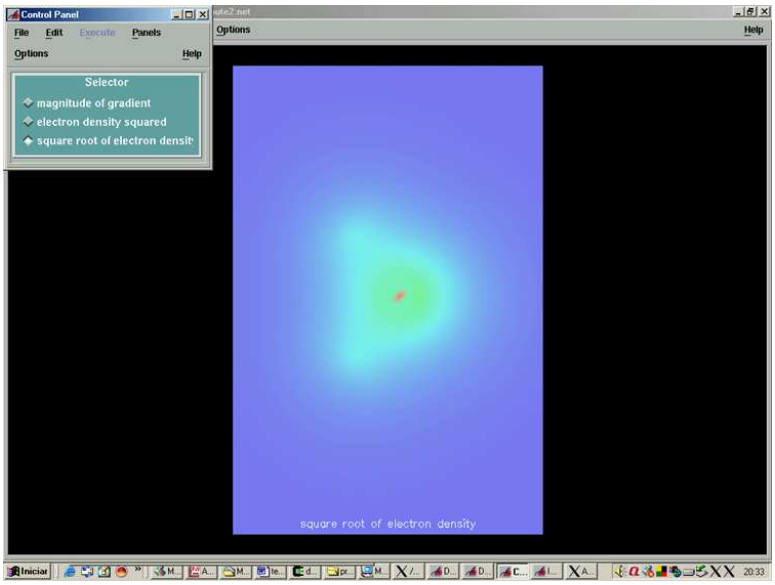

Figura 2.21: Raiz quadrada da densidade eletrônica da molécula

Capítulo 3

Descrição Lógica dos Programas Visuais

A construção dos programas, através do arranjo lógico dos módulos, será descrita nesse capítulo. Essa descrição é importante para usuários interessados em criar novas funcionalidades, novas versões ou novas aplicações a partir desse material.

Os programas desenvolvidos foram:

- **Visualizador_2D_NS**: Visualizador para simulações em duas dimensões;
- **Visualizador_3D_NS**: Visualizador para simulações em três dimensões;

Além dos arranjos lógicos dos módulos, há uma descrição dos próprios módulos que foram utilizados. Isso torna esse texto uma importante fonte para desenvolvedores interessados em um material de referência para novos programas. Contudo, esse material não substitui os manuais distribuídos pela IBM que continuam sendo a principal fonte para qualquer programador que pretenda utilizar o OpenDX.

3.1 VISUALIZADOR_2D_NS

Como foi descrito anteriormente, os programas no OpenDx geralmente são construídos arranjando-se os módulos funcionais no *canvas* do VPE (figura 2.11), formando um "diagrama" lógico de módulos. Para facilitar o entendimento do programa, foram feitas subdivisões nesse diagrama, as quais serão chamadas de *subgrupos*. Cada subgrupo une os módulos que respondem por uma determinada funcionalidade no programa. Os subgrupos foram duplicados para atender as simulações das variáveis escalares e vetoriais, por isso, existem dois fluxos de dados, um para campos vetoriais e outra para campos escalares.

3.1.1 Concatenação e invocação de arquivos

Os arquivos gerados pelo processador possuem o nome da variável e um número de seqüência. Assim por exemplo, os arquivos que armazenam as informações de temperatura possuem os nomes *Temperatura1.dx*, *Temperatura2.dx* e assim por diante. Cada um desses arquivos permite a geração de uma imagem ou de uma "fotografia"

do campo escalar de temperatura num determinado instante. Os números que acompanham o nome dos arquivos permitem que estes sejam ordenados de tal forma que é possível recriar a evolução temporal do campo escalar de temperatura (para melhor compreensão, ver *Controle de Programas*, no Capítulo 4, item 4.1). Para que a imagem correspondente aos dados armazenados em cada um dos arquivos seja gerada, estes precisam ser invocados pelo programa de visualização. Como o visualizador tem ainda a prerrogativa de gerar animações, o carregamento dos arquivos (invocações) pelo programa precisa ser seqüencial e automático.

Para implementar essas funcionalidades, os módulos FORMAT, SEQUENCER, STRING foram utilizados.

O módulo SEQUENCER gera uma seqüência de inteiros que é repassada ao módulo FORMAT. Possui como variáveis de input, os valores máximo e mínimo da seqüência que deve gerar e o valor de incremento que deve utilizar na reprodução da seqüência numérica (1 em 1, 2 em 2 etc).

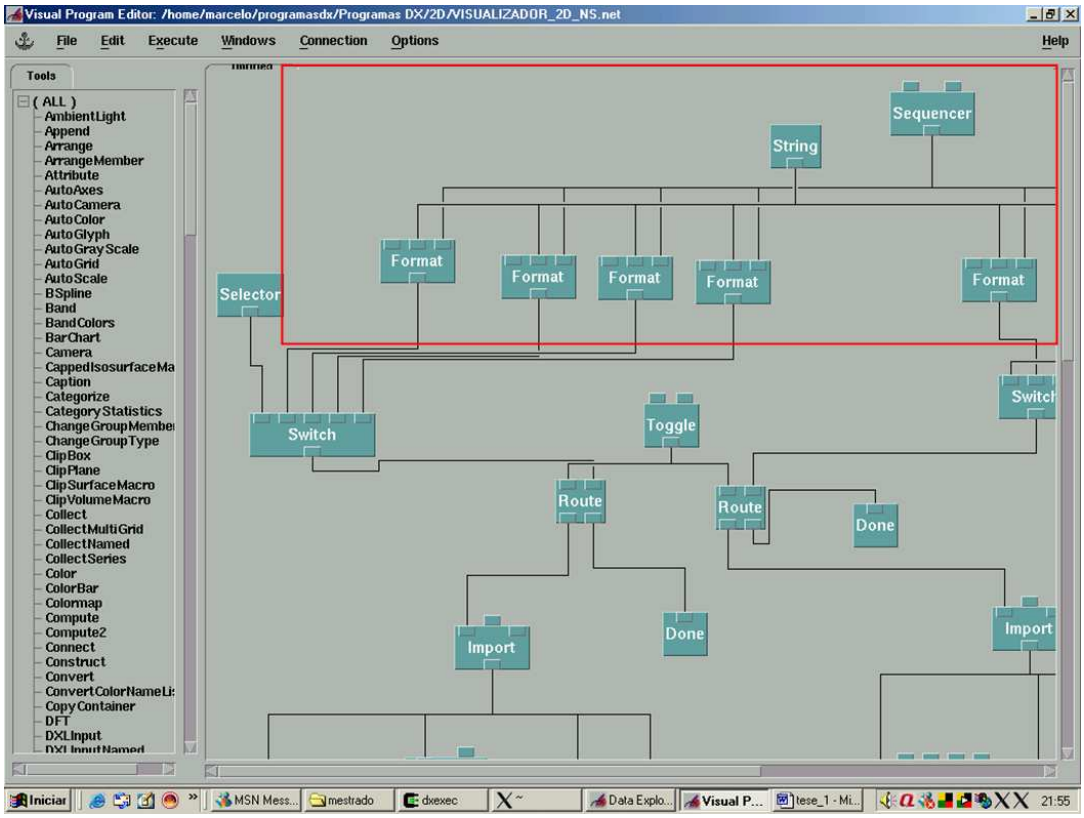

Figura 3.1: Subgrupo de concatenação e invocação de arquivos.

As parametrizações do SEQUENCER são geralmente feitas diretamente pelo usuário (ver *Controle do Sequencer* no Capítulo 4 item 4.1).

O módulo STRING insere uma caixa no *Painel de Controle* que permite ao usuário escrever uma *string* como um *input* no programa. O usuário pode, desta forma, determinar o caminho do diretório (*path*) onde os arquivos gerados pelo processador se encontram armazenados. Essa *string* é também repassada ao módulo FORMAT.

O módulo FORMAT faz a concatenação de entradas advindas dos módulos STRING e SEQUENCER e os repassa a outro módulo. Essa concatenação une:

- Caminho do diretório onde se encontram os arquivos ".dx" (*Input* STRING).
- Variável (nome da variável).
- Número inteiro (*input* SEQUENCER).

O nome da variável (que também é o nome do arquivo) está inscrito dentro do módulo FORMAT (figura 3.2). Por isso, para cada uma das variáveis escalares e vetoriais que há geração de imagens, existe um módulo FORMAT específico.

O módulo FORMAT da variável *temperatura* é apresentado na figura 3.2 .

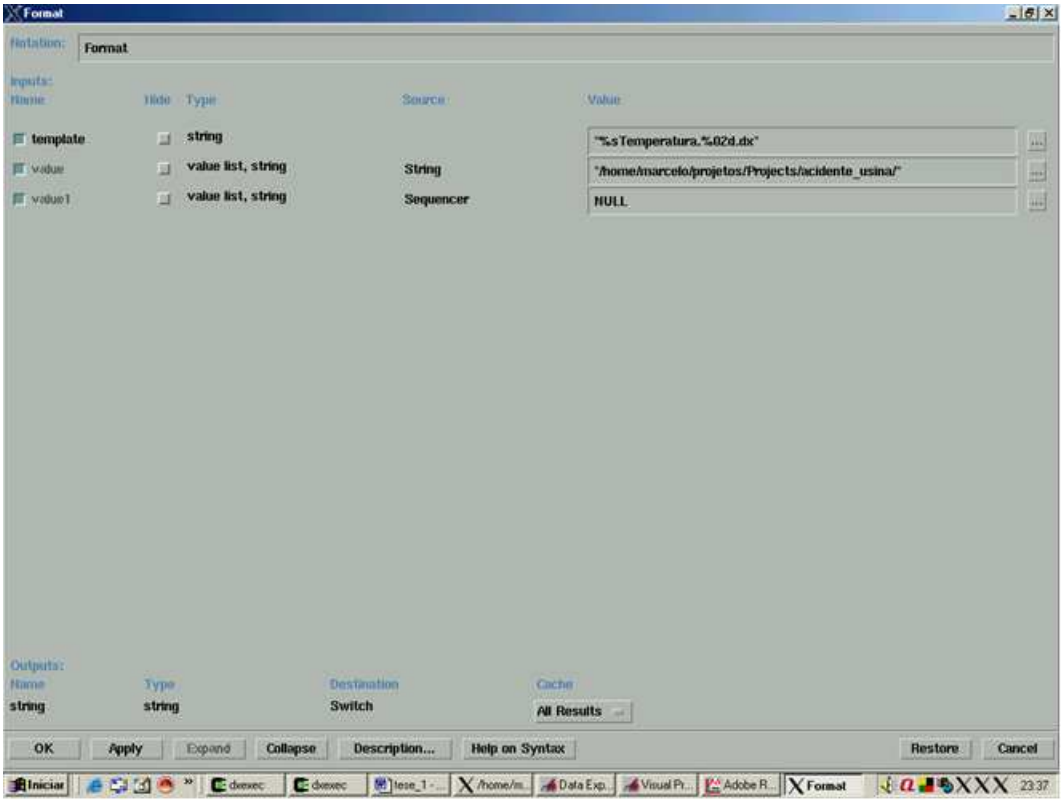

Figura 3.2: Configuração do módulo FORMAT.

No *template* do módulo está inscrito o nome do arquivo (que não se modifica) juntamente com as variáveis de conversão "%s" e "%d". A variável "%s" faz referência a *string* repassada pelo módulo STRING ao passo que "%d" faz referência ao número inteiro repassado pelo módulo SEQUENCER.

Nesse exemplo, as concatenações no módulo FORMAT formaram as seguintes *strings*,

/home/Marcelo/projetos/Projects/acidente_ usina/Temperatura.01.dx
(se o SEQUENCER repassar o número 1)

/home/Marcelo/projetos/Projects/acidente_ usina/Temperatura.02.dx
(se o SEQUENCER repassar o número 2)

Através do *Controle do Sequencer* o usuário gerencia a seqüência de concatenações como desejar. Pode assim invocar arquivos específicos para carregamento e visualização ou determinar o repasse de uma seqüência crescente de inteiros pressionando o botão *play*. Desta forma, automaticamente os números repassados ao módulo FORMAT vão sendo concatenados aos nomes dos arquivos e ao *path* dos arquivos *dx* originando os nomes dos arquivos e seus carregamentos.

3.1.2 Seleção da variável escalar ou vetorial para visualização

Para cada uma das variáveis escalares ou vetoriais, cujos campos podem ser visualizados no VISUALIZADOR_2D_NS, existe um módulo FORMAT dentro do qual há o nome da variável, que faz parte do nome dos arquivos *dx* que serão invocados. Sendo os programas desenvolvidos no OpenDx do tipo *data-flow*, é necessário implementar módulos que permitam ao usuário selecionar e decidir pela variável de interesse no momento da geração da imagem gráfica. Esses módulos - SWITCH e SELECTOR - compõem um segundo subgrupo (figura 3.3) .

O módulo SWITCH funciona como um "filtro", que permite somente a passagem de um dentre vários fluxos de dados possíveis. O usuário, através do módulo SELECTOR, faz sua opção. O SELECTOR repassa essa seleção ao SWITCH que faz a filtragem, estabelecendo qual fluxo continuará a partir de um certo ponto.

Em resumo, os módulos FORMAT concatenam o *path* do diretório onde se encontram os arquivos *dx*, os nomes da variáveis e os números do SEQUENCER, contudo o módulo SWITCH permite somente a passagem de um único conjunto concatenado, que seguirá o fluxo de dados. A seleção do conjunto concatenado é feita pelo usuário através do módulo SELECTOR.

3.1.3 Botão Executar (on/off)

O fluxo de dados selecionado no SWITCH é levado a um conjunto de módulos (ROUTE, TOGGLE, DONE) que funcionam como uma chave de liga/desliga ou seja, servem para o controle da execução do programa. Esse conjunto aparece no *Painel de Controle* como o botão *Executar* (figura 3.3).

O módulo ROUTE serve para direcionar o fluxo de dados para uma de duas direções possíveis. A escolha para qual direção o fluxo de dados deve ser orientada é feita pelo usuário através do módulo TOGGLE, que ligado ao ROUTE, aparece como um botão *on/off* no *painel de controle*. Quando acionado, o fluxo de dados é direcionado ao subgrupo seguinte; se desligado, o fluxo de dados é levado ao módulo DONE, onde é extinto.

3.1.4 Exibição dos modos de visualização das variáveis escalares

Os diferentes modos de visualização das variáveis escalares foram implementados nesse subgrupo, do qual fazem parte os módulos: IMPORT, AUTOCOLOR,

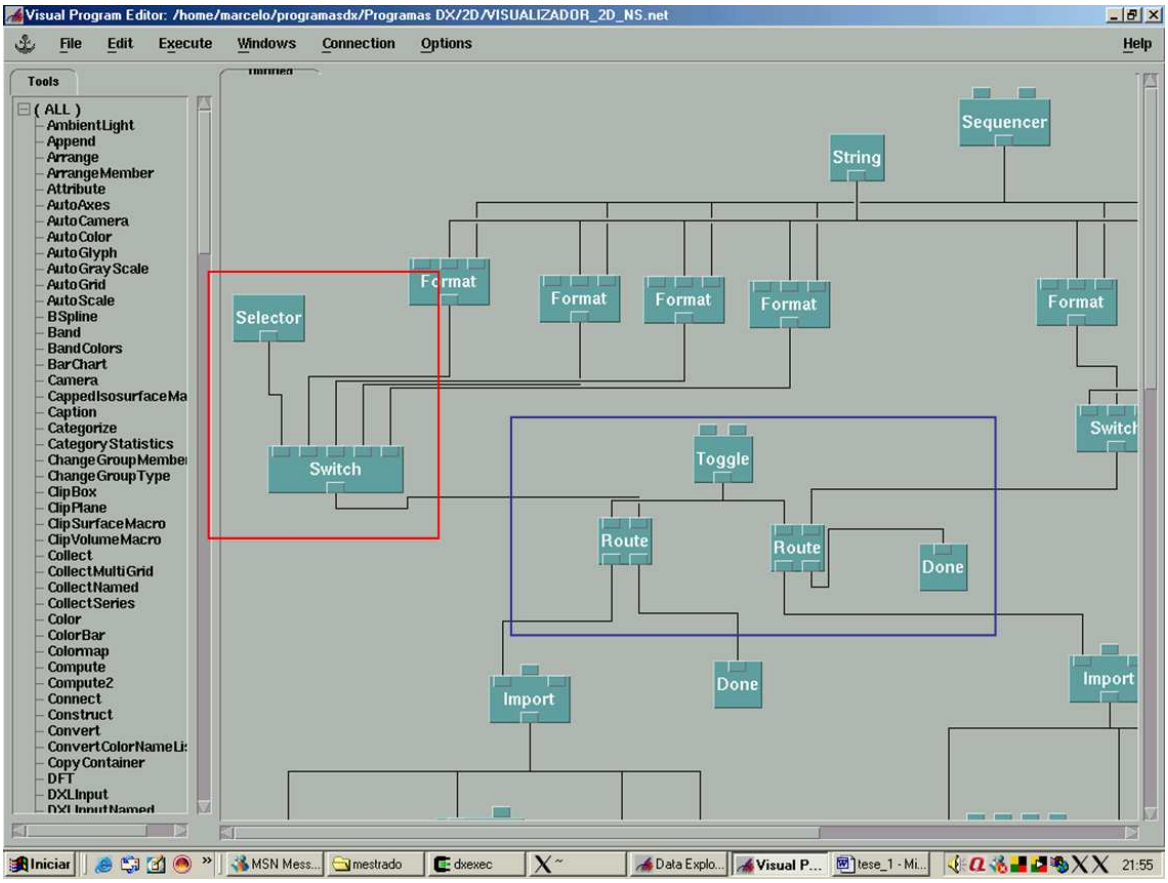

Figura 3.3: Subgrupos *Seleção de variável* (vermelho) e *Função Executar* (azul)

SHOWCONNECTIONS, SHOWBOUNDARY,COLLECT, e um segundo conjunto SWITCH e SELECTOR (figura 3.4) .

O conjunto concatenado no módulo FORMAT (path + nome da variável + número inteiro do SEQUENCER), é selecionado no módulo SWITCH, passa pelo módulo ROUTE (quando o botão executar é acionado) e é então levado ao módulo IMPORT. É esse módulo que efetivamente "lê" e carrega os arquivos *dx* que foram concatenados. O fluxo de dados, daí em diante, segue para quatro caminhos distintos e ao final de cada um deles há um tipo de processamento diferente dos dados, que correspondem a cada um dos modos de visualização que o usuário pode selecionar. Essa seleção do usuário implica na existência de outro módulo SWITCH associado a outro módulo SELECTOR. Cada um dos caminhos que resultam em diferentes processamentos envolve arranjos modulares distintos, que são descritos a seguir.

1. **Modo de visualização - Variável:** quando o usuário opta por esse modo de visualização, o fluxo de dados é direcionado ao módulo AUTOCOLOR . Esse módulo varre os valores do campo escalar em cada uma das coordenadas e atribui aos menores valores a cor azul e aos maiores valores a cor vermelha. Os valores intermediários são coloridos com as cores do espectro que variam do azul ao vermelho. Esse módulo confere visualmente aos dados a noção de gradiente

de campo. Saindo desse módulo, o fluxo de dados é direcionado ao módulo SWITCH.

2. **Modo de visualização - Variável com Malha:** quando há a opção por esse modo de visualização, o fluxo de dados é direcionado, concomitantemente, aos módulos AUTOCOLOR e SHOWCONNECTIONS. Os valores processados nesses dois módulos são então reunidos pelo módulo COLLECT e só então direcionados ao SWITCH. Da mesma forma que no modo anterior, o módulo AUTOCOLOR colore o gradiente, conferindo à imagem o gradiente de variação do campo escalar. Já o módulo SHOWCONNECTIONS exibe a imagem da malha com seus pontos e conexões. O módulo COLLECT agrupa diferentes *Fields* de dados para criar um objeto tipo *Group* (ver Capítulo 2 , seção 2.2.1 *Entrada de Dados*). Isso permite a composição de uma imagem que une diferentes saídas de diferentes módulos.

3. **Modo de visualização - Contorno:** a opção por esse modo de visualização leva ao direcionamento do fluxo de dados ao módulo SHOWBOUNDARY. Esse módulo transforma o *Field* que recebe deixando visível somente os contornos do domínio de dados (limites da simulação), criado no CAD do pré-processador. Saindo desse módulo, o fluxo de dados é direcionado ao módulo SWITCH.

4. **Modo de visualização - Contorno com Malha:** quando o usuário opta por esse modo de visualização, o fluxo de dados é direcionado, concomitantemente, aos módulos SHOWCONNECTIONS e SHOWBOUNDARY. Os valores processados nesses dois módulos são então reunidos pelo módulo COLLECT e só então direcionados ao SWITCH.

3.1.5 Exibição dos modos de visualização das variáveis vetoriais

Os diferentes modos de visualização das variáveis vetoriais foram implementados nesse subgrupo. Fazem parte desse subgrupo os módulos: IMPORT, VALUELIST, AUTOCOLOR, AUTOGLYPH, STREAMLINE, SHOWBOUNDARY,COLLECT, e um segundo conjunto SWITCH e SELECTOR (figura 3.5).

