DJALMA INÁCIO DA SILVA

VISUALIZAÇÃO CIENTÍFICA DE DADOS ANALÍTICOS FILTRADOS POR FORMULAÇÕES MATEMÁTICAS
ESTUDO DE CASO
"DISTRIBUIÇÃO DE TEMPERATURAS"

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Curso de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná - UFPR.

Orientador: Prof. Dr. Klaus de Geus

CURITIBA

2000

Ministério da Educação
Universidade Federal do Paraná
Mestrado em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática do aluno ***Djalma Inácio da Silva***, avaliamos o trabalho intitulado **Visualização Científica de Dados Analíticos Filtrados por Formulações Matemáticas - Estudo de Caso "Distribuição de Temperaturas"**, cuja defesa foi realizada no dia 11 de dezembro de 2000. Após a avaliação, decidimos pela aprovação do candidato.

Curitiba, 11 de dezembro de 2000.

Prof. Dr. Klaus de Geus
Presidente

Profª. Dra. Carla Maria Dal Sasso Freitas
Membro Externo - UFRGS

Profª. Dra. Olga Regina Pereira Bellon
DINF/UFPR

Prof. Dr. Hélio Pedrini
DINF/UFPR

DJALMA INÁCIO DA SILVA

VISUALIZAÇÃO CIENTÍFICA DE DADOS ANALÍTICOS FILTRADOS POR FORMULAÇÕES MATEMÁTICAS ESTUDO DE CASO "DISTRIBUIÇÃO DE TEMPERATURAS "

Dissertação aprovada como requisito parcial para obtenção do grau de Mestre no Curso de Pós-Graduação em Informática da Universidade Federal do Paraná, pela Comissão formada pelos professores:

Orientador: Prof. Dr. Klaus de Geus
Setor de Ciências Exatas, UFPR

Prof. Dra. Carla Maria dal Sasso Freitas
Instituto de Informática, UFRGS

Prof. Dra. Olga Regina Pereira Bellon
Setor de Ciências Exatas, UFPR

Prof. Dr. Hélio Pedrini
Setor de Ciências Exatas, UFPR

Curitiba, 11 de dezembro de 2000

Nada veremos se não tivermos, em nossos olhos, o meio de surpreender, de questionar e de dar forma a um número indefinido de configurações de cor e de espaço.

M. Ponty

À minha esposa e ao meu filho.

AGRADECIMENTOS

Ao Professor Dr. Klaus de Geus

pela orientação e paciência.

Aos Professores Dr. Germano Afonso, Dra. Olga R. Bellon e Dr. Adonai S. Sant'Anna

pela assistência

Dietmar W. Foryta

pelas várias sugestões de exemplos aplicados à astrofisica estelar e planetária

Aos professores: Msc. Regina M. H. P. Rodriguez e Dr. Dmitri I. Vlassov,

do Laboratório de Máquinas Térmicas da Engenharia Mecânica da UFPR,

pelo modelo experimental sugerido.

À Universidade Federal do Paraná, pelos recursos técnicos.

Aos colegas de pós-graduação, pelos estímulos.

Aos meus alunos de graduação, pela paciência.

À minha família, por todo apoio e compreensão.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

RESUMO

O desenvolvimento de programas visualizadores analíticos que permitam uma análise completa de fenômenos é uma tarefa complexa. Um aspecto que tem fundamental importância neste processo é a capacidade de produzir visualizações a partir das formulações físico-matemáticas que descrevem tais fenômenos e também a partir de dados não analíticos. Outro aspecto importante é referente aos recursos de investigação científica que o programa fornece. Dentro deste contexto está a possibilidade de selecionar áreas de interesse nos dados, ou sub-regiões de dados. No entanto, os sistemas de visualização existentes não abordam de maneira adequada esta questão, restringindo-se, em sua maioria, a contornos puramente geométricos.

Neste trabalho, técnicas baseadas em filtros físico-matemáticos são investigadas, os quais podem ser formulações para funções, matrizes, sistemas de equações, entre outras. O *software* de visualização desenvolvido para demonstrar as técnicas investigadas também contempla uma linguagem interpretada, desenvolvida para permitir a modelagem para as visualizações. O visualizador não exige o conhecimento de programação por parte do usuário, porque muitas visualizações são geradas através de scripts matemáticos que são, automática e posteriormente, convertidos pelo visualizador em código-fonte da linguagem interpretada. As características aqui descritas são validadas através de uma experiência realizada na área de transferência de calor.