O conjunto concatenado no módulo FORMAT (path + nome da variável + número inteiro do SEQUENCER), é selecionado no módulo SWITCH, passa pelo módulo ROUTE (quando o botão executar é acionado) então levado ao módulo IMPORT. O fluxo de dados, daí em diante, segue para dois caminhos distintos e ao final de cada um deles há um tipo de processamento diferente dos dados, que correspondem a cada um dos modos de visualização que o usuário pode selecionar. Essa seleção do usuário implica na existência de outro módulo SWITCH associado a outro módulo SELECTOR. Cada um dos caminhos, que resultam em diferentes processamentos, envolve arranjos modulares distintos, que são descritos a seguir.

1. **Modo de visualização - Representação Vetorial:** feita a opção por esse modo de visualização, o fluxo de dados é direcionado ao módulo AUTOGLYPH que cria vetores (com direção, sentido e intensidade) da variável vetorial em cada um dos pontos da malha de dados. O fluxo de dados é direcionado, na seqüência, para o módulo AUTOCOLOR que atribuirá cores aos vetores conforme a intensidade dos mesmos. Os vetores de menor intensidade recebem a cor azul ao passo que os de maior intensidade a cor vermelha, cores extremas do espectro

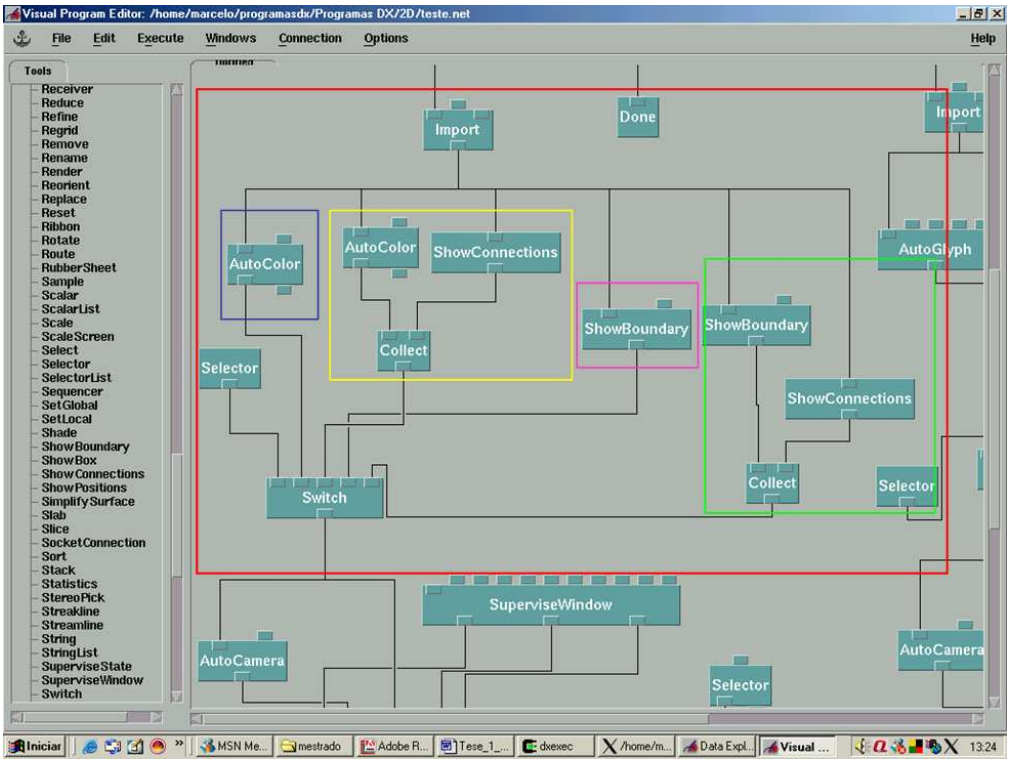

Figura 3.4: Subgrupo *Exibição de variáveis escalares* (vermelho). *Modo de visualização - Variável* (azul), *Modo de visualização - Variável com Malha* (amarelo), *Modo de visualização - Contorno* (rosa) e *Modo de visualização - Contorno com Malha* (verde)

visível. Os demais vetores, com intensidade variando entre o mínimo e o máximo, recebem as cores intermediárias do espectro. Essa variação de cores confere ao campo vetorial o mapeamento das variações de intensidade (módulo) da variável vetorial em análise. Saindo desse módulo (AUTOCOLOR), o fluxo de dados é direcionado ao módulo SWITCH.

2. **Modo de visualização - Representação Linhas de Corrente:** quando se opta por esse modo de visualização, o fluxo de dados é direcionado concomitantemente aos módulos STREAMLINE e SHOWBOUNDARY. Os valores processados nesses dois módulos são então reunidos pelo módulo COLLECT e só então direcionados ao SWITCH. O módulo STREAMLINE cria linhas de corrente a partir de pontos de partida dentro do domínio de dados. Essas linhas representam o percurso que uma partícula descreveria caso partisse desse dado ponto do domínio. O módulo VALUELIST cria uma entrada de dados no *Painel de Controle* através do qual o usuário pode definir pontos do domínio que servirão de origem para as linhas de corrente.

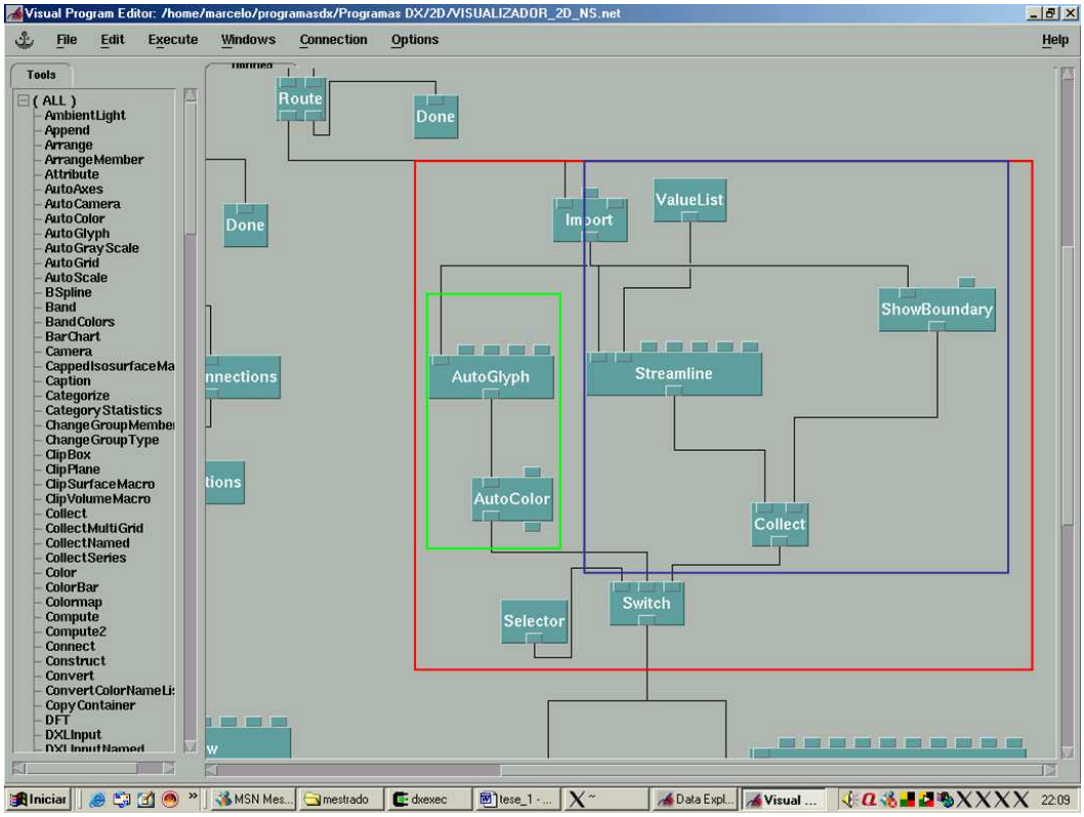

Figura 3.5: Subgrupo *Exibição de variáveis vetoriais* (vermelho). Modo de visualização - Representação Vetorial (verde), Modo de visualização - Representação Linhas de Corrente (azul)

3.1.6 Exibição e edição da imagem

O usuário define, interativamente, qual modo de visualização ele deseja (vetorial e escalar). O fluxo de dados é então filtrado no módulo SWITCH e precisa ser processado por um novo subgrupo, que responderá pela renderização (exibição da imagem) e pelos controles de edição da mesma (*zoom*, rotação, movimentação etc). Esse subgrupo possui os seguintes módulos: AUTOCAMERA, SUPERVISEWINDOW, SUPERVISESTATE, SELECTOR e DISPLAY.

O módulo AUTOCAMERA possui as funções que determinam de que posição o usuário observa o objeto/domínio em análise ou seja, determina a direção da observação (se de cima, por baixo, frontalmente, anteriormente, à esquerda, à direita, centralmente) e o ângulo da observação. Funciona como uma "câmera" através da qual o usuário observa o objeto de análise.

O módulo SUPERVISEWINDOW captura as interações do *mouse* ou teclado com a imagem e repassa para o módulo seguinte. São repassados os seguintes tipos de evento: clique, arrasto, acionamento de teclas.

Os dados dos módulos AUTOCAMERA e SUPERVISEWINDOW são repassados para o módulo SUPERVISESTATE que faz o gerenciamento da AUTOCAMARA a partir dos estímulos recebidos pelo SUPERVISEWINDOW. É o SUPERVISESTATE

que posiciona a "câmera" em relação ao objeto em observação, aproximando ou afastando-se (*zoom in*/ *zoom out*), rotacionando-se em relação ao mesmo (*rotate*) ou movendo-se em relação a sua posição inicial (*move*). Como essas possíveis edições da imagem são relativas, o usuário tem a impressão que o objeto é movido ao invés da "câmera" que ele usa para observá-lo. Através de módulo SELECTOR, o usuário repassa ao módulo SUPERVISESTATE como ele deseja que ocorra a interação com a imagem através do módulo AUTOCAMARA, selecionando através do *mouse* as opções: *zoom*, *move* ou *rotate*. Selecionada uma opção, basta arrastar o cursor do *mouse* na imagem para se produzir o efeito interativo - o módulo SUPERVISEWINDOW irá captar o comando e repassá-lo ao módulo SUPERVISESTATE, que fará o posicionamento da "câmera". Os dados gerenciados pelo módulo SUPERVISESTATE são então direcionados ao módulo DISPLAY, que exporta a imagem para a tela do computador através de uma janela para a visualização/interação com o usuário.

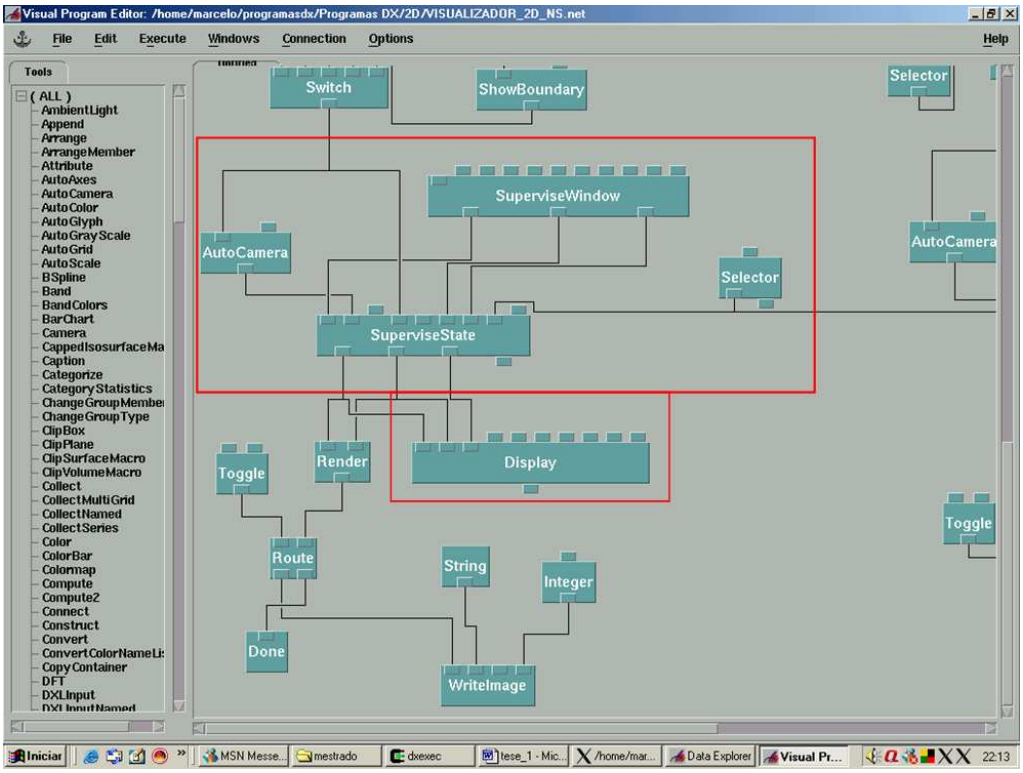

Figura 3.6: Subgrupo *Exibição e edição da imagem*

3.1.7 Geração da animação

A imagem gerada e exportada ao usuário através do módulo DISPLAY pode ser também exportada para outros módulos. E isso é feito no momento em que se deseja gerar uma animação. O módulo RENDER captura o fluxo de dados direcionado ao módulo DISPLAY e o direciona a outro módulo que pode gravar imagens num arquivo: o WRITEIMAGE. Esse módulo pode gravar sucessivas imagens e gerar um arquivo de extensão *miff* facilmente convertido para *mpeg* em programas de código aberto

encontrados nas distribuições do linux). Para completar a construção desse subgrupo, foram unidos ao módulo WRITEIMAGE outros dois módulos: STRING e INTEGER. O módulo STRING repassa ao WRITEIMAGE o *path* e o nome do arquivo de extensão *miff* que deverá ser gerado ao passo que o módulo INTEGER repassa o número de imagens que deverão ser sucessivamente gravadas no arquivo *miff*.

Para dar ao usuário o controle sobre o fluxo de geração dos arquivos *miff*, foi adicionado outro botão *on/off*. O botão aparece para o usuário com o rótulo *Gerar miff*.

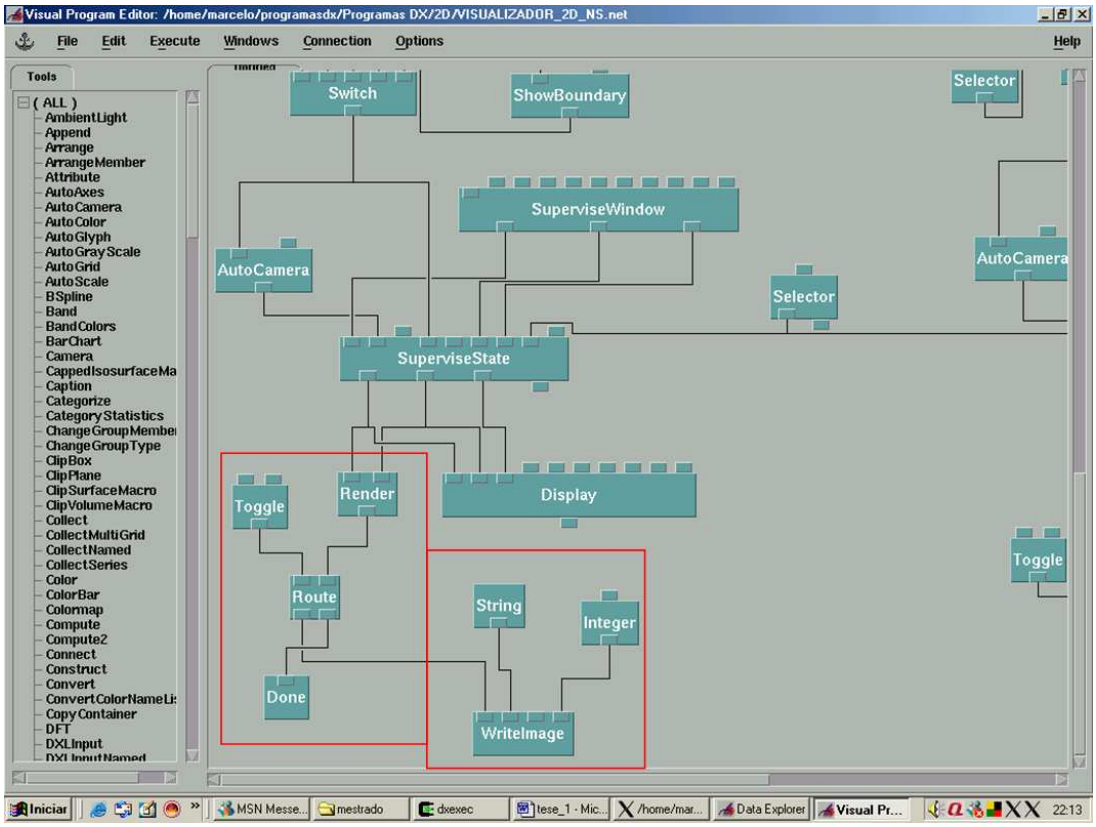

Figura 3.7: Subgrupo *Geração da animação*

3.2 VISUALIZADOR_3D_NS

As diferenças do Visualizador_3D_NS em relação à sua versão em 2D estão basicamente nos subgrupos de geração dos modos de visualização das variáveis escalares e vetoriais. Os demais subgrupos são idênticos e interconectados entre si como já foi descrito anteriormente para Visualizador_2D_NS. Segue abaixo o detalhamento dos subgrupos diferenciados da versão 3D.

3.2.1 Geração dos modos de visualização das variáveis escalares

1. **Modo de visualização - "Gelatina":** trata-se do mesmo arranjo construído no *Modo de Visualização - Variável* (item 1 da seção 3.1.4). O fluxo de dados é direcionado primeiramente ao módulo AUTOCOLOR e em seguida ao módulo SWITCH, através do qual é executada a seleção do modo de visualização desejado. O módulo AUTOCOLOR colore o *Field* 3D e automaticamente confere o efeito "gelatina", através do qual é possível visualizar o campo escalar em toda a profundidade do volume. Esse módulo varre os valores do campo escalar em cada uma das coordenadas e atribui aos menores valores a cor azul e aos maiores valores a cor vermelha. Os valores intermediários são coloridos com as cores do espectro que variam do azul ao vermelho. Desta forma, confere-se visualmente aos dados a noção de gradiente de campo (figura 3.8).

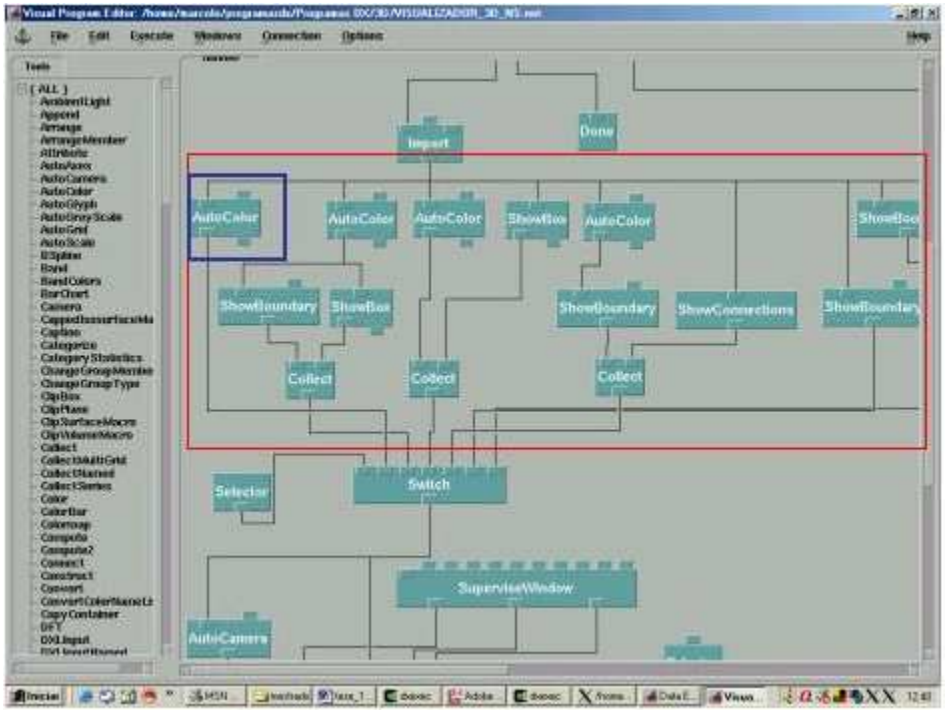

Figura 3.8: Subgrupo *Geração dos modos de visualização das variáveis escalares* (vermelho). *Modo de Visualização -"Gelatina"* (azul).

2. **Modo de visualização - "Gelatina com Caixa":** o fluxo de dados é direcionado concomitantemente aos módulos AUTOCOLOR e SHOWBOX . O fluxo de saída de cada um desse módulos é então reunido no módulo COLLECT antes de ser repassado ao módulo SWITCH. O módulo SHOWBOX automaticamente cria uma "caixa" que delimita o volume e dá ao usuário a noção de profundidade e limites da imagem tridimensional (figura 3.9).

3. **Modo de visualização - Variável no Contorno:** o fluxo de dados é direcionado ao módulo AUTOCOLOR e em seguida ao módulo SHOWBOUNDARY. Essa associação permite a visualização do gradiente por variação de cores (mó-

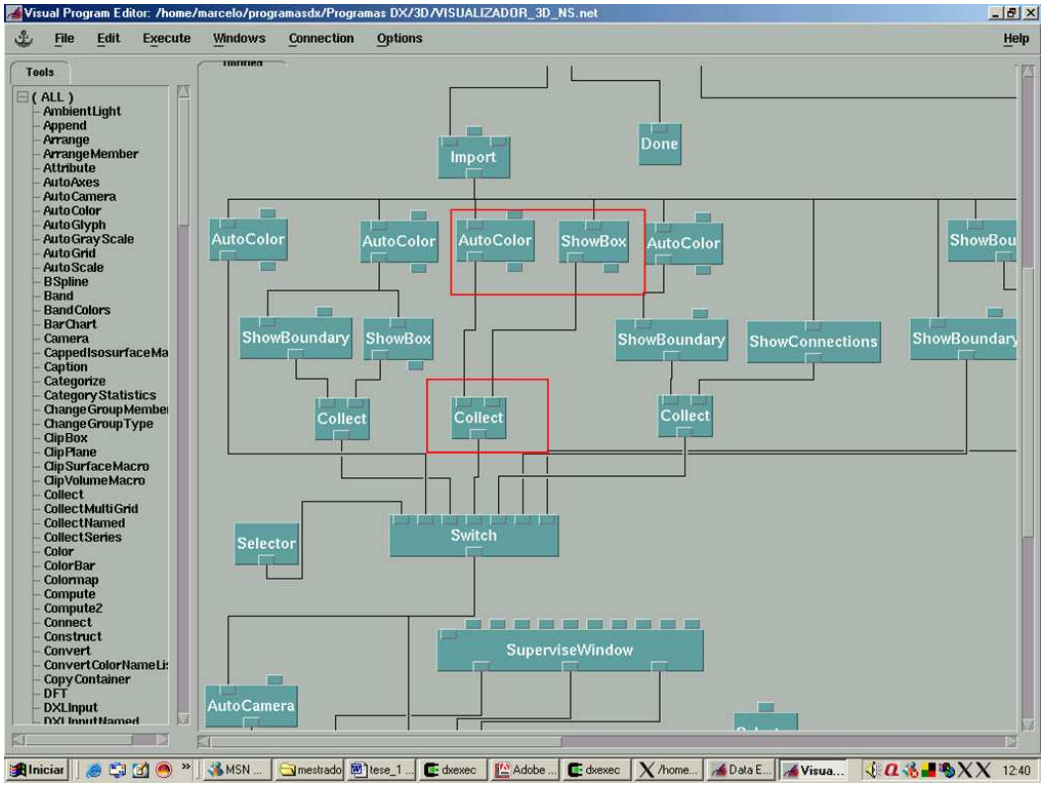

Figura 3.9: *Modo de Visualização -"Gelatina" com Caixa.*

dulo AUTOCOLOR) somente na superfície do volume - característica conferida pelo módulo SHOWBOUNDARY aos *Fields* em 3D. Para delimitar o volume, foi incorporada à imagem uma "caixa" através do módulo SHOWBOX (figura 3.10).

4. **Modo de visualização - Variável no Contorno com Malha :** como foi explicado no item anterior, a associação em série dos módulos AUTOCOLOR e SHOWBOUNDARY permitem a geração da imagem do gradiente na superfície do volume. Para inserir a imagem da malha, bastou introduzir o módulo SHOWCONNECTIONS.

   O fluxo de dados é concomitantemente direcionado aos módulos AUTOCOLOR e SHOWCONNECTIONS no início do subgrupo. Os fluxos resultantes dos módulos SHOWCONNECTIONS e SHOWBOUNDARY são reunidos no módulo COLLECT e daí direcionados ao módulo SWITCH (figura 3.11).

5. **Modo de visualização - Contorno:** O fluxo de dados é diretamente dirigido a um módulo SHOWBOUNDARY e daí para o módulo SWITCH (figura 3.12).

6. **Modo de visualização - Contorno com Malha:** O fluxo de dados é dirigido aos módulos SHOWBOUNDARY e SHOWCONNECTIONS e então reunido em um módulo COLLECT para então ser repassado ao módulo SWITCH (figura 3.12).

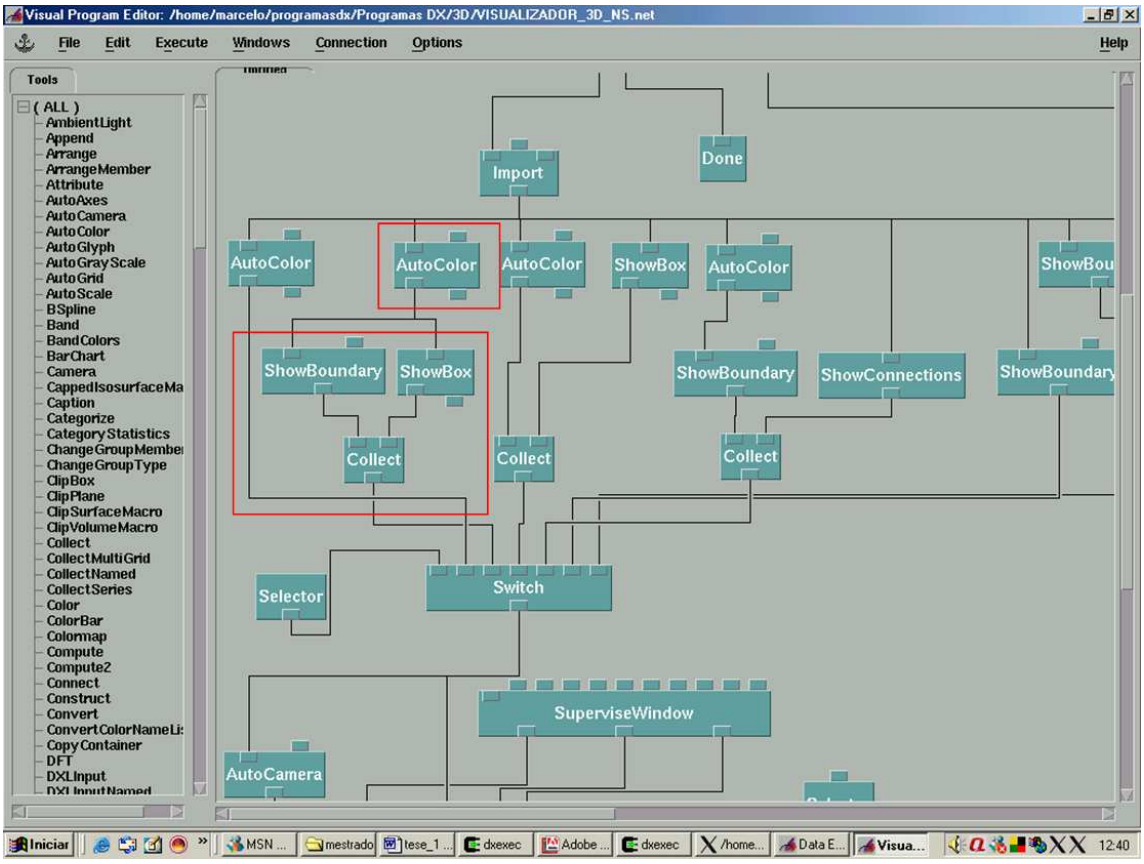

Figura 3.10: *Modo de Visualização - Variável no Contorno.*

3.2.2 Geração dos modos de visualização das variáveis vetoriais

1. **Modo de visualização - Representação Vetorial**: trata-se do mesmo arranjo construído no *Modo de visualização - Representação Vetorial* do programa 2D (item 1 da seção 3.1.5).

2. **Modo de visualização - Representação Vetorial com Caixa**: nesse caso tem-se o mesmo arranjo anterior acrescido de um módulo SHOWBOX. O arranjo é mostrado na figura 3.13 e já foi utilizado no subgrupo que trata de variáveis escalares (no programa 3D).

3. **Modo de visualização - Linhas de corrente:** O fluxo de dados oriundo do módulo IMPORT é direcionado concomitantemente a três outros módulos, a saber:

   - STREAMLINE : esse módulo gera as linhas de corrente. Está associado a um módulo VALUELIST que serve para inserir as coordenadas cartesianas dos pontos de origem das linhas de corrente;
   - SHOWBOX: módulo responsável pela geração da "caixa" que delimita o volume do domínio;

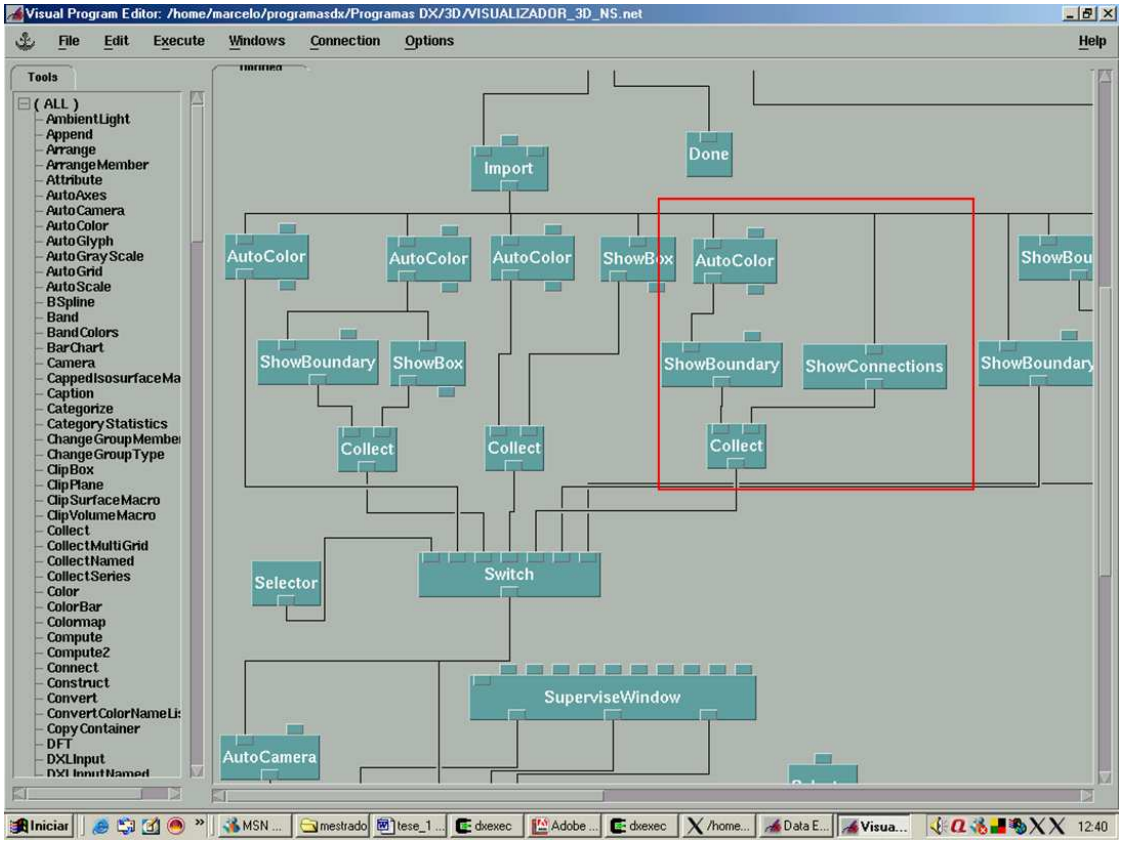

Figura 3.11: *Modo de Visualização - Variável no Contorno com Malha.*

- AUTOCOLOR: utilizado para colorir o campo escalar do módulo da variável vetorial, mas com efeito "Gelatina".

Na versão 2D foi utilizado o módulo SHOWBOUNDARY para que fosse possível observar o domínio da simulação e situar as linhas de corrente. Contudo, na versão 3D, esse módulo restringiria a observação à superfície do volume e impediria a visualização das linhas de corrente no seu interior. Por isso o módulo SHOWBOUNDARY foi substituído pelo módulo AUTOCOLOR que confere o efeito "gelatina" permitindo assim que se visualize o volume todo e conseqüentemente a evolução das linhas de corrente.

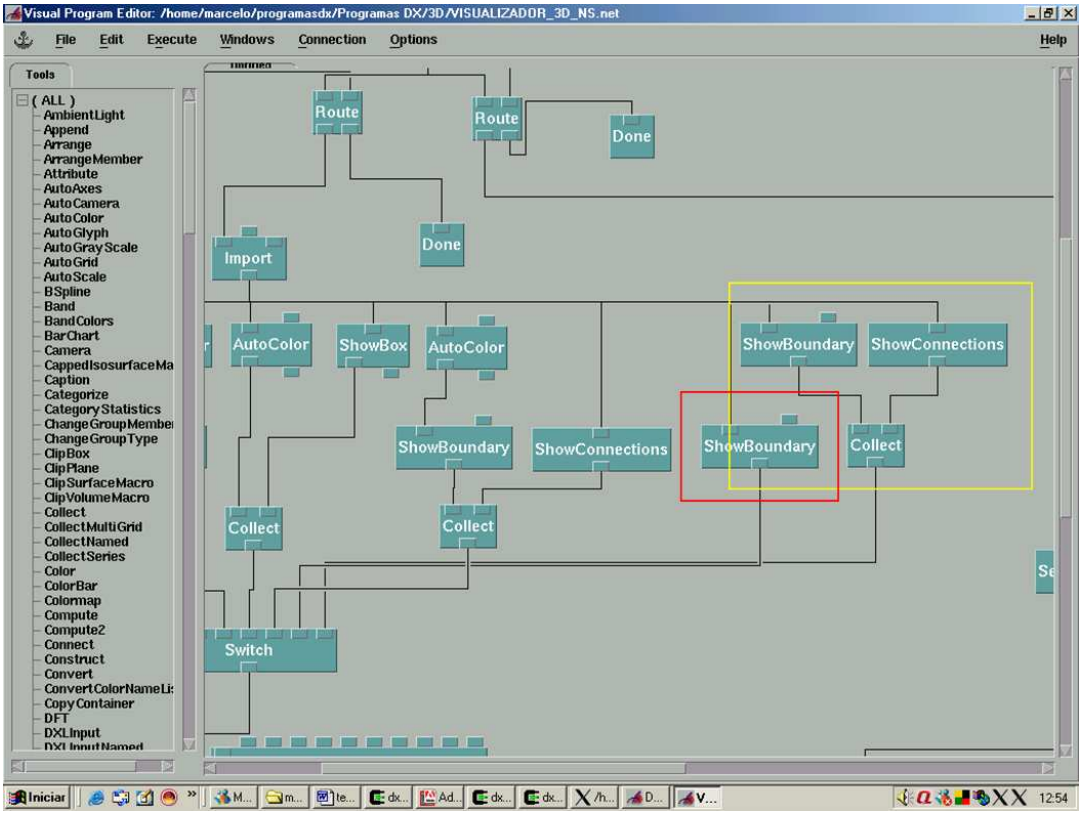

Figura 3.12: *Modo de Visualização - Contorno* (vermelho) e *Modo de Visualização - Contorno com Malha* (amarelo).

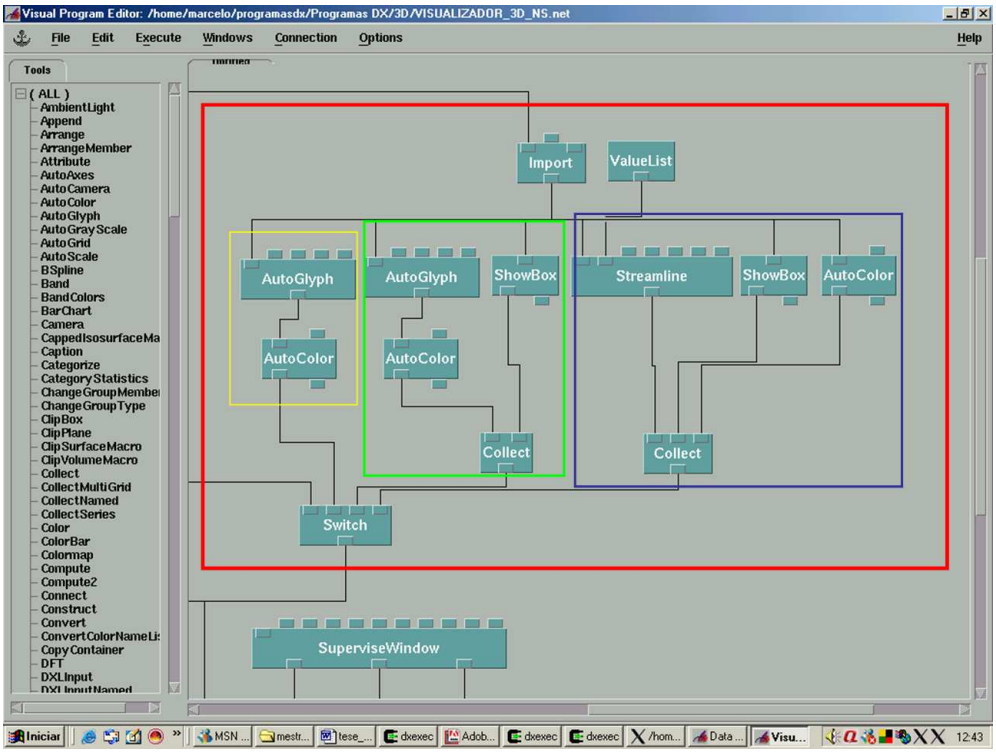

Figura 3.13: Subgrupo Geração dos modos de visualização das variáveis vetoriais (vemelho). *Modo de Visualização - Representação Vetorial* (amarelo), *Modo de visualização - Representação Vetorial com Caixa* (verde) e *Modo de Visualização - Linhas de corrente* (azul)

Capítulo 4

Descrição dos Programas de Visualização

Nesse capítulo é feita uma descrição dos programas de visualização construídos com a utilização do OpenDX e expor suas funcionalidades e modos de utilização. O detalhamento da lógica empregada na construção ou programação foi feito no Capítulo 3.

4.1 VISUALIZADOR_2D_NS

Para o carregamento do Visualizador_2D_NS, a ferramenta OpenDx é inicializada no *prompt* de comando. Em seguida, seleciona-se a opção *Run Visual Programs...*(figura 4.1) .

Navegando até o diretório onde se encontra o programa, seleciona-se o arquivo "Visualizador_2D_NS" para que seja executado (figura 4.2).

O programa é então carregado. Podemos dividi-lo em três partes para facilitar sua descrição (figura 4.3):

- Controle do *Sequencer*;
- Janelas de visualização;
- Controles de programas;

Controle do Sequencer: o *SEQUENCER* é um módulo do OpenDx empregado no controle e geração de animações. Ele permite o seqüenciamento de *frames*, durante a criação ou simples visualização das animações. O *Controle do Sequencer* é a interface gráfica, oferecida ao usuário, para controle das operações que o módulo SEQUENCER oferece (figura 4.4). O *Controle do Sequencer* conta com os seguintes botões:

1. **Play:** seqüencia os *frames* em ordem crescente.
2. **Back:** seqüencia os *frames* em ordem decrescente.
3. **Stop:** pára o seqüenciamento.
4. **Pause:** pausa o seqüenciamento.

Figura 4.1: Inicialização do OpenDX

5. **Repeat:** repete a execução de um seqüenciamento, sempre em ordem crescente de *frames*.

6. **Repeat/Reverse:** repete a execução de um seqüenciamento de *frames* em ordem crescente e decrescente.

7. **Step:** permite o seqüenciamento *frame* a *frame* quando os botões 1 e 2 são pressionados.

8. **Options:** configura o modo de execução dos *frames*. Permite que seja definido o primeiro, o segundo e o último *frame* que deverão ser exibidos durante a execução ou geração da animação. O incremento também pode ser alterado segundo as necessidades do usuário. Esse recurso permite que a exibição ocorra com "saltos", por exemplo, exibindo o primeiro, o terceiro, o quinto... *frames*, caso o incremento seja de 2; ou o primeiro, o quarto, o sétimo... caso o incremento seja de 3 e assim por diante (figura 4.5).

Janelas de Visualização de Imagens: o programa visual gera imagens de campos escalares e vetoriais. Por essa razão, existem duas janelas nas quais as imagens são exibidas: *variável escalar* e *variável vetorial* (figura 4.3).