ABSTRACT

The development of analytic visualization software systems which allow full analysis of phenomena is a complex task. A feature which plays a major role in this process is the ability to produce visualization images from physic-mathematical formulations which describe such phenomena and also from non-analytic data. Another important issue refers to resources for scientific investigation provided by the system. In this context, the possibility to select areas of interest within the data is especially important. However, existing visualization systems fail to address this issue adequately, most of them restricting the selection of sub-regions to purely geometric contours.

In this work, techniques based upon physic-mathematical filters are investigated, which can be formulations for functions, matrices, equation systems, among others. The visualization system developed to demonstrate the techniques which were investigated also introduces an interpreted language, which was developed to allow for visualization modelling. The visualization system does not require programming skills, since many visualizations are generated through mathematical scripts which are automatically converted to the interpreted language source code. A heat transfer experiment was performed in order to validate the features described here.

1 INTRODUÇÃO

Efetivamente a Visualização em Computação Científica (ViSC) surgiu no evento da SIGGRAPH de 1987 (ROSENBLUM, 1994, p. 4) pela apresentação de um painel criado por McCormick, DeFanti e Brawn. Neste painel estavam propostas as seguintes definições:

> *Applying graphics and imaging techniques to computational science is a whole new area of endeavour, which Panel members termed Visualization in Scientific Computing.*
>
> *The ability of scientists to visualize complex computations and situations is absolutely essential to ensure the integrity of analyses, to provoke insights and to communicate those insights to others.*

A citação acima poderia ser entendida como "A aplicação de técnicas gráficas e de imagens na ciência computacional é uma área inteiramente nova designada como Visualização em Computação Científica. A habilidade dos cientistas em visualizar computações complexas e outras situações é absolutamente essencial para assegurar a validade das análises, inferindo sobre novas proposições e comunicando tais descobertas".

Atualmente, a Visualização em Computação Científica é designada apenas como Visualização Científica (ROSENBLUM, 1994, Introdução). Entretanto, a utilização de gráficos e outras representações visuais vêm de longa data. A necessidade do registro e interpretação visual é imperativa ao *homo sapiens* desde os tempos das cavernas com suas pinturas rupestres. Também conta a lenda que o célebre Arquimedes (287-212 a.C.), considerado o maior matemático da antigüidade, foi morto enquanto desenhava figuras geométricas no solo (ROSENBLUM, 1994, Introdução). Em 1637, período renascentista, o famoso filósofo, matemático e físico francês René Descartes (1596 - 1650 d.C.) afirma: "A imaginação ou a visualização, e em particular a utilização de diagramas, tem um papel crucial no desenrolar da investigação científica" (EARNSHAW, 1993, p. 4). Além de

Descartes, cientistas do século XVII como Halley, Watt, Lambert e tantos outros, também empregavam algum tipo de técnica para a representação visual de dados (EARNSHAW, 1993, p. 3). Obviamente com o surgimento dos computadores, a partir de 1960, a representação de dados ficou muito mais facilitada e levou ao surgimento da área de Visualização em Computação Científica, sendo provavelmente o ramo da Computação Gráfica que mais recebe o ingresso de pesquisadores das mais diferentes áreas do conhecimento.

A finalidade principal da Visualização Científica é fornecer uma representação visível de fenômenos e/ou entidades e dar subsídios qualitativos e quantitativos para a sua investigação científica. Estas características denotam a fundamental importância que a Visualização Científica tem nas áreas de sua aplicação. Na medicina, por exemplo, a ViSC tem um importantíssimo papel em modalidades de imageamento tais como topografia e ressonância magnética. Como, também, em muitos ensaios laboratoriais e simulações. Praticamente todas as Engenharias, setores ligados à Biologia e à Medicina, e também à Física e à Química, utilizam a Visualização Científica para a representação visual e análise científica de dados. Por se tratar de assunto de interesse direto de pesquisadores das mais diversas áreas da ciência, a Visualização Científica recebe a adesão de novos cientistas.

Pertinente à introdução desta dissertação, é necessária a enunciação da proposição fundamental a demonstrar neste trabalho, e esta pode ser expressa como: A construção de um protótipo de um software, o qual permita ao pesquisador, principalmente das áreas ligadas à Física e à Matemática, a visualização de dados obtidos a partir da modelagem físico-matemática do fenômeno principal definido pelo usuário, sendo que tais dados podem sofrer uma filtragem de caráter seletivo através de formulações igualmente matemáticas, alternativamente designadas neste trabalho como **Filtros Físico-Matemáticos**. Esta capacidade de filtragem deve ser uma opção inerente ao ambiente do programa, evitando uma codificação complexa pelo usuário e os filtros, estabelecidos pelo usuário, são definidos como

entidades matemáticas que se apresentam sob a forma de equações, representações vetoriais ou matriciais, sistemas lineares ou não-lineares, entre outras, as quais podem compor uma condição limitante puramente geométrica ou podem, em si, ser modelagens matemáticas de fenômenos físicos correlatos ao fenômeno principal investigado e que, em qualquer uma dentre as duas circunstâncias, limitam a região dos dados a ser visualizada.

Deve-se observar que é de responsabilidade do usuário pesquisador, o fornecimento de filtros físico-matemáticos ao programa visualizador na entrada dos parâmetros relacionados aos fenômenos e/ou entidades principais a serem visualizados. O estudo de caso utilizado neste trabalho, para efeito da demonstração de aplicabilidade do software desenvolvido, segundo a proposição fundamental supra enunciada, é o fenômeno da distribuição de temperaturas[1], em regime permanente, ao longo de uma barra de aço.

1.1 VISUALIZADORES ANALÍTICOS

Notoriamente, como será revelado pela análise, os programas disponíveis permitem poucas opções ou, em alguns casos, nenhuma opção quanto à escolha das camadas de dados a serem visualizadas. Freqüentemente são permitidas escolhas de limitantes puramente geométricos como planos de seccionamento ou valores de extremidade para um domínio de valores, chamados neste trabalho de contornos ou sentinelas do intervalo de dados visualizados. Então, os dados oriundos de arquivos em memória secundária, como discos rígidos, são modelados e apresentados, dentro de tais limites, segundo a seleção de algum tipo de representação, a saber: isosuperfícies, campos vetoriais, glifos, entre outras. Existem situações que não envolvem a dificuldade de seleção de áreas do fenômeno para a visualização, mas a praticidade de resposta do software quanto à interface ser mais amigável ao usuário.

---

[1] Este tipo de problema da área de Transferência de Calor é estudado pelas Engenharias em Laboratórios de Máquinas Térmicas.

Apesar da dificuldade de acesso a muitos dos programas visualizadores existentes, foi realizada uma análise dos programas visualizadores mais difundidos da atualidade a partir de informações, na maioria das vezes, disponíveis nos sites das empresas proprietárias, ou a partir de prospectos informativos. As informações tornaram-se mais completas com a busca de artigos relacionados à proposição central desta dissertação. Foram selecionadas publicações dos últimos 10 anos em revistas especializadas e reconhecidas e artigos publicados a partir de encontros como congressos sobre Computação Gráfica como o **Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens** (SIBGRAPI) e o maior evento na área de Computação Gráfica o ***Special Interest Group on Computer Graphics*** (SIGGRAPH).

A análise dos programas incluiu a categoria de programas solucionadores analíticos de problemas matemáticos, os quais não são, por completo e *a priori*, visualizadores. Entretanto, muitos destes programas incorporam há algum tempo recursos de visualização como o *Maple 6.0*[2] e *Matlab 5.3.1*[3]. Em tais programas, o uso de formulações matemáticas constitui a própria finalidade básica dos programas, e a obtenção de seções geradas através de equacionamentos matemáticos é plausível. Contudo, ainda que estes programas empreguem formulações matemáticas inerentes à sua aplicação, tais formulações não estão intrinsecamente relacionadas às capacidades filtrantes enunciadas neste trabalho como uma condição natural do ambiente destes programas.

A maioria dos programas visualizadores possuem uma característica em comum, a qual consiste na utilização de dados provenientes de algum formato nativo de arquivo em memória secundária como, por exemplo, arquivos de dados em discos rígidos. Os dados lidos

---

[2] Informações sobre o *Maple 6* podem ser encontradas no site: http://www.maplesoft.com. O *Maple* é produzido pela empresa canadense: *Waterloo Maple Inc.* - Ontário, Canadá.

[3] O *Matlab* foi criado pela empresa americana *MathWorks* - Natick, Massachusetts, EUA e informações a respeito do software podem ser encontradas no site: http://www.mathworks.com Em 1999 foi lançada a versão 5.3 do software *Matlab* e nesta versão o programa permitia a criação de visualizações por isosuperfícies e campos vetoriais. A idéia foi tornar o *Matlab* um software com algumas capacidades de um visualizador, além das suas capacidades de solução analítica.

são modelados e visualizados segundo algum tipo de representação escolhido pelo usuário. Porém, essa característica de entrada a partir de dados contidos em arquivos, normalmente, dificulta o intercâmbio de tais dados com aplicativos iminentemente analíticos (RUMPF et al., 1996).