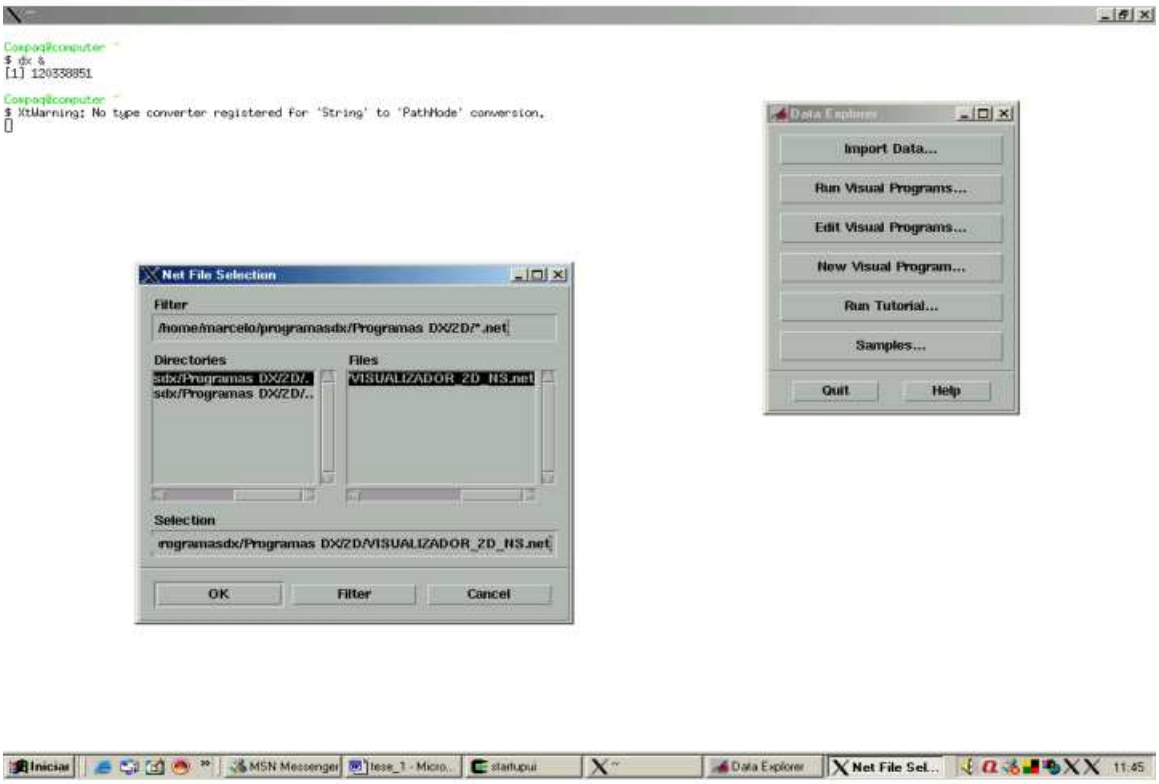

Figura 4.2: Selecionando-se programa visual para execução

A *janela variável vetorial* exibe campos de variáveis vetoriais. Os vetores gerados possuem cores que variam do vermelho ao azul. Quanto maior a intensidade do vetor, mais sua coloração tende ao vermelho. Da mesma forma, quanto menor sua intensidade, mais sua coloração tende ao azul. Além da coloração, o tamanho do vetor também é proporcional ao seu módulo.

A *janela variável escalar* exibe campos de variáveis escalares. Nesses campos, a coloração varia do azul (valor mais baixo do gradiente) ao vermelho (maior valor do gradiente). Os valores escalares entre o mínimo e o máximo são coloridos com as as cores intermediárias do espectro visível. Essa distribuição de cores gera a percepção de como varia o gradiente da variável no domínio de estudo.

Controles de Programa: na *janela de controle* estão todas as funções e recursos disponíveis para que o usuário gere e edite, da forma como desejar, as imagens e animações.

O pós-processador se utiliza dos dados que são gerados pelo processador para a geração das imagens. O processador imprime arquivos com dados que são "lidos" pelo programa de visualização e transformados em imagens. Cada arquivo pode ser transformado numa imagem ou num *frame*, quando o objetivo é a geração de animações.

Especificamente nas aplicações para as quais o Visualizador_2D_NS foi desenvolvido, os arquivos gerados pelo processador têm o nome da variável em análise

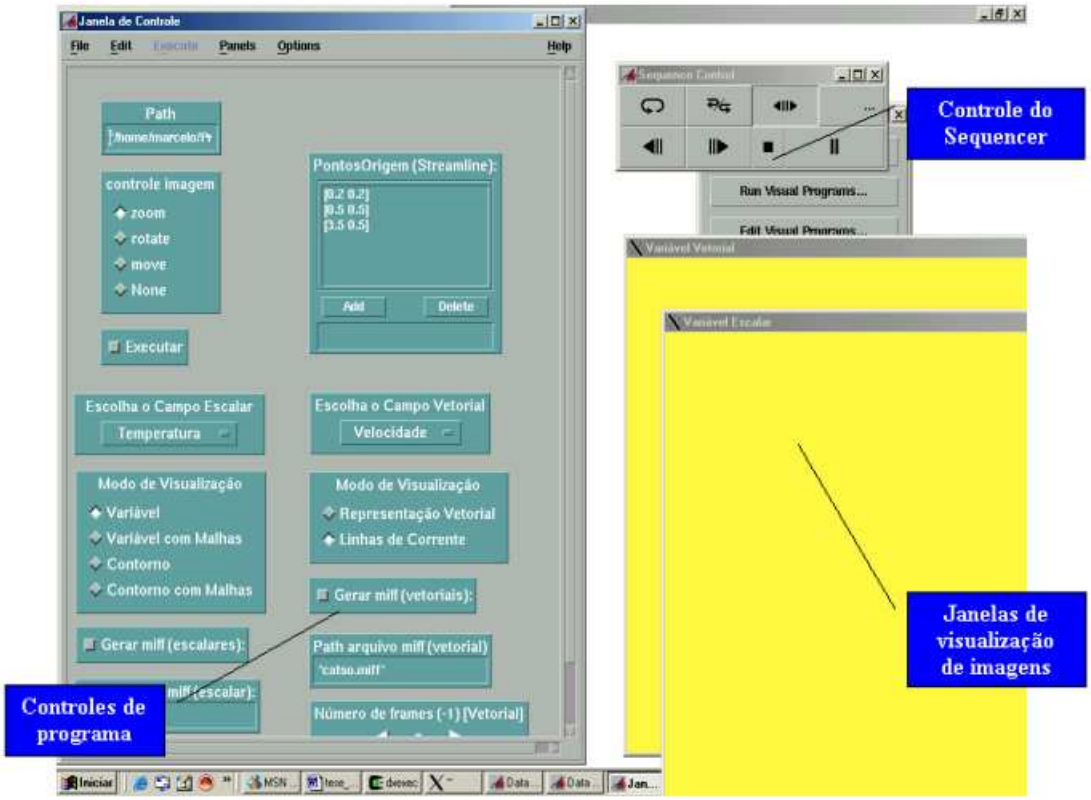

Figura 4.3: Visualizador_2D_NS

acompanhado de um número, que indica a ordem do *frame* dentre uma seqüência de *frames* gerados. Esses *frames* gerados, quando exibidos em seqüência, em ordem crescente, formam uma animação. O usuário ou quem assiste à exibição seqüencial dos *frames* consegue, dessa forma, visualizar a evolução de um fenômeno ao longo do tempo. Em outras palavras, cada *frame* pode ser entendido como uma "fotografia" que registra um momento do desencadeamento de um fenômeno que fora numericamente simulado e, a partir da justaposição dessas "fotografias", consegue-se recriar, do início ao fim, o desenrolar do fenômeno.

Assim por exemplo, os arquivos gerados pelo processador têm o nome "Temperatura1.dx", "Temperatura2.dx", "Temperatura3.dx" e assim por diante, quando a variável sobre a qual se deseja gerar uma imagem é temperatura. Esses arquivos são armazenados num diretório com o nome do projeto sobre o qual se processa a simulação (figura 4.6).

Na janela *Controles de Programa*, a caixa PATH serve para designar o caminho de diretórios e subdiretórios onde se encontram os arquivos de extensão *dx* de um projeto. É nesse "endereço" que o programa visualizador buscará os arquivos para transformá-los em imagens. Uma vez digitado o caminho, deve-se pressionar a tecla ENTER para que seja reconhecido pelo programa.

Logo abaixo, o usuário encontra o botão *Executar*. O programa só buscará os

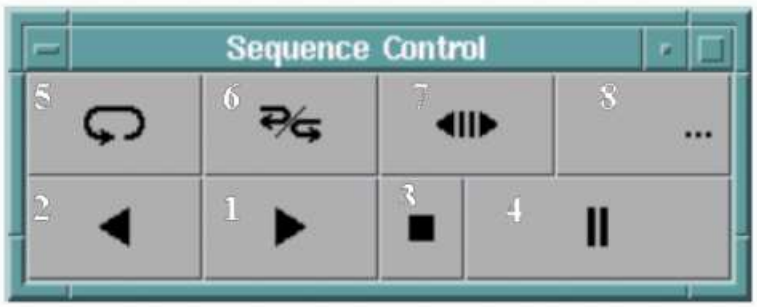

Figura 4.4: Controle do *Sequencer*

arquivos *dx* no diretório discriminado no PATH se este botão for pressionado (figura 4.7).

Abaixo do botão *Executar*, o usuário encontra os menus *drop-down*: *Escolha o Campo Escalar* e *Escolha o Campo Vetorial*(figuras 4.8 e 4.9). São nesses menus que o usuário define qual variável será observada nas *janelas de Variável Escalar* e *Variável Vetorial*.

No menu *Escolha o Campo Escalar*, as opções de campos escalares que podem ser selecionadas pelo usuário são:

- Temperatura;
- Pressão;
- Módulo de Velocidade;
- Módulo do Fluxo de Calor;

No menu *Escolha o Campo Vetorial*, as opções de campos vetoriais que podem ser selecionadas pelo usuário são:

- Velocidade;

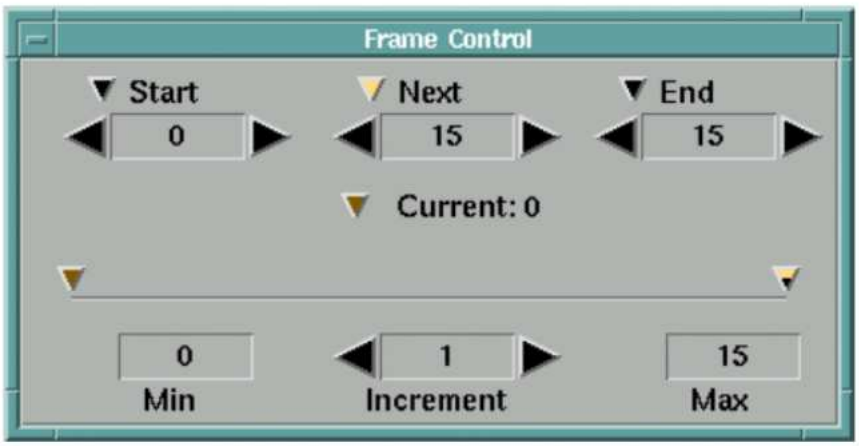

Figura 4.5: *Frame control*

- Fluxo de Calor;

Antes ou durante a exibição das imagens ou animações, o usuário pode selecionar diferentes modos de visualização. Esse diferentes modos permitem ao usuário visualizar o domínio do problema, os campos escalares e vetoriais e a malha de elementos finitos.

Para uma melhor compreensão, esses modos são descritos nas próximas seções. As figuras que ilustram os recursos dos programas visualizadores foram extraídas da referência bibliográfica [21], que trata da dispersão atmosférica de radio nuclídeos na vizinhança próxima a um acidente em central nuclear. A simulação mostra as turbulências geradas pela interação do vento com a edificação e com as forças convectivas da corrente do material liberado nas proximidades das centrais nucleares.

4.1.1 Modos Visualização Variável Escalar

- **Variável:** permite a visualização do campo escalar da variável selecionada. Na figura 4.10, é apresentado o campo escalar de temperatura de uma simulação.
- **Variável com Malha:** permite a visualização o campo escalar da variável selecionada e da malha de elementos finitos utilizada na simulação numérica. Na figura 4.11, é apresentado o campo escalar e a malha de uma simulação.
- **Contorno:** essa opção exibe o desenho do domínio de simulação, estabelecido com o uso do pré-processador (figura 4.12).
- **Contorno com malha:** essa opção exibe o desenho do domínio e a malha de elementos finitos utilizada na simulação numérica (figura 4.13).

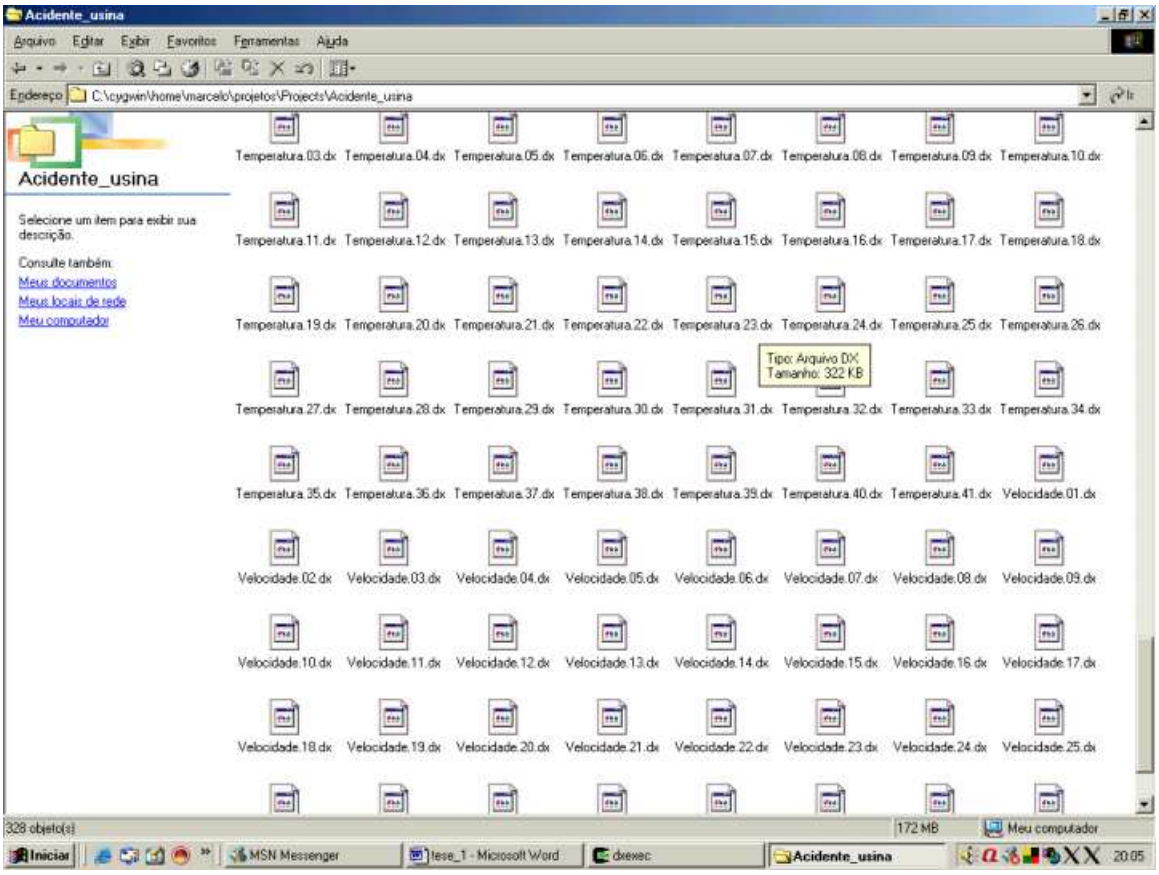

Figura 4.6: Diretório do projeto Acidente_Usina, onde são armazenados os arquivos ".dx" das variáveis "Temperatura" e "Velocidade". O número que acompanha o nome do arquivo permite o seqüenciamento de frames para a geração de animações.

4.1.2 Modos Visualização Variável Vetorial

- **Representação vetorial:** o campo vetorial pode ser visualizado. Os vetores são gerados em tamanhos e cores proporcionais à intensidade no pontos em que são apresentados (figura 4.14).

- **Linhas de corrente:** são linhas que descrevem a trajetória de partículas arrastadas pelo escoamento do fluido (figura 4.15).

  A trajetória depende do posição inicial da partícula. Esses pontos podem ser determinados pelo usuário na janela de controle, através da caixa

  PontosOrigem(*Streamline*)

  Basta digitar as coordenadas cartesianas do(s) ponto(s) na parte inferior da caixa e pressionar o botão *Add*. Ele então aparecerá na porção superior da caixa. Caso deseje-se remover o ponto, basta selecioná-lo com o *mouse* e pressionar o botão *Delete*.

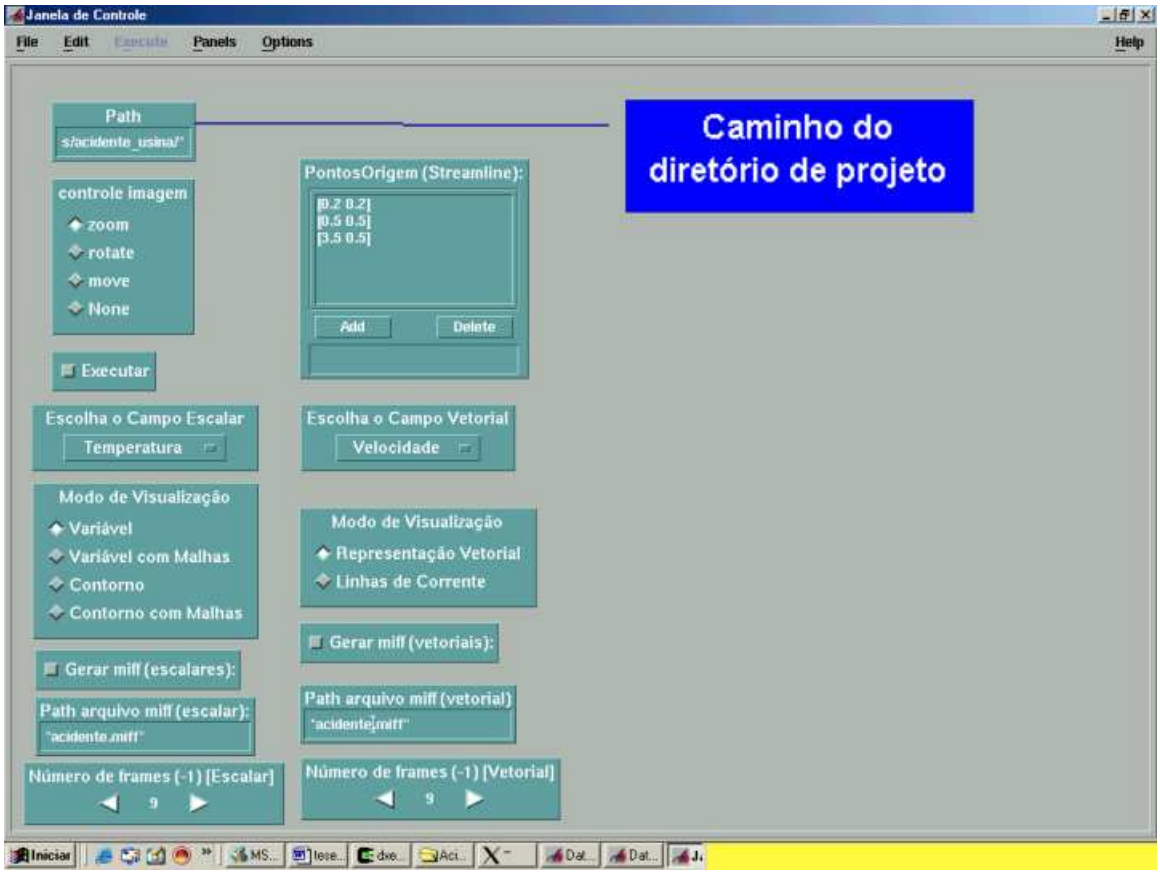

Figura 4.7: Detalhe dos *Controles de Programa*.

4.1.3 Geração de animações

Uma das vantagens de se utilizar o OpenDx para construção de visualizadores são os recursos que ele oferece para a geração de animações e vídeos. Essa funcionalidade foi implementada no Visualizador_2D_NS.

Uma animação é gerada a partir dos dados contidos dentro dos arquivos impressos pelo processador. Esses arquivos são carregados e transformados em imagens nas *Janelas de Visualização de Imagens*. Quando exibidos um a um, temos a sensação de movimento, ou seja temos uma animação.

A partir da animação podemos criar um vídeo. Quando as imagens (*frames*) são exibidas na *Janela de Visualização de Imagens*, a partir dos arquivos numéricos, um outro dispositivo "fotografa" essas imagens. Essas "fotografias" dos frames são gravadas num arquivo de vídeo e quando exibidas seguidamente, geram a sensação de movimento. Nesse caso temos um vídeo.

A diferença entre animação e vídeo está na origem das imagens portanto. Enquanto a animação exibe imagens geradas a partir de arquivos, o vídeo exibe imagens "fotografadas".

Nos vídeos, entre uma fotografia e outra, podem ser incluídas mais "fotografias" geradas a partir da interpolação de imagens criadas computacionalmente. Esses recursos

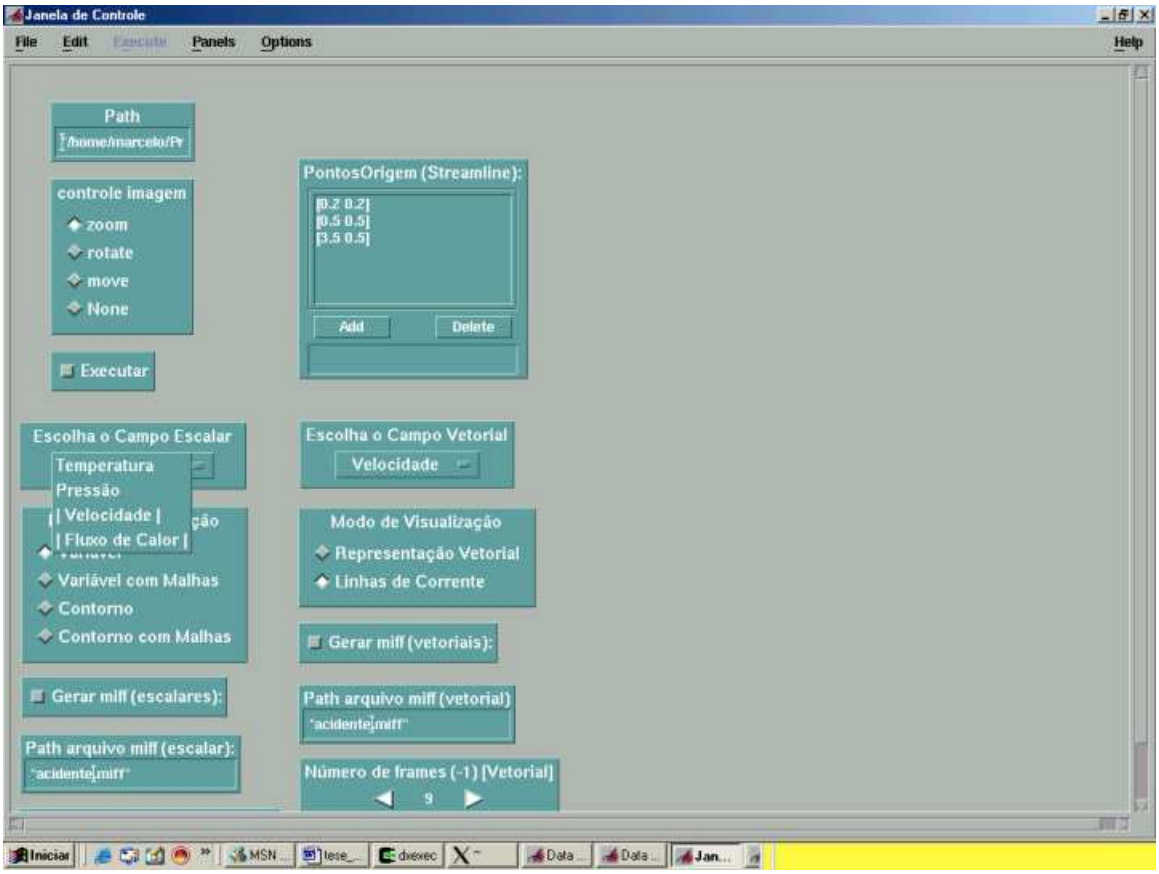

Figura 4.8: Variáveis escalares

permitem a produção de exibições mais robustas e completas.

No Visualizador_2D_NS, os vídeos são produzidos no formato de arquivos com extensão *miff*.

Para gerar arquivos de extensão *.miff*, primeiramente o usuário deve digitar na caixa *Path arquivo Miff* (escalar ou vetorial) o caminho dos diretório e o nome do arquivo que será gerado e possuirá extensão *miff*. Em seguida, é necessário determinar, na caixa *Número de Frames (-1)*, a quantidade de *frames* que serão carregados a partir dos arquivos nas *Janelas de Visualização de Imagens*, subtraído de uma unidade.

Durante o carregamento dos *frames* nas *Janelas de Visualização de Imagens*, o usuário deve pressionar o botão *Gerar Miff* (escalar ou vetorial). Nesse momento, todos os frames carregados nas *Janelas de Visualização de Imagens* passam a ser "fotografados" no arquivo de vídeo *miff*. Terminado o carregamento dos *frames* basta desativar o botão *Gerar Miff*.

Gerado o arquivo de vídeo *miff*, pode-se interpolar mais imagens entre as imagens já "fotografadas" dentro do arquivo. Isso dá maior consistência ao vídeo. Para tanto, é utilizado um programa de conversão disponível nas distribuições Linux. Basta digitar o comando abaixo no console para converter o arquivo *miff* em *mpeg* e introduzir as interpolações:

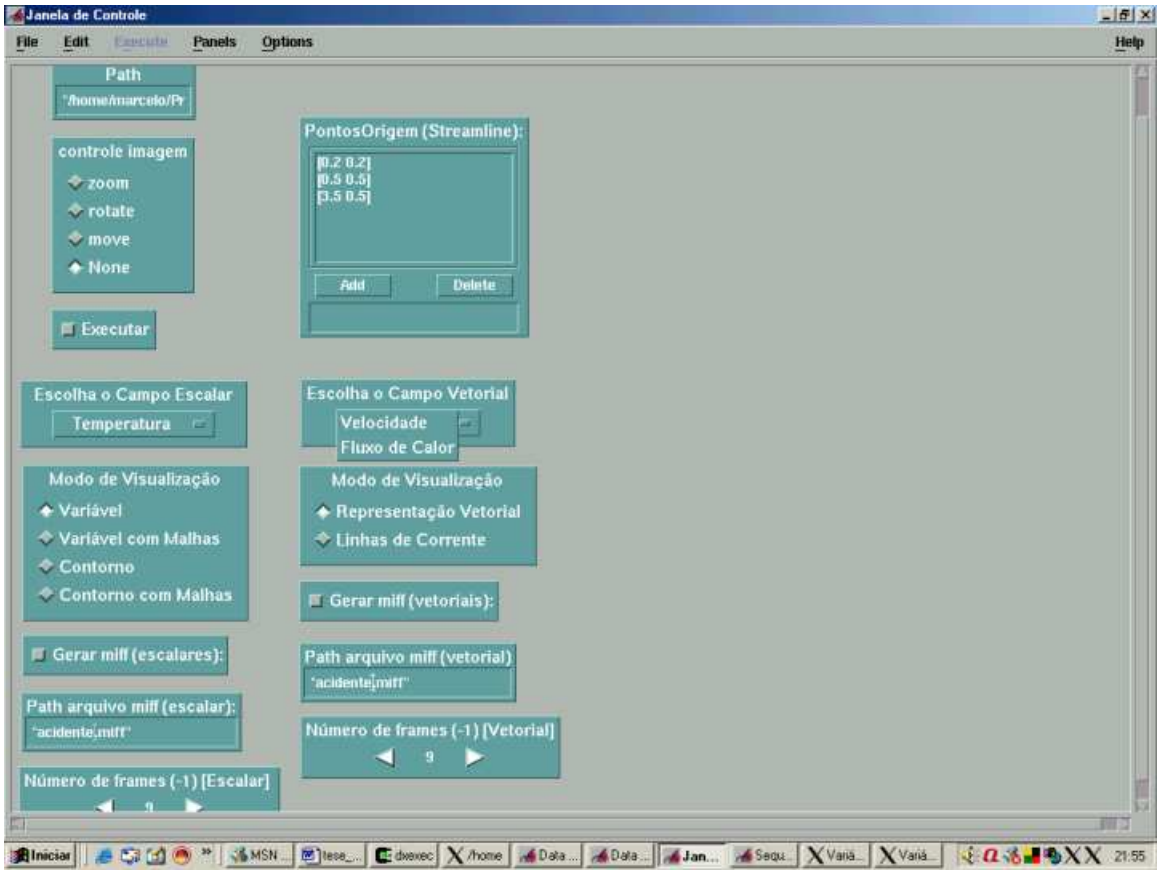

Figura 4.9: Variáveis Vetoriais

convert -delay 10.0 nomedovideo.miff nomedovideo.mpeg

Com esse comando, entre duas imagens do arquivo *miff* são interpoladas outras 8 imagens, perfazendo o total de 10. É isso o que significa a especificação *-delay 10.0* após o camando *convert*. Caso essa especificação fosse *-delay 5.0*, seriam interpoladas 3 imagens perfazendo o total de 5.

Gerado o arquivo *mpeg*, este pode ser executado em qualquer programa de exibição de vídeos.

No Visualizador_2D_NS existe um gerador de arquivos de extensão *miff* para a *Janela de Variável Escalar* e outra para a *Janela de Variável Vetorial*. Isso significa que o programa pode gerar dois arquivos de vídeo concomitantemente: um escalar e outro vetorial. Isso é possível porque a exibição das imagens das variáveis escalares e vetoriais ocorre simultaneamente e em janelas separadas.

4.2 VISUALIZADOR_3D_NS

Todas as funcionalidades descritas no VISUALIZADOR_2D_NS estão presentes nessa versão 3D. As diferenças em relação ao VISUALIZADOR_2D_NS estão nos mo-

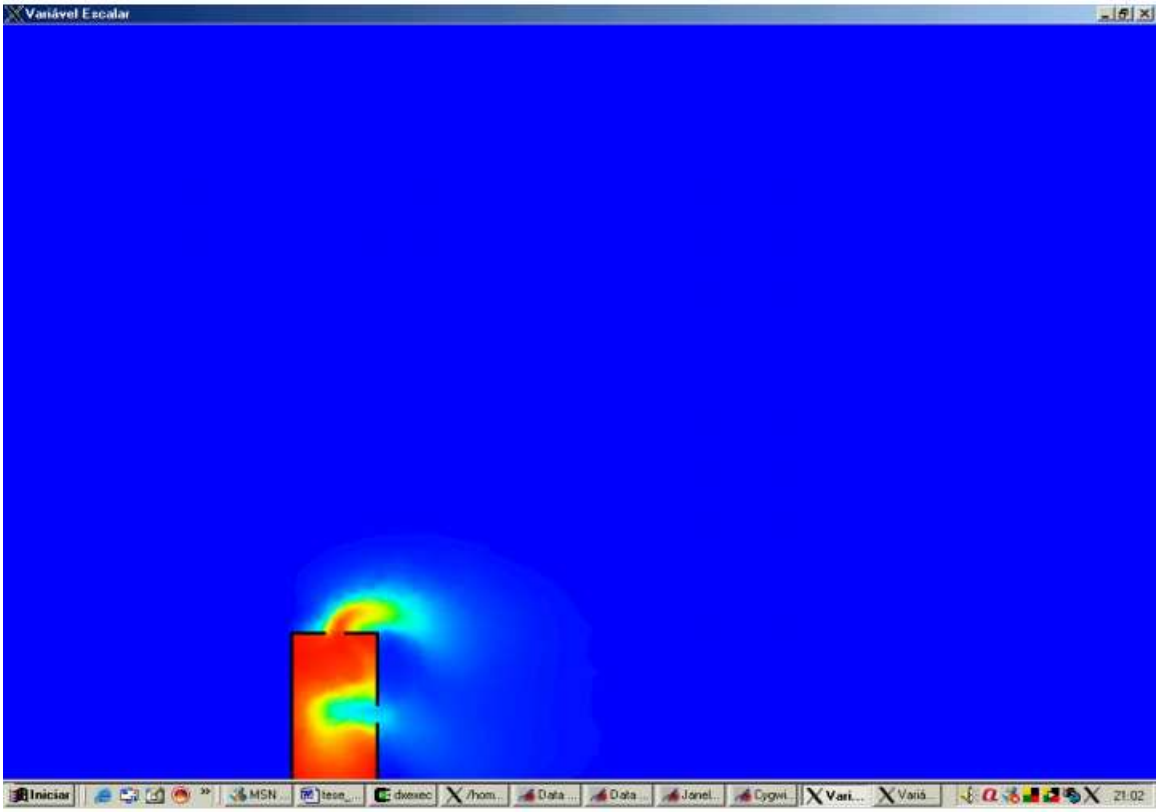

Figura 4.10: Representação da Variável Escalar

dos de visualização disponíveis na caixa *Modo de Visualização* das variáveis escalares (figura 4.16). Uma descrição das opções disponíveis é feita a seguir.

- **"Gelatina":** essa opção dá ao usuário a visão do campo escalar no volume tridimensional. Consegue-se visualizar o gradiente em toda a profundidade do volume (figura 4.17).
- **"Gelatina" com caixa:** a caixa é uma estrutura tridimensional que ajuda o usuário a situar espacialmente o volume. Ela é importante principalmente quando associada à opção gelatina para facilitar a localização dos limites do volume (figura 4.18).
- **Variável no contorno:** essa opção permite a visualização do campo escalar no contorno, ou superfície mais externa, do volume. Diferentemente da "gelatina", não é possível avaliar o gradiente em toda a profundidade do volume (figura 4.19).
- **Variável no contorno com malha:** permite a visualização da variável no contorno e da malha de elementos finitos na superfície do volume (figuras 4.20 e 4.21).

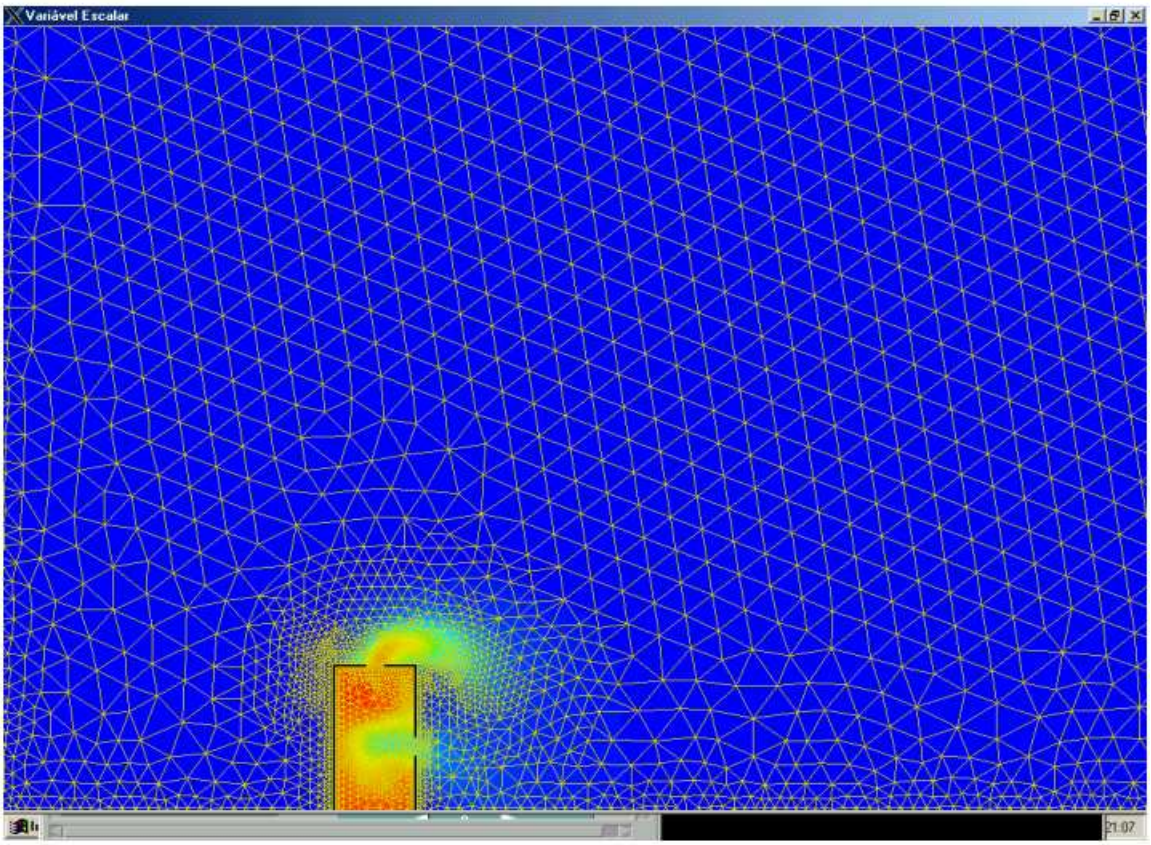

Figura 4.11: Representação da variável escalar com malha

- **Contorno:** Permite a visualização da superfície do volume (figura 4.22).
- **Contorno com malha:** Associa a imagem da superfície do volume e da malha de elementos finitos nessa superfície (figura 4.23).