Alguns programas propõem um núcleo de rotinas gráficas de visualização, cuja utilização é interativa, mas as rotinas devem ser chamadas por programas criados pelo usuário. Neste caso, o usuário deverá saber utilizar linguagens de programação[4], as quais, na área científica, são freqüentemente a linguagem C e a linguagem FORTRAN. Um dos representantes deste tipo de biblioteca gráfica interativa destinada à visualização é o *VISUAL 3*[5], o qual é um ambiente gráfico interativo, baseado na utilização do padrão OpenGL[6] (WOO, 1999, p. 2). Este ambiente é chamado a partir de programas escritos em linguagem FORTRAN ou linguagem C.

O *Visualization ToolKit* (VTK[7]) é também um sistema aberto e distribuído gratuitamente para o desenvolvimento de interfaces gráficas para aplicações em Computação Gráfica 3D, Processamento de Imagens e áreas afins. O VTK pode ser utilizado em conjunto com a linguagem C/C++, implementação de classes para objetos, e permite a utilização de linguagens interpretadas como *Tool Command Language* (Tcl/Tk, a qual é baseada em scripts constituídos por comandos, os quais são strings que podem ou não receber argumentos), *Java*

---

[4] A linguagem de programação C foi criada em 1970 por Dennis Ritchie e é uma das linguagens mais utilizadas para desenvolvimento de programas que exigem alto desempenho, como os sistemas operacionais. O termo FORTRAN vem da contração das palavras Formula Translation e é uma linguagem de programação muito empregada na área científica. Sua primeira implementação foi criada entre os anos de 1954 e 1957, por uma equipe da IBM liderada por John W. Backus

[5] O *VISUAL 3* é um produto desenvolvido pelo Massachussets Institute of Technology (MIT) e mais informações sobre o *VISUAL 3* podem ser obtidas no endereço: http://raphael.mit.edu/visual3/visual3.html.

[6] OpenGL é uma biblioteca de funções gráficas da *Silicon Graphics Inc* de âmbito geral, sendo que esta biblioteca está portada em um grande número de plataformas que utilizem alguma saída gráfica padronizada. Outras informações podem ser obtidas a partir do seguinte endereço: http://www.sgi.com

[7] Maiores informações sobre o *VTK* podem ser encontradas nos sites: http://www.kitware.com/vtk.html e http://www.tcltk.com/itcl/

e *Python*. Através destas linguagens é possível a construção extremamente rápida de aplicativos especialmente na área de Visualização. O VTK possui implementações para plataformas baseadas em UNIX e Windows 9x/NT e é considerado o mais alto nível de abstração de bibliotecas de funções gráficas como OpenGL ou PEX. Em si o VTK é um sistema que suporta a inclusão de um grande número de algoritmos para visualização tais como: *scalar, vector, tensor, texture*, entre outros, e técnicas avançadas de modelagem como: *implicit modelling, polygon reduction, mesh smoothing, cutting, contouring* e *Delaunay triangulation*. A idéia central do VTK é ser uma ferramenta de relativa facilidade para ser utilizada por programadores e pessoas que possuam um suficiente conhecimento computacional. Algumas características da combinação entre o VTK e o TCL incluem:

- Escrita robusta de programas orientados a objetos.
- Proodução de *interfaces* complexas com uma linha de comandos (através de *Widgets*).
- Display de gráficos 3D com texturas elaboradas (*TSIPP*).
- Interação com bases de dados comuns do mercado como o *Oracle* e *Sybase* (*Sybtcl* e *Oratcl*).
- Chamadas remotas em redes através de procedimentos (*expect*).
- Uso de estruturas de dados complexas e chamadas em sistema *UNIX* (*TclX*).
- Chamadas encapsuladas de programas em C (*Embed Tk*, abreviadamente *ET*).

Grande parte dos programas visualizadores são propostos segundo um ambiente de visualização modular (*MVEs*) como citado em FONSECA et al. (1997). Este tipo de organização é constituída de um programa principal, o qual chama, ou vincula, módulos específicos à finalidade de aplicação determinada pelo usuário. Alguns programas têm uma arquitetura mais fechada permitindo apenas módulos propostos pelo próprio desenvolvedor do programa visualizador, outros permitem a inclusão de módulos externos gerados de forma independente. A maioria dos programas de visualização são modulares e os programas visualizadores modulares mais conhecidos e abordados neste trabalho são o *Application*

Visualization System (conhecido como *AVS*[8]), o *IRIS Explorer*[9], o *Open Visualization Data Explorer*[10] e o *IDL*[11]. Além destes mais difundidos, foram estudados outros menos conhecidos.

A análise dos principais programas de visualização, e alguns programas da categoria de solucionadores numéricos, objetivou a determinação e relação das principais características e recursos destes programas. Observando as características e recursos de tais programas, constatou-se que não foi estudada neles, de maneira satisfatória, a abordagem adotada neste trabalho. A conclusão deste estudo foi o subsídio suficiente à enunciação da proposição principal deste trabalho, restando a sua demonstração.

O primeiro programa visualizador analisado foi o *AVS*, o qual possui, entre outras características, os seguintes principais elementos em sua área de trabalho chamada de *Network Editor Window*:

- *Data Input*: Módulo que lê arquivos de dados ou gera automaticamente os dados.
- *Filters*: Módulo que realiza operações de conversão sobre os dados como o tipo dos dados ou operações de processamento de imagens.
- *Mappers*: Módulo que converte dados sobre objetos gráficos.
- *Data Output*: Módulo que escreve os dados, ou visualiza tais dados.

Muitas ferramentas de visualização[12] e métodos de seccionamento estão disponíveis no programa *AVS*. Este software também proporciona suporte ao uso de grades de

---

[8] O *AVS* é um produto da *Advanced Visual Systems Inc.* de Waltham Massachusetts e para maiores informações consultar os sites: http://bach.ncsa.uiuc.edu/Viz/AVS/AVSintro.html e www.cica.indiana.edu/cica/faq/avs/avs.index.html

[9] Informações sobre *IRIS Explorer* estão no endereço: http://www.nag.co.uk/visual/IE/iecbb/DOC/Nt/tutorial/chap01.htm. O *IRIS Explorer* foi criado pela *Silicon Graphics Inc.*, mas o grupo *Numerical Algorithms Group Ltd* (conhecido como *NAG*), de Oxford UK, passou a desenvolvê-lo e representá-lo comercialmente.

[10] O *Open Visualization Data Explorer* é um visualizador produzido pela *IBM* e outras informações podem ser encontradas no endereço http://www.research.ibm.com/dx.

[11] O *IDL* é um software da empresa *Floating Point System UK Ltd*, de Berkshire UK, e pode ser localizado na home page: http://www.floating.co.uk

[12] Ferramentas ou técnicas de visualização compreendem as diferentes formas como os dados serão representados visualmente em um sistema. O *AVS* possui várias ferramentas como *isosurface*, *volumetric rendering*, *stream line* e *particle traces*, entre outras. Estas e outras ferramentas são definidas no capítulo 3 desta dissertação.

elementos finitos[13] (BESANT, 1983, p.144) e ao processamento de imagens[14] (GOMES, 1994, p. 2). Além destas características gerais, permite a interface com módulos gerados a partir da linguagem FORTRAN e da linguagem C. Pode-se, ainda, programar com o *AVS* através de uma linguagem interpretada chamada *Graph Viewer Command Language Interpreter* (abreviadamente *CLI*), incluindo a utilização de *Scripts*[15] de comandos, porém não exatamente em um nível de abstração, o qual evite a exigência do conhecimento de programação por parte do usuário. Obviamente, pode-se construir regiões de dados a partir de formulações matemáticas em muitos visualizadores, incluindo o *AVS*, mas evitar a escrita de código em linguagens de nível mais baixo não foi uma característica encontrada no *AVS*.

Um dos mais poderosos programas de visualização, que foi analisado, é *o IRIS Explorer*. O *IRIS Explorer* é um rico ambiente de programação visual para visualização de dados e é composto pelos módulos: *Open InventorTM*, *ImageVisionTM* e *OpenGLTM libraries*. Este programa tira partido dos inúmeros recursos da biblioteca gráfica OpenGL para produzir as visualizações dos dados. Disponibilizando diversas ferramentas de visualização e trabalhando em conjunto com uma extensa biblioteca de classes de soluções numéricas do *Numerical Algorithms Group (NAG)*[16], o ambiente do *IRIS Explorer* foi planejado para facilitar a personalização da criação de aplicações pela seleção de módulos de bibliotecas de rotinas existentes ou desenvolvidas pelo usuário. A inclusão destas bibliotecas, quando realizada, evita a digitação de linhas de código pelo usuário para a criação de uma determinada aplicação. Tal aplicação pode ser alterada e atualizada de forma simples. Isto permite ao usuário uma grande interatividade para análise de coleções de dados e sua

---

[13] O método dos elementos finitos, muito empregado nas engenharias, é uma estrutura elástica ou contínua, sendo representada por muitos componentes distintos ou elementos interconectados em um número finito de pontos nodais situados nos limites do elemento, onde os deslocamentos dos pontos são os parâmetros básicos desconhecidos do problema.

[14] No processamento de imagens, o sistema admite como entrada uma imagem que, após processada, produz outra imagem como saída.

[15] *Scripts* são recursos empregados em programas como o *3D Studio*, da empresa americana *Autodesk*, para a enumeração de ações que devem ser realizadas para a obtenção de uma tarefa, como por exemplo as ações que geram animações.