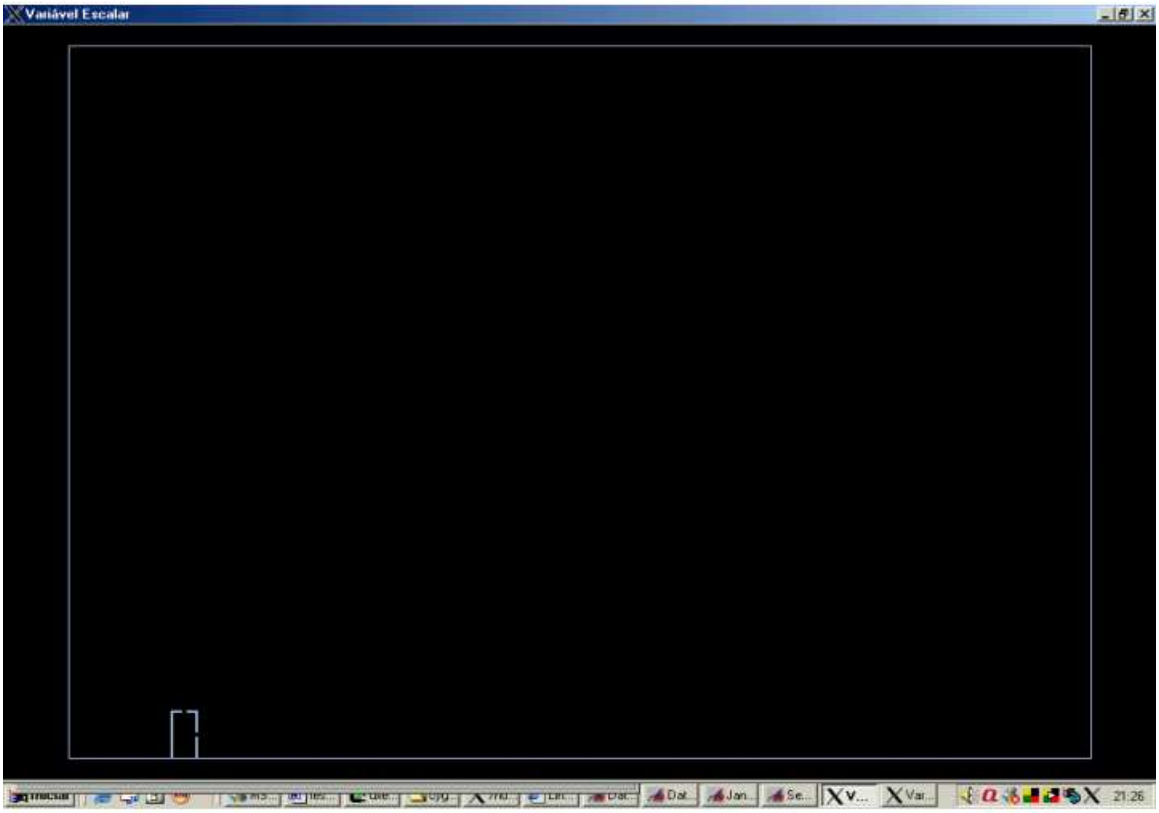

Figura 4.12: Contorno/domínio do problema

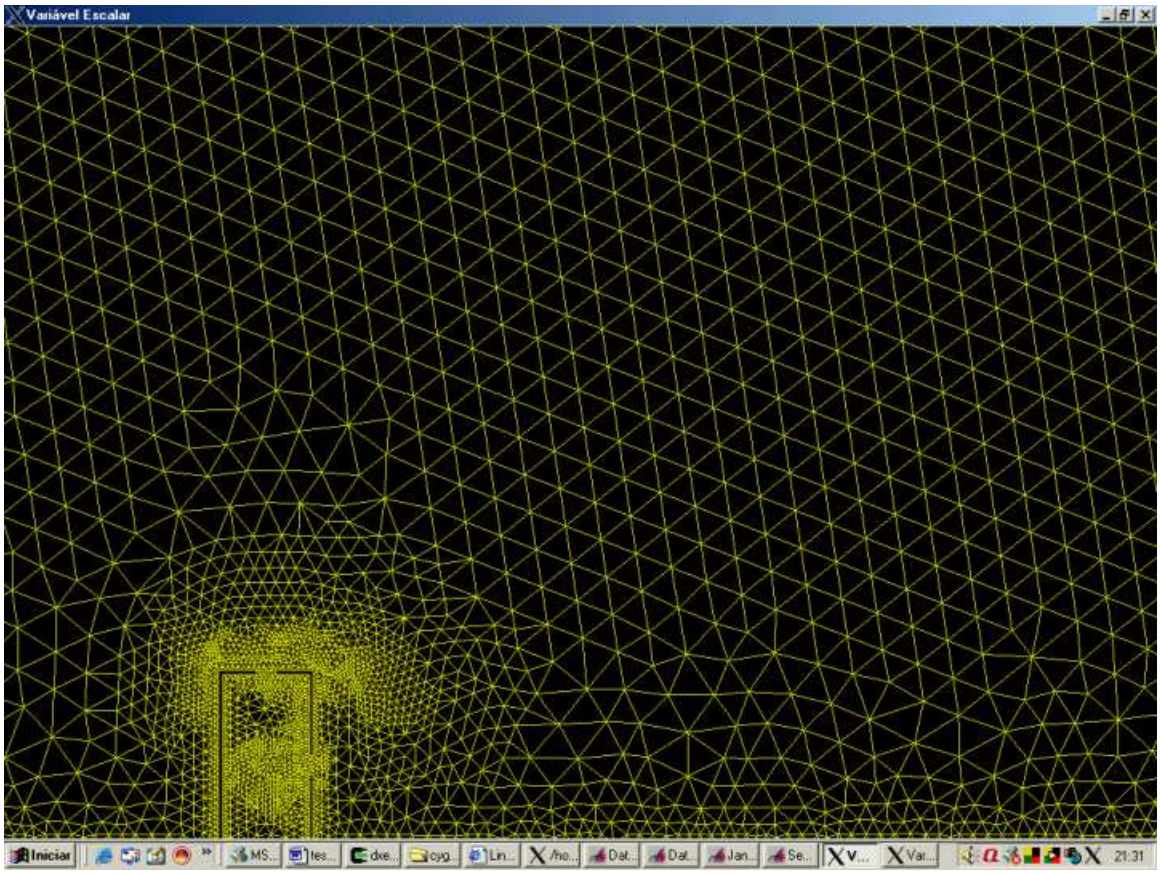

Figura 4.13: Contorno com malha

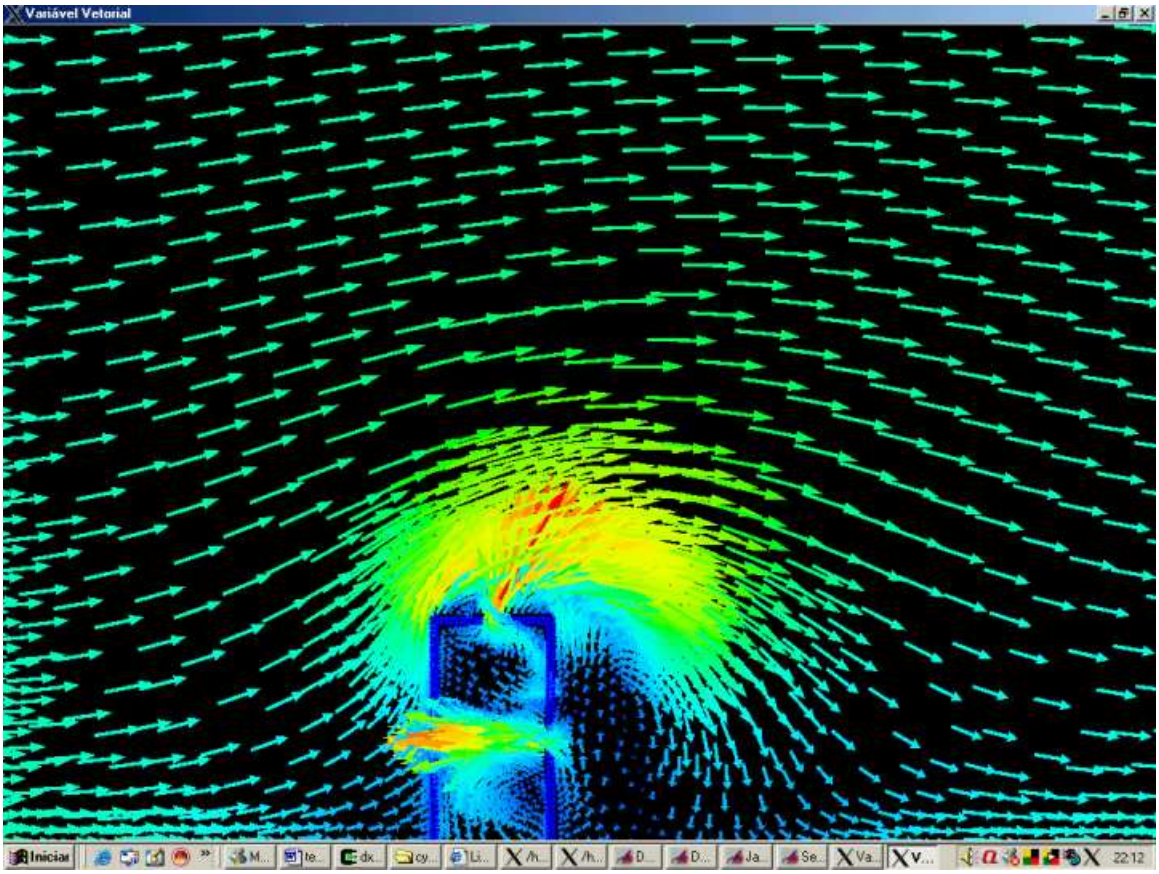

Figura 4.14: Variável vetorial - cores corresponde ao gradiente de intensidade

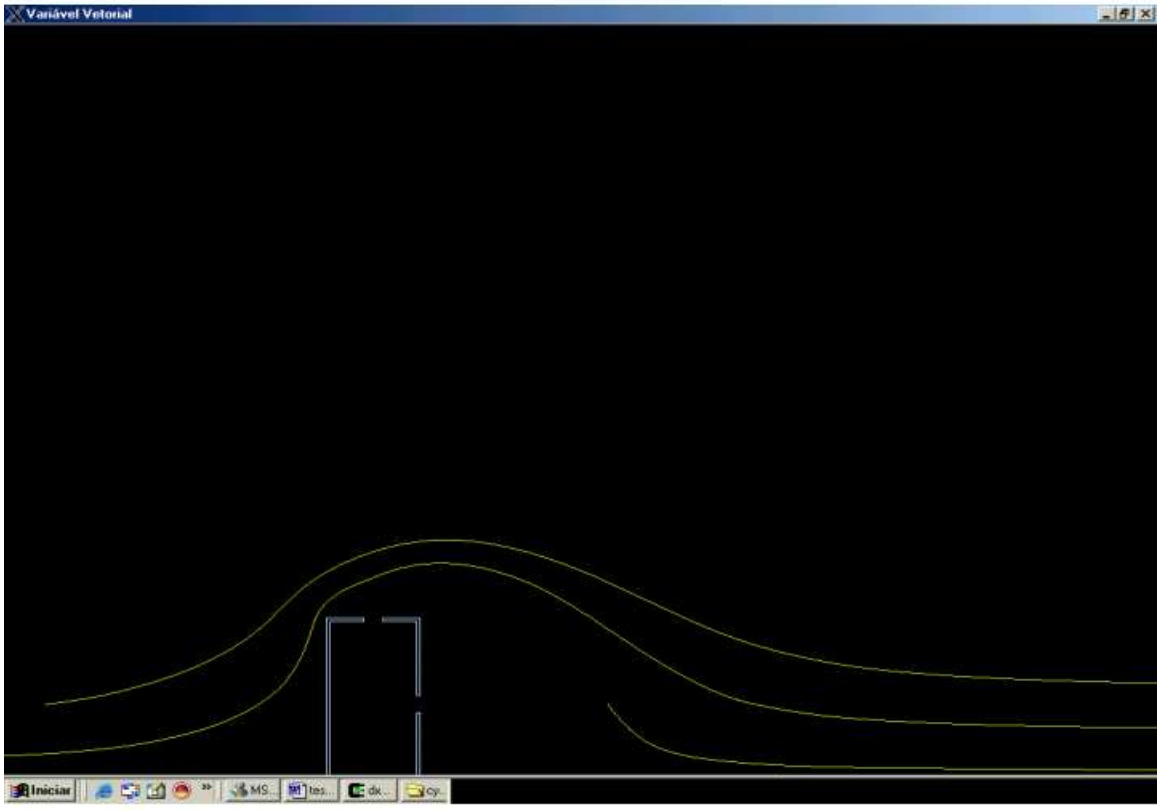

Figura 4.15: Linhas de corrente

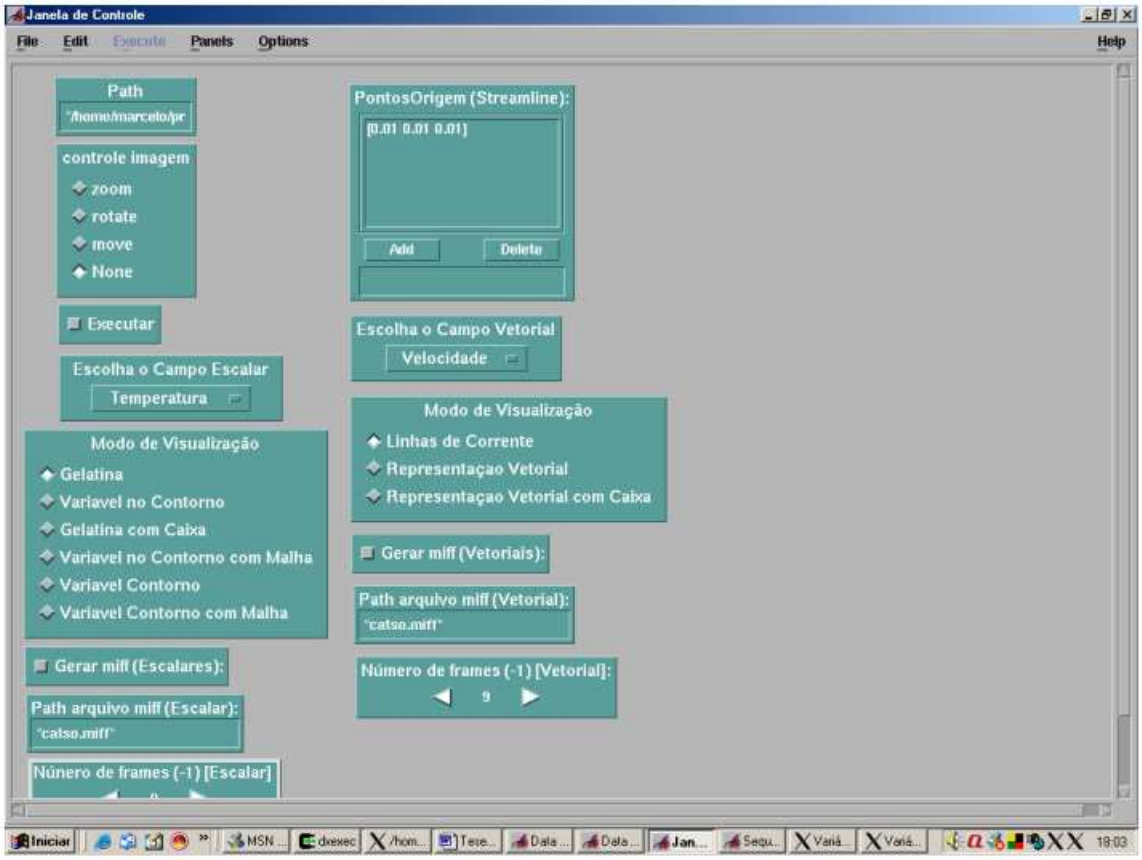

Figura 4.16: Painel de Controle do Visualizador_3D_NS.

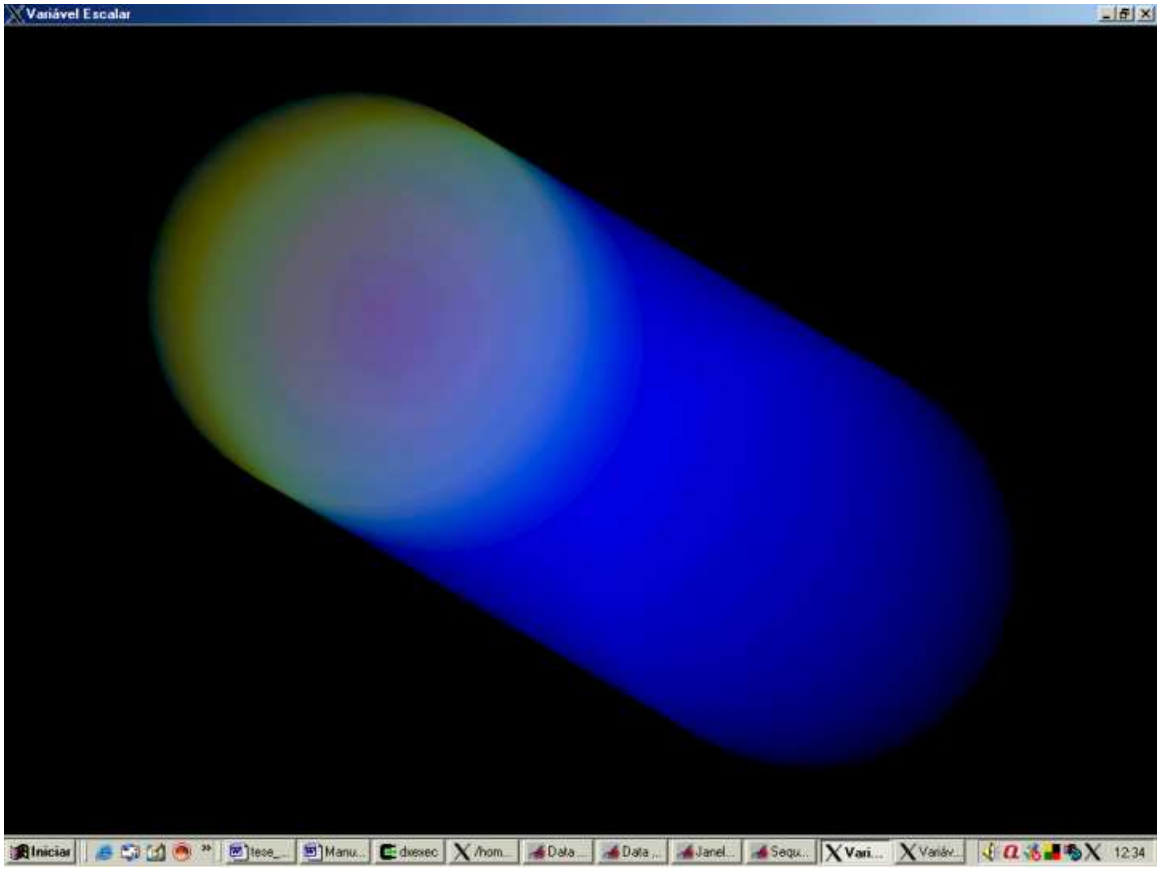

Figura 4.17: Modo de visualização gelatina no escoamento de um fluido por um tubo cilíndrico.

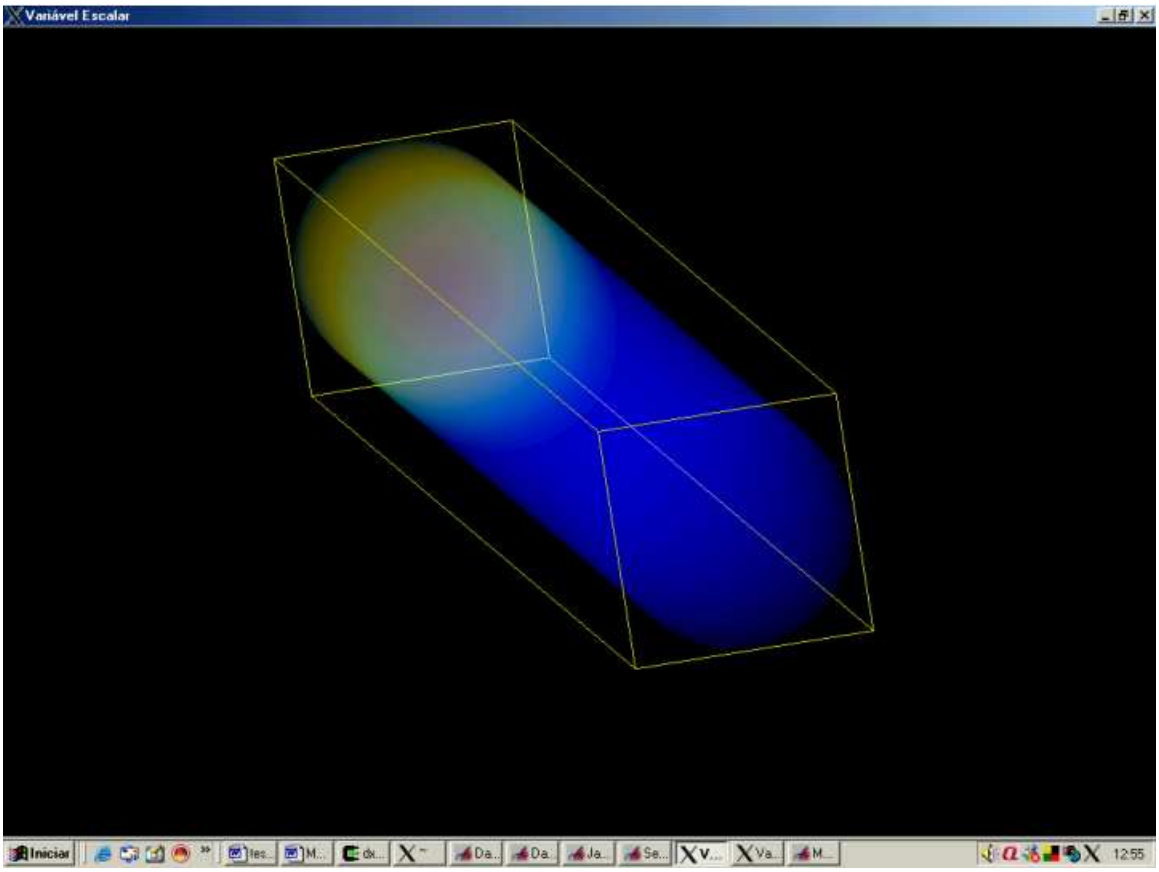

Figura 4.18: Modo de visualização gelatina com caixa no escoamento de um fluido por um tubo cilíndrico.