[16] O *Numerical Algorithms Group Ltd (NAG)*, de Oxford UK, é um dos maiores desenvolvedores de soluções numéricas do mundo. Vide endereço http://www.nag.co.uk para outras informações.

conseqüente visualização. Tais características apontam o *IRIS Explorer* como um software capaz de realizar as operações de filtragem, mencionadas neste trabalho, mas não há uma opção direta e específica neste sentido no ambiente principal do programa. As classes pertencentes às ferramentas de visualização, propostas no software desenvolvido neste trabalho, visam à inclusão deste conceito de forma automática e direta ao usuário e não como módulos independentes e intercambiáveis.

O *Open Visualization Data Explorer* da empresa americana *IBM* é a evolução do conhecido *IBM Data Explorer*. Esse aprimorado sistema de visualização tem muitas das características encontradas no *AVS* e *IRIS Explorer*, incluindo suporte à biblioteca OpenGL e uso de linguagem de programação em *scripts*. As características deste programa são:

- Flexível importação de dados e exportação de funções
- *Kit* de desenvolvimento de aplicações
- Fácil de usar os recursos da *interface*
- *Hardware rendering*[17]
- Entrada/Saída de imagens comprimidas
- Carga dinâmica de módulos
- *One-shots* para simulação

Apesar dos excelentes recursos apresentados no *Open Visualization Data Explorer* não foram encontradas referências a recursos de filtragem por formulações matemáticas.

Outro software analisado foi o *IDL* da empresa britânica *Floating Point System*. Neste visualizador são encontrados recursos semelhantes aos existentes no *AVS e IRIS Explorer*. Menos completo no escopo de tipos de ferramentas de visualização, este software permite a programação em módulos externos a partir de linguagens de programação como

---

[17] *Rendering* é o processo pelo qual obtém-se uma ou mais imagens finais a partir de modelos de dados.

FORTRAN e C. O *IDL* permite dados estruturados na forma de escalares, vetoriais, matriciais e estruturas agregadas. Vários tratamentos numéricos estão disponíveis a partir de bibliotecas de funções matemáticas. Alguns exemplos destes tratamentos são a integração numérica, operadores vetoriais e matriciais, aritmética complexa, interpolações entre outros. Efetivamente não foram encontrados recursos próprios do software que demonstrassem as características dos filtros físico-matemáticos, ainda que o programa permita a definição de funções e procedimentos criados pelo usuário.

Um dos trabalhos que mais se aproxima do proposto nesta dissertação é baseado em planilhas eletrônicas (LEVOY, 1984). O autor propõe um sistema de visualização de dados espelhado nas clássicas planilhas eletrônicas, onde as células desta planilha contêm objetos gráficos como imagens, volumes e filmes ou, ainda, botões, editores de curva e fórmulas. Os objetos contidos nas células são exibidos em miniatura. As fórmulas, que são inseridas nas células, são escritas em uma linguagem de programação de propósito geral, chamada *TCL*. Tais fórmulas podem ser alimentadas com operadores para manipulação de matrizes, processamento de imagens e *rendering*. Apesar da facilidade de programação das células, descrita por LEVOY, em relação aos sistemas estilo *flow chart*[18], o sistema de visualização baseada em planilhas, da forma que foi proposto, não contempla opções para os filtros formulados matematicamente.

Um software Visualizador para Dados Paralelos (chamado de *DPV*) é proposto por WAGNER e BERGERON (1995). Os autores afirmam que os programas de visualização paralela de dados e depuradores a eles destinados, requerem novas técnicas para reunir, mostrar e controlar os vários *displays*[19] gerados. Um grau satisfatório de interação entre o usuário e o programa de tratamento paralelo é também requerido para facilitar o processo de

---

[18] *Flow chart* é uma representação para códigos, normalmente algorítmica, que utiliza diagramas de blocos ligados por linhas, as quais indicam a seqüência de ações. Os diagramas contêm as ações a serem realizadas.

[19] *Display* é o ato de representar uma imagem em um dispositivo gráfico.

depuração paralela. É importante salientar que um dos exemplos de aplicação utilizado no *DPV* e citado no referido artigo, para efeito de demonstração das qualidades do programa, está inexoravelmente relacionado ao estudo de caso aplicado ao software descrito neste trabalho. Tal aplicação está intitulada como: "Visualização de um algoritmo de Jacobi para encontrar a distribuição de temperaturas, sob regime térmico permanente e propagação bidimensional, em superfícies, dadas as condições constantes de contorno". Todavia, apesar de sua aplicação na mesma área de fenômenos da Transferência de Calor, não foram encontradas alusões às caracteristicas de tratamento matemático no referido trabalho.

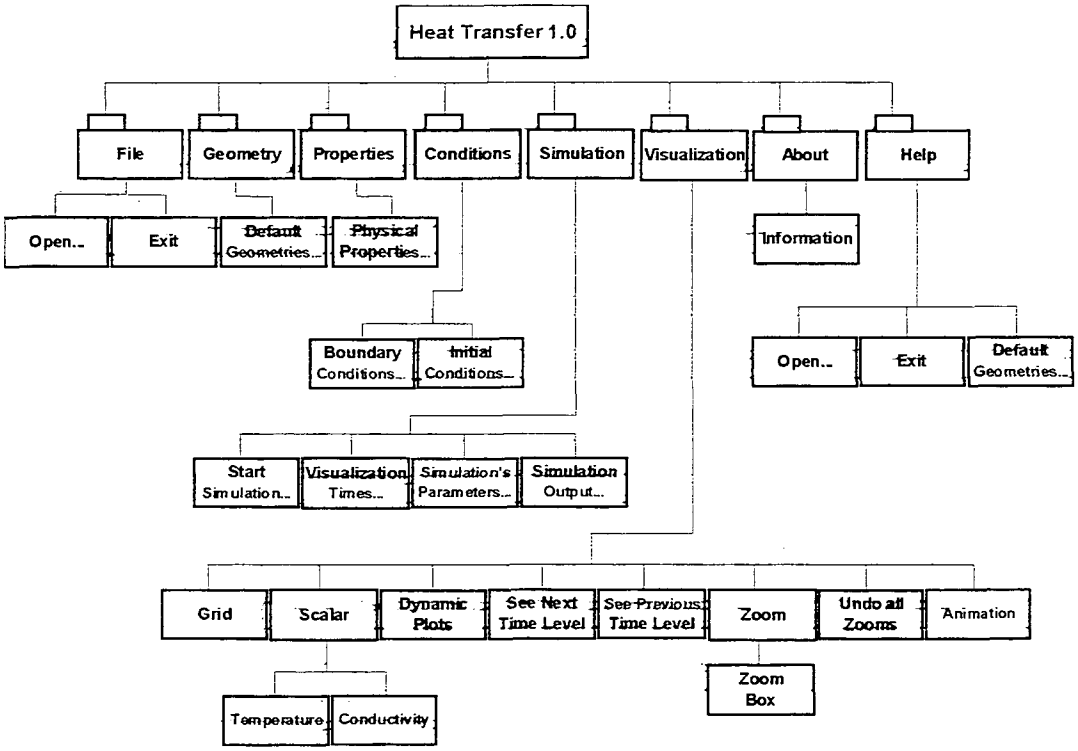
Alguns outros visualizadores investigados são de aplicação específica à área da Transferência de Calor (MALISKA et al., 1998) e da Dinâmica de Fluidos (MALISKA e DIHLMANN, 1992) . Entretanto, por se tratarem de assuntos imediatos ao tema principal abordado neste trabalho, merecem uma atenção especial, principalmente os visualizadores destinados à área de Transferência de Calor, a qual é apresentada de forma resumida no capítulo 2 desta dissertação.

Um software visualizador específico analisado, que está estritamente relacionado ao desenvolvimento descrito nesta dissertação, foi o *HEAT TRANSFER 1.0* (MALISKA et al., 1997). Este programa, que é de domínio público[20], foi desenvolvido pelo grupo de Simulação Numérica em Mecânica de Fluidos e Transferência de Calor (cuja abreviatura é *SINMEC*) da Universidade Federal de Santa Catarina (UFSC), com o propósito de ser um software de auxílio no ensino da condução de calor. Destinado aos alunos de graduação, mais especificamente os da Engenharia Mecânica, este programa possui recursos de visualização restritos à sua área de atuação. Obviamente, não cabe aqui uma explicação pormenorizada de todas as opções que o programa possui, mas a Figura 1 ilustra as opções de menu que tal programa fornece em seu ambiente de trabalho:

---

[20] Para se obter uma cópia do *Heat Transfer 1.0* (ou da versão em Português chamada *TransCal*) é indicado o seguinte endereço: http://www.sinmec.ufsc.br.

FIGURA 1: ESTRUTURA DE MENUS DO PROGRAMA HEAT TRANSFER 1.0



FONTE: NATIONAL HEAT TRANSFER CONFERENCE, VOLUME 6 ASME 1997

O programa permite a visualização das distribuições de temperaturas em regime térmico permanente de forma similar à apresentada no presente trabalho. Entretanto, não possui opções dentro do próprio ambiente para a filtragem formulada matematicamente e não permite a utilização de módulos externos que realizem a tarefa da filtragem por tais formulações nas regiões de interesse dos dados visualizados. Apesar do *Heat Transfer* não ser um *software* visualizador de aplicação geral, tem implicações importantes relacionadas ao estudo de caso abordado no trabalho descrito nesta dissertação.