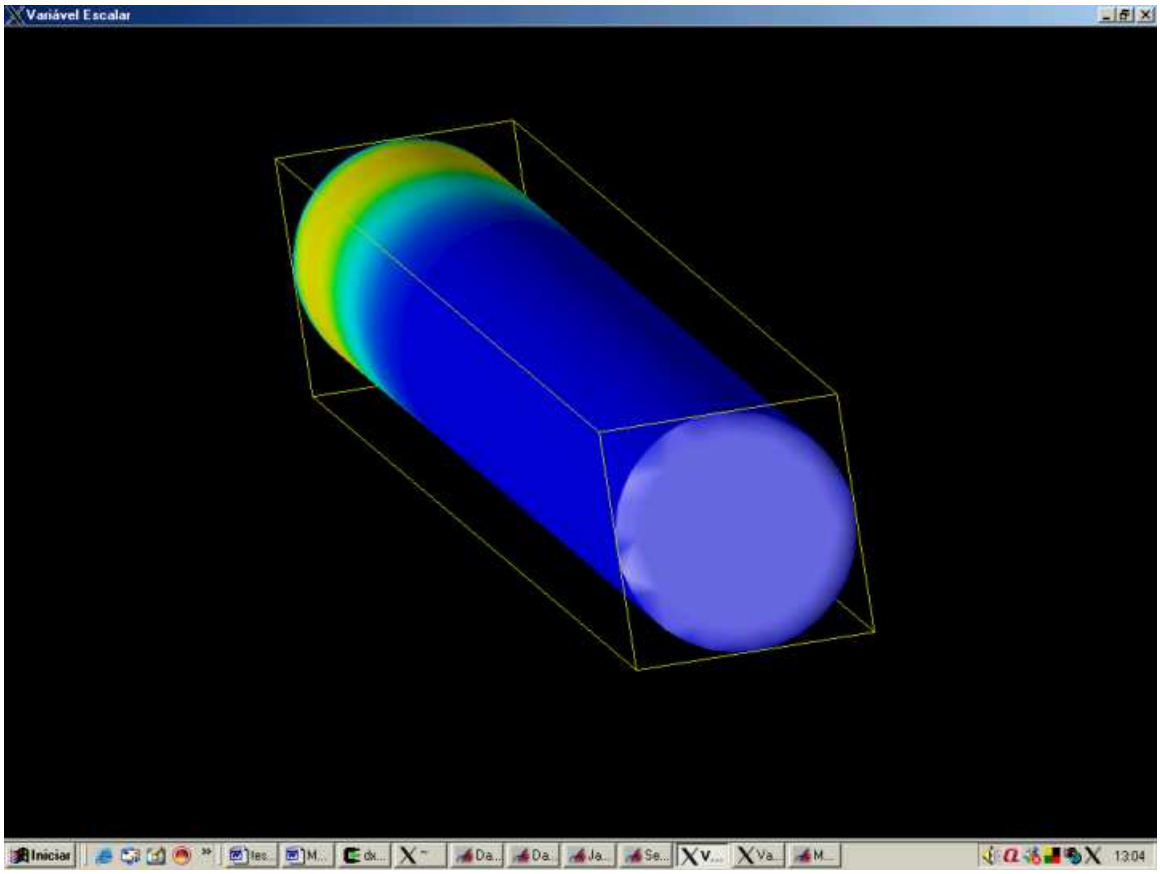

Figura 4.19: Variável no contorno - escoamento em cilindro

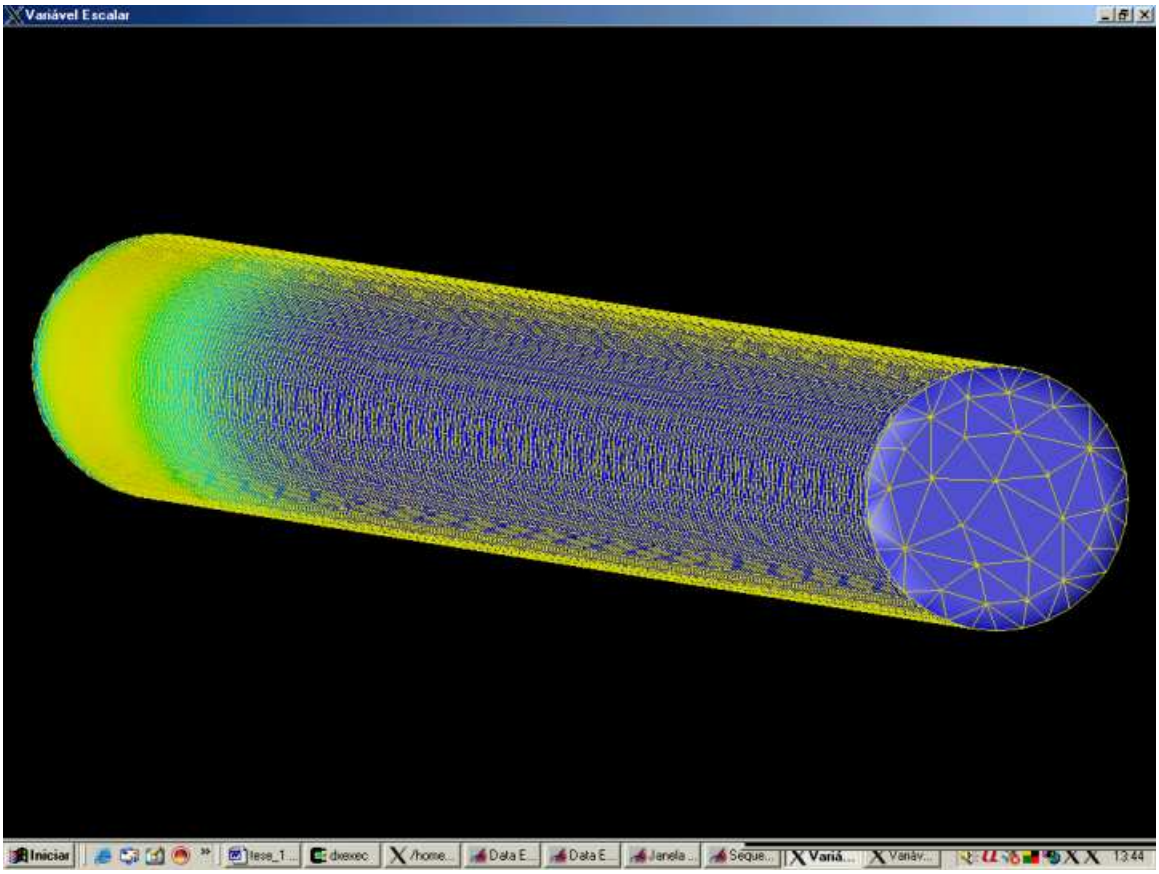

Figura 4.20: Variável no contorno com malha na superfície

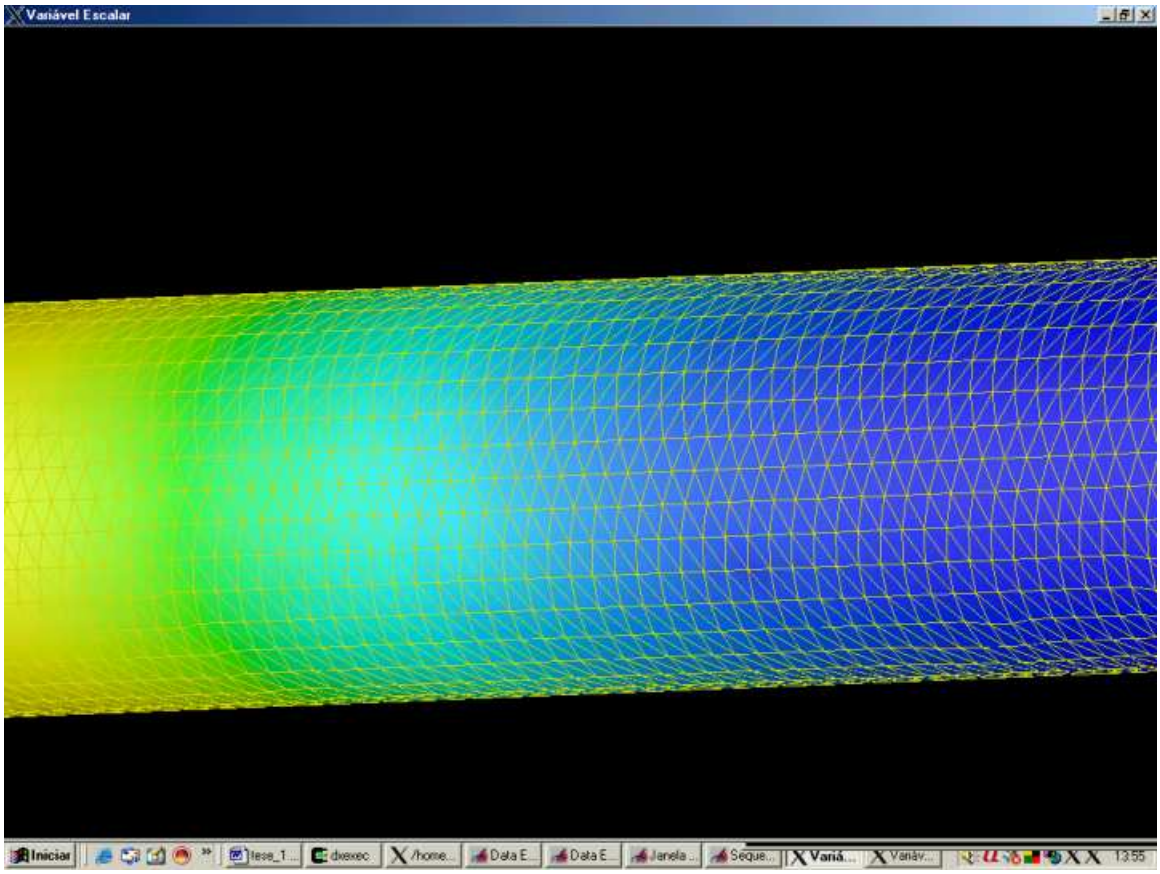

Figura 4.21: Detalhe da variável no contorno com malha na superfície

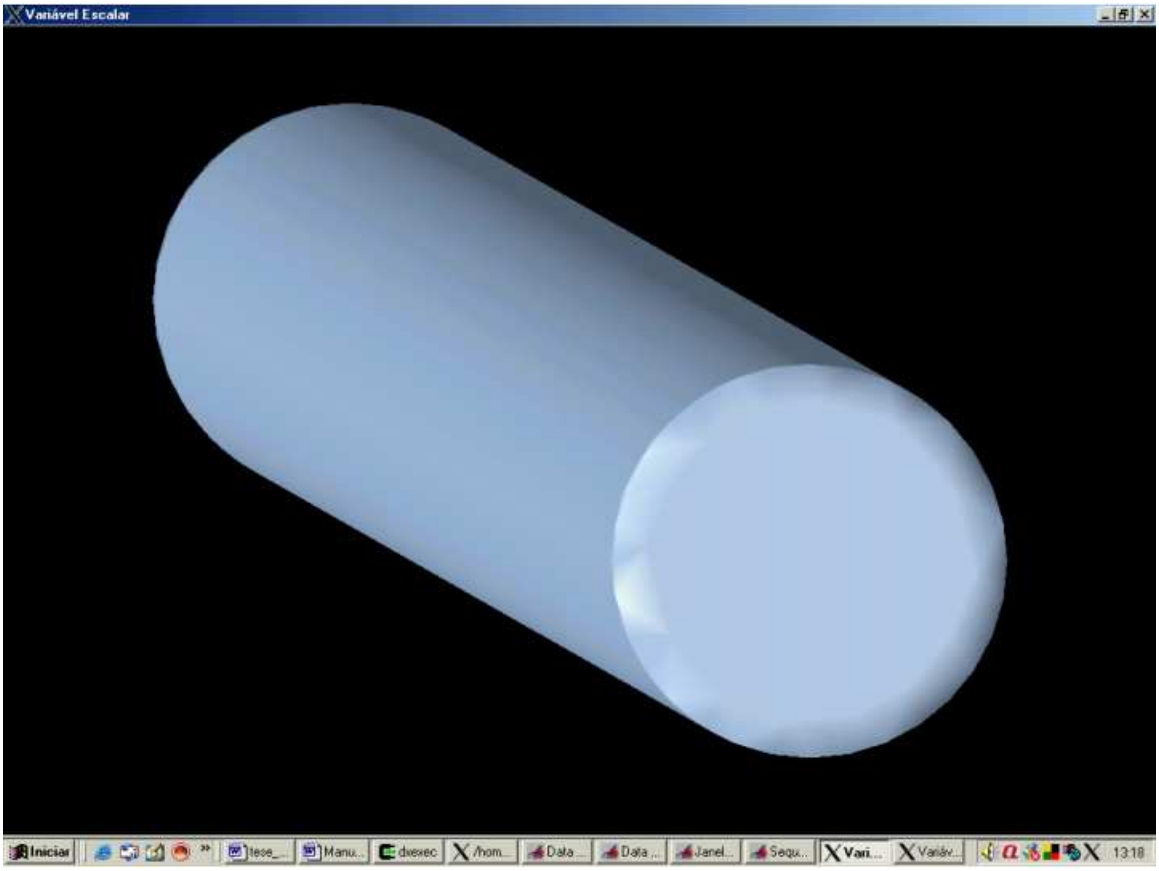

Figura 4.22: Superfície do volume - domínio da simulação

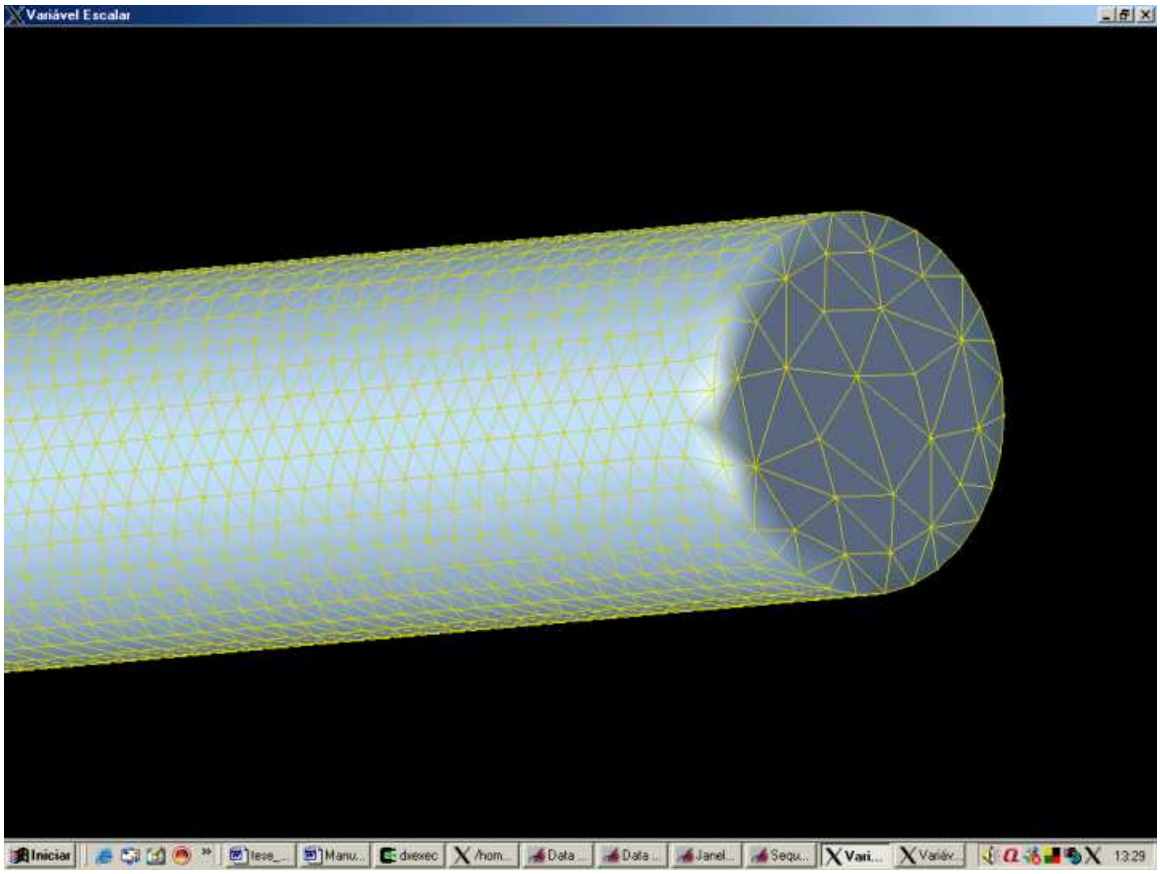

Figura 4.23: Superfície do volume com malha

Capítulo 5

Conclusões

Os programas construídos e entregues são ferramentas importantes para os trabalhos desenvolvidos no Laboratório de Computação Paralela do IEN. Fundamentais para o desenvolvimento de diversos trabalhos na área de CFD, que foram ou ainda encontram-se em andamento, esses programas atendem plenamente às necessidades acadêmicas, foram desenvolvidos a partir de códigos livres com baixíssimo custo.

As construções desses programas foram feitas em função das necessidades de hoje, mas sua estrutura permite melhorias e adaptações que possam ser essenciais para se criar novas aplicações ou introduzir recursos que venham a se tornar as necessidades de amanhã.

Restringindo-se à utilização do OpenDx no desenvolvimento de pós-processadores, muitas dessas melhorias e adaptações podem já ser vislumbradas hoje e são relacionadas a seguir:

- A utilização do módulo IMAGE diretamente no programa é um desafio para quem vier a desenvolver as futuras versões dos programas visuais. Esse módulo, responsável pela geração final da imagem, possui muitos recursos e exige uma complexa estruturação de outros módulos funcionais para ser substituído. Nos programas visuais desenvolvidos, sua utilização foi inviabilizada porque a necessidade de carregamento de arquivos diretamente pelo código para a geração das animações anulava os recursos de interação com a imagem oferecidos pelo módulo IMAGE. Esse problema foi contornado com a construção de outro arranjo, utilizando os módulos DISPLAY e SUPERVISESTATE. Contudo, outros recursos oferecidos pelo IMAGE não ficaram disponíveis.

- A versão atual dos programas conta com duas janelas de visualização: uma para variáveis escalares e outra para variáveis vetoriais. Uma sugestão para as próximas versões é desenvolver o programa e utilizar-se de uma única janela para ambas as variáveis. Isso simplificaria os diagramas de blocos pois eliminaria duplicações nos arranjos desenvolvidos.

- A interação do usuário com a imagem se dá, nas atuais versões dos programas, através do *mouse*. Uma melhoria importante seria desenvolver essa interação através do teclado. Isso daria mais agilidade e controle sobre as instruções interativas.

- Outra melhoria interessante de ser implementada é a construção de um recurso de multivisualização, em que o usuário possa assistir concomitantemente a mesma

simulação, sob pontos de vista diferentes (diferentes ângulos) ou visualizando diferentes variáveis. Isso permitiria ao pesquisador correlacionar os campos das diferentes variáveis por exemplo.

- A criação de animações em que o ponto de observação se modifique com o decorrer da simulação é outro recurso que poderá ser futuramente desenvolvido. Isso permitiria ao usuário determinar previamente "um passeio" pela animação deslocando-se de um ponto de interesse a outro para melhor observação. Esse recurso seria particularmente importante nas animações em 3 dimensões.

- Nas simulações que se utilizam dos diagramas de cores, para observação do gradiente dos campos, pode ser desenvolvido um recurso de escala que mostraria, no decorrer da simulação, a correspondente variação de valores, dos mais baixos (em azul) aos mais altos (em vermelho). Isso ajudaria o pesquisador tanto na análise da simulação quanto na impressão de *screenshots* para publicações científicas.

Em resumo, esse texto é muito útil a todos os pesquisadores que necessitam construir programas de visualização, pois embora não substitua os manuais oficiais, é uma fonte simplificada e direta para aquisição do conhecimento necessário à aplicação e utilização do OpenDx.

Referências Bibliográficas

[1] NASCIMENTO,H.A.D.;FERREIRA,C.B.R. 2005. "Visualização de Informações - Uma Abordagem Prática", *XXV Congresso Brasileiro de Computação - XXVI Jornadas de Atualização em Informática* p.1262-1278

[2] OpenDX Site
http://www.opendx.org/about.html
Consultado em 26/05/2007;

[3] CARD,S.K.;MACKINLAY,J.D.;SHNEIDERMAN,B.;CARD M. 1999. "Readings in Information Visualization: Using Vision to Think. ", *Morgan Kaufmann Series in Interactive Technologies*

[4] BATTAIOLA,A.L.;SOARES,L.P. 1998 "Estudo e Uso Exploratório de Ferramentas de Visualização Científica", *VII Semana de Informatica da UFBA*

[5] SOLER,J.G.M;ALMEIDA,J.P.M. 2003 "Uma alternativa para pré e pós processador gráfico por meio de arquivos de texto para programas de elementos finitos", *Mecánica Computacional* v.22

[6] QUARESMA,J.E. 2004 "Desenvolvimento de um pós-processador para visualização de resultados de simulação numérica em aqüíferos", Dissertação - Escola de Engenharia de São Carlos (EESC), Hidráulica e Saneamento.

[7] NIELSON, G.M. 1991 "Visualization in Scientific and Engineering Computation", *Computer*, 24(9), p. 58-66.

[8] CFD Projeto SinFlow
http://www.sinmec.ufsc.br/cfd/doc/pt/cfd-classes/introduction/history.html
Consultado em 23/05/2007;

[9] OpenDX User's Guide Release 3 Modification 1 Version 4
http://opendx.npaci.edu/docs/pdf/userguide.pdf

[10] OpenDX User's Reference Release 1 Modification 4 Version 3
http://opendx.npaci.edu/docs/pdf/userref.pdf

[11] OpenDX Program's Reference Release 1 Modification 4 Version 3
http://opendx.npaci.edu/docs/pdf/userguide.pdf

[12] Introduction to Open DX Cornell University - Center for advancer computing
www.tc.cornell.edu/Services/Education/Topics/OpenDX/Intro/2.1+History.html
Consultado em 26/05/2007;

[13] MANSSOUR,I.H. , COHEN,M. , 2006 "Introdução à Computação Gráfica", *Revista de Informática Teórica e Aplicada* v.13 n. 2

[14] ELLERSHAW,S.;OUDSHOORN,M. 1994 "Program Visualization - The State of the Art", Department of Computer Science, University of Adelaide.
http://citeseer.ist.psu.edu/ellershaw94program.html

[15] PRICE,A.;SMALL,I.S.;BAECKER,M. 1992 "Taxonomy of software visualization", *In Proceedings of the 25th Hawaii International Conference on System Sciences*, v.2, p.597-606

[16] SCV site
http://scv.bu.edu/SCV/Graphics/avs.html
Consultado em 26/05/2007;

[17] DE SAMPAIO, P.A.B., 2005. "A finite element formulation for transient incompressible viscous flows stabilized by local time-step", *Computer Methods in Applied Mechanics and Engineering*, 194, p. 2095-2108.

[18] DE SAMPAIO, P.A.B., 2006. "A stabilized finite element method for incompressible flow and heat transfer: A natural derivation based on the use of local time-steps´´, *Computer Methods in Applied Mechanics and Engineering*, v. 195, n. 44-47, p. 6177-6190.

[19] DE SAMPAIO, P.A.B.; JUNIOR, M.A.G.; Lapa, C.M.F., 2008. "A CFD approach to the atmospheric dispersion of radionuclides in the vicinity of NPPs", *Nuclear Engineering and Design* (aceito, a aparecer).

[20] GUIMARÃES C.S. ; DE SAMPAIO P.A.B, 2005. "Utilização da Interface Gráfica OpenDX para Visualização de Simulações Computacionais", relatório interno LCP/IEN/CNEN.

[21] GONÇALVES JUNIOR M.A., 2006. "Aplicação da fluido dinâmica computacional à dispersão atmosférica de radionuclídeos na vizinhança de uma central nuclear", Dissertação de mestrado - IEN/CNEN.

Apêndice A

Arquivos DX

A.1 Exemplo de arquivo de dados escalares em domínio tridimensional

object 1 class array type float rank 1 shape 3 items 10739 msb
data file Coordenadas.27.dx,0
attribute "dep" string "positions"

object 2 class array type int rank 1 shape 4 items 40789 msb
data file Elementos.27.dx,0
attribute "element type" string "tetrahedra"
attribute "ref" string "positions"

object 3 class array type float rank 0 items 10739 data follows

0.73440
0.73323
0.73273
0.73844
0.75165
0.73577
0.73985
0.74630
0.76138
0.73870
. . .
0.00000
0.00000
0.00000
0.00000
0.00000
0.00000
0.00000
0.00000

attribute “dep” string “positions”
object “irregular positions irregular connections file” class field
component “positions” value 1
component “connections” value 2
component “data” value 3
end

A.2 Arquivo de dados vetoriais em domínio bidimensional

Exemplo 1

object 1 class array type float rank 1 shape 2 items 17140 msb
data file Coordenadas.13.dx,0
attribute “dep” string “positions”

object 2 class array type int rank 1 shape 3 items 32697 msb
data file Elementos.13.dx,0
attribute “element type” string “triangles”
attribute “ref” string “positions”

object 3 class array type float rank 1 shape 2 items 17140 data follows

1.00044	-0.00042
1.00043	-0.00029
1.00053	-0.00048
1.00055	-0.00036
1.00041	-0.00007
1.00073	-0.00049
1.00051	-0.00010
1.00089	-0.00040
1.00076	-0.00014
1.00050	0.00018
1.00103	-0.00053
1.00065	0.00013
1.00110	-0.00019
1.00059	0.00038
1.00121	-0.00045
1.00098	0.00009
1.00132	-0.00058
1.00086	0.00037
. . .	. . .
0.50180	0.18790
0.93676	0.29629
0.70724	0.24543
0.83827	-0.79920
0.83843	-0.80109

attribute “dep” string “positions”
object “irregular positions irregular connections file” class field
component “positions” value 1
component “connections” value 2
component “data” value 3
end

A.3 Exemplo de arquivo com dependência das conexões

object 1 class array type float rank 1 shape 2 items 17687 msb
data file Coordenadas.32.dx,0
attribute “dep” string “positions”

object 2 class array type int rank 1 shape 3 items 33781 msb
data file Elementos.32.dx,0
attribute “element type” string “triangles”
attribute “ref” string “positions”

object 3 class array type float rank 1 shape 2 items 33781 data follows

-0.33523	-0.27570
0.87891	22.28931
-0.02047	0.05032
0.03685	-0.36390
-0.34204	-0.20992
-0.04604	0.17913
-0.35940	-0.32418
-0.09943	0.29478
-0.06812	0.99140
-0.38806	-0.16945
. . .	. . .
0.00618	-0.02048
0.00444	-0.02189
0.00801	-0.02479
0.00816	-0.02481
0.00207	-0.02367
-0.02869	0.43543

attribute “dep” string “connections”
object “irregular positions irregular connections file” class field
component “positions” value 1
component “connections” value 2
component “data” value 3
end

A.4 Arquivo de dado vetorial em domínio tridimensional

object 1 class array type float rank 1 shape 3 items 10739 msb
data file Coordenadas.22.dx,0
attribute “dep” string “positions”

object 2 class array type int rank 1 shape 4 items 40789 msb
data file Elementos.22.dx,0
attribute “element type” string “tetrahedra”
attribute “ref” string “positions”

object 3 class array type float rank 1 shape 3 items 10739 data follows

0.00000	0.00000	0.00000
0.00000	0.00000	0.00000
0.00000	0.00000	0.00000
0.00000	0.00000	0.00000
0.00000	0.00000	0.81342
0.00000	0.00000	0.00000
0.00000	0.00000	0.00000
0.00000	0.00000	0.74878
0.00133	0.00187	0.87251
0.00000	0.00000	0.00000
0.00000	0.00000	0.00000
0.00000	0.00000	0.00000
0.00000	0.00000	0.00000
0.00000	0.00000	0.00000
0.00000	0.00000	0.00000
0.00000	0.00000	0.79639
0.00000	0.00000	0.00000
0.00000	0.00000	0.00000
0.00000	0.00000	1.47202
-0.00215	0.00090	0.79159
. . .	. . .	. . .
0.00000	0.00000	1.00000
0.00000	0.00000	1.00000
0.00000	0.00000	1.00000
0.00000	0.00000	1.00000

attribute “dep” string “positions”
object “irregular positions irregular connections file” class field
component “positions” value 1
component “connections” value 2
component “data” value 3
end

Apêndice B

Exemplo de *Script* de Programa Visual

```
    // time: Tue Apr 29 09:07:50 1997
// // version: 3.1.1 (format), 3.1.4 (DX)
// // // MODULE main
// // comment: This visual program computes the gradient of a scalar field on a plane. The gradient is autocolored, with
colors representing the magnitude of the gradient,
and glyphs (arrows) are drawn to show the direction of the gradient field.
// comment:
// comment: The gradient measures the direction and magnitude of change of a scalar field.
// workspace: width = 417, height = 593
// layout: snap = 0, width = 50, height = 50, align = UL
// macro main(

    ) -> (

    )

    //
    // node Import[1]: x = 97, y = 57, inputs = 6, label = Import
// input[1]: defaulting = 0, visible = 1, type = 32, value = "rainwater"
//
main_Import_1_out_1 =
Import( main_Import_1_in_1,

main_Import_1_in_2,

main_Import_1_in_3,

main_Import_1_in_4,

main_Import_1_in_5,

main_Import_1_in_6
) [instance: 1,
cache: 1];
// // node MapToPlane[1]: x = 116, y = 156, inputs = 3, label = MapToPlane
// input[2]: defaulting = 1, visible = 1, type = 8, value = [1000 1000 10000]
// main_MapToPlane_1_out_1 =
MapToPlane(
main_Import_1_out_1,
main_MapToPlane_1_in_2,
main_MapToPlane_1_in_3
) [instance: 1, cache: 1];
// // node Gradient[1]: x = 159, y = 243, inputs = 2, label = Gradient
// main_Gradient_1_out_1 =
Gradient(
main_MapToPlane_1_out_1,
main_Gradient_1_in_2
) [instance: 1, cache: 1];
// // node AutoColor[1]: x = 115, y = 333, inputs = 10, label = AutoColor
// main_AutoColor_1_out_1,
main_AutoColor_1_out_2 =
AutoColor(
main_Gradient_1_out_1,
main_AutoColor_1_in_2,
main_AutoColor_1_in_3,
main_AutoColor_1_in_4,
main_AutoColor_1_in_5,
main_AutoColor_1_in_6,
main_AutoColor_1_in_7,
main_AutoColor_1_in_8,
main_AutoColor_1_in_9,
main_AutoColor_1_in_10 ) [instance: 1,
cache: 1];
// // node Sample[1]: x = 47, y = 429,inputs = 2, label = Sample
// input[2]: defaulting = 0,visible = 1, type = 1,value = 500
```

```
// main_Sample_1_out_1 =
Sample(
main_AutoColor_1_out_1,
main_Sample_1_in_2 ) [instance: 1,
cache: 1];
// // node AutoGlyph[1]: x = 137,y = 430, inputs = 7,label = AutoGlyph
// input[4]: defaulting = 0,visible = 1, type = 5, value = .4
/ main_AutoGlyph_1_out_1 =
AutoGlyph(
main_Sample_1_out_1,
main_AutoGlyph_1_in_2,
main_AutoGlyph_1_in_3,
main_AutoGlyph_1_in_4,
main_AutoGlyph_1_in_5,
main_AutoGlyph_1_in_6,
main_AutoGlyph_1_in_7
) [instance: 1,
cache: 1];
// // node Collect[1]: x = 299, y = 426, inputs = 2, label = Collect // main_Collect_1_out_1 =
Collect(
main_AutoGlyph_1_out_1,
main_AutoColor_1_out_1
) [instance: 1,
cache: 1];
// // node Image[2]: x = 345, y = 531, inputs = 48, label = Image
// input[1]: defaulting = 0, visible = 0, type = 67108863, value = "Image_2"
// input[4]: defaulting = 0, visible = 0, type = 1, value = 1
// input[5]: defaulting = 0, visible = 0, type = 8, value = [30711.1 8251.01 27000]
// input[6]: defaulting = 0, visible = 0, type = 8, value = [30711.1 8251.01 281292]
// input[7]: defaulting = 0, visible = 0, type = 5, value = 15330.9
// input[8]: defaulting = 0, visible = 0, type = 1, value = 640
// input[9]: defaulting = 0, visible = 0, type = 5, value = 0.75
// input[10]: defaulting = 0, visible = 0, type = 8, value = [0 1 0]
// input[11]: defaulting = 1, visible = 0, type = 5, value = 3.45323
// input[12]: defaulting = 0, visible = 0, type = 1, value = 0
// input[14]: defaulting = 0, visible = 0, type = 1, value = 1
// input[19]: defaulting = 0, visible = 0, type = 3, value = 0
// input[29]: defaulting = 1, visible = 0, type = 3, value = 0
// depth: value = 24
// window: position = (0.3742,0.0596), size = 0.5109x0.5117
// internal caching: 1
// main_Image_2_out_1,
main_Image_2_out_2,
main_Image_2_out_3 =
Image(
main_Image_2_in_1,
main_Collect_1_out_1,
main_Image_2_in_3,
main_Image_2_in_4,
main_Image_2_in_5,
main_Image_2_in_6,
main_Image_2_in_7,
main_Image_2_in_8,
main_Image_2_in_9,
main_Image_2_in_10,
main_Image_2_in_11,
main_Image_2_in_12,
main_Image_2_in_13,
main_Image_2_in_14,
main_Image_2_in_15,
main_Image_2_in_16,
main_Image_2_in_17,
main_Image_2_in_18,
main_Image_2_in_19,
main_Image_2_in_20,
main_Image_2_in_21,
main_Image_2_in_22,
main_Image_2_in_23,
main_Image_2_in_24,
main_Image_2_in_25,
main_Image_2_in_26,
main_Image_2_in_27,
main_Image_2_in_28,
main_Image_2_in_29,
main_Image_2_in_30,
main_Image_2_in_31,
main_Image_2_in_32,
main_Image_2_in_33,
main_Image_2_in_34,
main_Image_2_in_35,
main_Image_2_in_36,
main_Image_2_in_37,
main_Image_2_in_38,
main_Image_2_in_39,
main_Image_2_in_40,
main_Image_2_in_41,
main_Image_2_in_42,
main_Image_2_in_43,
main_Image_2_in_44,
main_Image_2_in_45,
main_Image_2_in_46,
main_Image_2_in_47,
main_Image_2_in_48 ) [instance: 2, cache: 1];
// network: end of macro body CacheScene("Image_2", main_Image_2_out_1, main_Image_2_out_2);

main_Import_1_in_1 = "rainwater";
main_Import_1_in_2 = NULL;
main_Import_1_in_3 = NULL;
main_Import_1_in_4 = NULL;
```

```
main_Import_1_in_5 = NULL;
main_Import_1_in_6 = NULL;
main_Import_1_out_1 = NULL;
main_MapToPlane_1_in_2 = NULL;
main_MapToPlane_1_in_3 = NULL;
main_MapToPlane_1_out_1 = NULL;
main_Gradient_1_in_2 = NULL;
main_Gradient_1_out_1 = NULL;
main_AutoColor_1_in_2 = NULL;
main_AutoColor_1_in_3 = NULL;
main_AutoColor_1_in_4 = NULL;
main_AutoColor_1_in_5 = NULL;
main_AutoColor_1_in_6 = NULL;
main_AutoColor_1_in_7 = NULL;
main_AutoColor_1_in_8 = NULL;
main_AutoColor_1_in_9 = NULL;
main_AutoColor_1_in_10 = NULL;
main_AutoColor_1_out_1 = NULL;
main_Sample_1_in_2 = 500;
main_Sample_1_out_1 = NULL;
main_AutoGlyph_1_in_2 = NULL;
main_AutoGlyph_1_in_3 = NULL;
main_AutoGlyph_1_in_4 = .4;
main_AutoGlyph_1_in_5 = NULL;
main_AutoGlyph_1_in_6 = NULL;
main_AutoGlyph_1_in_7 = NULL;
main_AutoGlyph_1_out_1 = NULL;
main_Collect_1_out_1 = NULL;
macro Image(
id,
object,
where,
useVector,
to,
from,
width,
resolution,
aspect,
up,
viewAngle,
perspective,
options,
buttonState = 1,
buttonUpApprox = "none",
buttonDownApprox = "none",
buttonUpDensity = 1,
buttonDownDensity = 1,
renderMode = 0,
defaultCamera,
reset,
backgroundColor,
throttle,
RECenable = 0,
RECfile,
RECformat,
RECresolution,
RECaspect,
AAenable = 0,
AAlabels,
AAticks,
AAcorners,
AAframe,
AAadjust,
AAcursor,
AAgrid,
AAcolors,
AAannotation,
AAlabelscale,
AAfont,
interactionMode,
title,
AAxTickLocs,
AAyTickLocs,
AAzTickLocs,
AAxTickLabels,
AAyTickLabels,
AAzTickLabels) -> (
object,
camera,
where)

ImageMessage(
id,
backgroundColor,
throttle,
RECenable,
RECfile,
RECformat,
RECresolution,
RECaspect,
AAenable,
AAlabels,
AAticks,
AAcorners,
AAframe,
AAadjust,
AAcursor,
AAgrid,
AAcolors,
```

```
AAannotation,
AAlabelscale,
AAfont,
AAxTickLocs,
AAyTickLocs,
AAzTickLocs,
AAxTickLabels,
AAyTickLabels,
AAzTickLabels,
interactionMode,
title,
renderMode,
buttonUpApprox,
buttonDownApprox,
buttonUpDensity,
buttonDownDensity) [instance: 1, cache: 1];
autoCamera =
AutoCamera( object,
"front",
object,
resolution,
aspect,
      ,0
,
perspective,
viewAngle,
backgroundColor) [instance: 1, cache: 1];
realCamera =
Camera(
to,
from,
width,
resolution,
aspect,
up,
perspective,
viewAngle,
backgroundColor) [instance: 1, cache: 1];
coloredDefaultCamera =
UpdateCamera(defaultCamera,
background=backgroundColor) [instance: 1, cache: 1];
nullDefaultCamera =
Inquire(defaultCamera,
"is null + 1") [instance: 1, cache: 1];
resetCamera =
Switch(
nullDefaultCamera,
coloredDefaultCamera,
autoCamera) [instance: 1, cache: 1];
resetNull =
Inquire(
reset,
"is null + 1") [instance: 2, cache: 1];
reset =
Switch(
resetNull,
reset,
0) [instance: 2, cache: 1];
whichCamera =
Compute(
"($0 != 0 || $1 == 0) ? 1 : 2",
reset,
useVector) [instance: 1, cache: 1];
camera = Switch(
whichCamera,
resetCamera,
realCamera) [instance: 3, cache: 1];
AAobject =
AutoAxes(
object,
camera,
AAlabels,
AAticks,
AAcorners,
AAframe,
AAadjust,
AAcursor,
AAgrid,
AAcolors,
AAannotation,
AAlabelscale,
AAfont,
AAxTickLocs,
AAyTickLocs,
AAzTickLocs,
AAxTickLabels,
AAyTickLabels,
AAzTickLabels) [instance: 1, cache: 1];
switchAAenable = Compute("$0+1",
AAenable) [instance: 2, cache: 1];
object = Switch(
switchAAenable,
object,
AAobject) [instance:4, cache: 1];
SWapproximation_options =
Switch(
buttonState,
buttonUpApprox,
buttonDownApprox) [instance: 5, cache: 1];
```

```
SWdensity_options =
Switch(
buttonState,
buttonUpDensity,
buttonDownDensity) [instance: 6, cache: 1];
HWapproximation_options =
Format(
"%s,
%s",
buttonDownApprox,
buttonUpApprox) [instance: 1, cache: 1];
HWdensity_options =
Format(
"%d,%d",
buttonDownDensity,
buttonUpDensity) [instance: 2, cache: 1];
switchRenderMode = Compute(
"$0+1",
renderMode) [instance: 3, cache: 1];
approximation_options = Switch(
switchRenderMode,
SWapproximation_options,
HWapproximation_options) [instance: 7, cache: 1];
density_options = Switch(
switchRenderMode,
SWdensity_options,
HWdensity_options) [instance: 8, cache: 1];
renderModeString = Switch(
switchRenderMode,
"software",
"hardware")[instance: 9, cache: 1];
object_tag = Inquire(
object,
"object tag")[instance: 3, cache: 1];
annoted_object =
Options(
object,
"send boxes",
0,
"cache",
1,
"object tag",
object_tag,
"ddcamera",
whichCamera,
"rendering approximation",
approximation_options,
"render every",
density_options,
"button state",
buttonState,
"rendering mode",
renderModeString) [instance: 1, cache: 1];
RECresNull =
Inquire(
RECresolution,
"is null + 1") [instance: 4, cache: 1];
ImageResolution =
Inquire(
camera,
"camera resolution") [instance: 5, cache: 1];
RECresolution = Switch(
RECresNull,
RECresolution,
ImageResolution) [instance: 10, cache: 1];
RECaspectNull =
Inquire(
RECaspect,
"is null + 1") [instance: 6, cache: 1];
ImageAspect =
Inquire(
camera,
"camera aspect") [instance: 7, cache: 1];
RECaspect = Switch(
RECaspectNull,
RECaspect,
ImageAspect) [instance: 11, cache: 1];
switchRECenable = Compute( "$0 == 0 ? 1 : (($2 == $3)  ($4 == $5)) ? ($1 == 1 ? 2 : 3) : 4",
RECenable,
switchRenderMode,
RECresolution,
ImageResolution,
RECaspect,
ImageAspect) [instance: 4, cache: 1];
NoRECobject, RECNoRerenderObject, RECNoRerHW, RECRerenderObject = Route(switchRECenable, annoted_object);
Display(
NoRECobject,
camera,
where,
throttle) [instance: 1, cache: 1];
image =
Render(
RECNoRerenderObject,
camera) [instance: 1, cache: 1];
Display(
image,
NULL,
where,
throttle) [instance: 2, cache: 1];
```

```
WriteImage(
image,
RECfile,
RECformat) [instance: 1, cache: 1];
rec_where = Display(
RECNoRerHW,
camera,
where,
throttle) [instance: 1, cache: 0];
rec_image = ReadImageWindow(
rec_where) [instance: 1, cache: 1];
WriteImage(
rec_image,
RECfile,
RECformat) [instance: 1, cache: 1];
RECupdateCamera =
UpdateCamera(
camera,
resolution=RECresolution,
aspect=RECaspect) [instance: 2, cache: 1];
Display(
RECRerenderObject,
camera,
where,
throttle) [instance: 1, cache: 1];
RECRerenderObject =
ScaleScreen(
RECRerenderObject,
NULL,
RECresolution,
camera) [instance: 1, cache: 1];
image =
Render(
RECRerenderObject,
RECupdateCamera) [instance: 2, cache: 1];
WriteImage(
image,
RECfile,
RECformat) [instance: 2, cache: 1];
main_Image_2_in_1 = "Image_2";
main_Image_2_in_3 = "X24,";
main_Image_2_in_4 = 1;
main_Image_2_in_5 = [30711.1 8251.01 27000];
main_Image_2_in_6 = [30711.1 8251.01 281292];
main_Image_2_in_7 = 15330.9;
main_Image_2_in_8 = 640;
main_Image_2_in_9 = 0.75;
main_Image_2_in_10 = [0 1 0];
main_Image_2_in_11 = NULL;
main_Image_2_in_12 = 0;
main_Image_2_in_13 = NULL;
main_Image_2_in_14 = 1;
main_Image_2_in_15 = NULL;
main_Image_2_in_16 = NULL;
main_Image_2_in_17 = NULL;
main_Image_2_in_18 = NULL;
main_Image_2_in_19 = 0;
main_Image_2_in_20 = NULL;
main_Image_2_in_21 = NULL;
main_Image_2_in_22 = NULL;
main_Image_2_in_23 = NULL;
main_Image_2_in_25 = NULL;
main_Image_2_in_26 = NULL;
main_Image_2_in_27 = NULL;
main_Image_2_in_28 = NULL;
main_Image_2_in_29 = NULL;
main_Image_2_in_30 = NULL;
main_Image_2_in_31 = NULL;
main_Image_2_in_32 = NULL;
main_Image_2_in_33 = NULL;
main_Image_2_in_34 = NULL;
main_Image_2_in_35 = NULL;
main_Image_2_in_36 = NULL;
main_Image_2_in_37 = NULL;
main_Image_2_in_38 = NULL;
main_Image_2_in_39 = NULL;
main_Image_2_in_40 = NULL;
main_Image_2_in_41 = NULL;
main_Image_2_in_42 = NULL;
main_Image_2_in_43 = NULL;
main_Image_2_in_44 = NULL;
main_Image_2_in_45 = NULL;
main_Image_2_in_46 = NULL;
main_Image_2_in_47 = NULL;
main_Image_2_in_48 = NULL;
Executive("product version 3 1 4");
$sync main();
```

Apêndice C

OpenDx no Windows

Esse apêndice serve para orientar àqueles que desejam utilizar o OpenDx no Windows.

A primeira coisa que é preciso fazer, é instalar um ambiente Linux para Windows. Recomenda-se a instalação do Cygwin. Esse programa auxilia os desenvolvedores na migração de aplicativos do UNIX/Linux para a plataforma Windows 98/Me/2000/XP. No site

http://cygwin.org/

é possível encontrar informações sobre o software. O download pode ser feito gratuitamente em vários sites.

Durante a instalação, o usuário é requisitado a informar quais pacotes deseja instalar de uma lista. Deve clicar sobre os pacotes até aparecer o status "install" e depois continue a instalação através das janelas. A instalação completa demora um pouco. Ao final do procedimento deverão aparecer ícones do *cygwin* e do *X windows* (um X preto) no desktop.

No site:

http://opendx.npaci.edu/bin/

existe disponível uma versão do openDX, própria para rodar no cygwin. Deve-se fazer o downoad desse versão (ou mais atualizada, mas que seja desenvolvida para o cygwin) no diretório C:.

Versão que deve ser gravada no diretório C: opendx-4.3.0-rtc-cygwin.tar.gz

O usuário deve então clicar sobre o ícone do Cygwin que aparece no desktop (o cygwin já foi instalado). Abrir-se-á uma janela que é o *bash* do emulador.

Digitar *exatamente*:

cd /

Teclar "*Enter*"

Digitar em seguida:

tar zxvf /cygdrive/c/opendx-4.3.0-rtc-cygwin.tar.gz

Teclar "*Enter*"

Dessa forma, os binários serão extraídos.
Concluída a etapa acima, deve-se digitar no prompt do bash do cygwin:

startxwin.sh

Esse comando iniciará a shell do X-windows idêntica ao terminal do linux. Nela ao se digitar "dx", inicializa-se o Opendx, já instalado.

Dentro da pasta C: do Windows, existirá uma pasta com o nome de Cygwin. Dentro dela, existirão as pastas nativas do linux, como se este estivesse todo dentro deste diretório.