A conclusão a que se chega, após a análise de visualizadores existentes, é a ausência, até onde se sabe, da característica da proposição deste trabalho. Caso existisse um software que reunisse as virtudes de visualização de um programa, como o *IRIS Explorer*, com os recursos matemáticos de um programa, como o *Maple* 6, este software poderia implementar facilmente a filtragem de dados a partir de formulações matemáticas como uma opção inerente ao ambiente de tal programa.

1.2 MOTIVAÇÃO

Aparentemente os visualizadores analisados não estão bem adequados às finalidades descritas neste trabalho de limitantes formulados matematicamente e que sejam incorporados como uma opção base do sistema de visualização, surgindo a necessidade do desenvolvimento de um visualizador para as aplicações que exigem a extração de regiões de dados a serem visualizadas a partir de formulações matemáticas.

A idéia da utilização de ferramentas visuais de programação orientada a objetos, em conjunto com bibliotecas de funções gráficas existentes no mercado, como o OpenGL, e algumas rotinas clássicas para criação de linguagens interpretadas, como proposta por SCHILDT (1988, p. 247), e estruturas de dados, como as presentes em VILLAS (1993, p. 110), que são de domínio público, motivou a construção de um protótipo de software visualizador, o qual espelha as proposições enunciadas nesta dissertação.

1.3 OBJETIVOS

O desafio maior deste trabalho foi o desenvolvimento de um programa visualizador, o qual foi chamado de ***Mathematical Filters in Scientific Visualization***, versão1.0 (abreviado, doravante, por *MFSV*). Houve a necessidade da criação de uma linguagem interpretada de programação, a qual foi baseada na clássica representação brasileira para algoritmos conhecida como *Portugol*[21] (FORBELLONE, 2000). Por ter sido inspirada em tal representação algorítmica, a linguagem interpretada desenvolvida e utilizada pelo programa o MFSV foi chamada de **Linguagem Interpretada em Portugol** (doravante abreviada como *LIP*). A linguagem LIP tem as suas palavras reservadas em Português.

---

[21] *Portugol* é uma palavra resultante da contração da palavra Português e o sufixo 'ol' de algumas linguagens de programação.

O objetivo principal deste trabalho, e razão para o desenvolvimento do MFSV, é a demonstração da proposição desta dissertação e este desenvolvimento passou por várias etapas e estudos diferenciados. É importante ressaltar que a finalidade, neste trabalho, do software apresentado não é ser um sistema completo para visualização, mas que possua um número suficiente de opções permitindo a inferência sobre as premissas apresentadas neste trabalho.

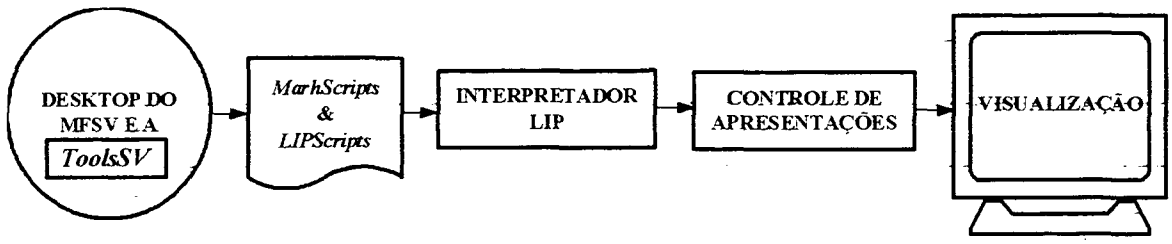
Outro objetivo deste trabalho é permitir que o MFSV receba a entrada de equações algébricas, variáveis e montagem de sistemas de equações lineares ou não-lineares através de áreas de edição de textos chamadas de *Scripts*. O software proposto foi inspirado nesses scripts para a entrada do usuário em uma sintaxe que respeitasse o formalismo matemático, ou permitisse a programação em LIP ao estilo clássico utilizado em linguagens como C e FORTRAN. Portanto, outro objetivo do trabalho é dar tais características à linguagem LIP. É importante frisar que existem duas áreas distintas de edição de textos no MFSV e conseqüentemente dois tipos de *scripts*:

- *Mathematical Scripts*: Chamados no programa MFSV de *MathScripts*, onde o usuário entra com um texto descritivo para a geração da visualização e a notação procura utilizar uma simbologia mais matemática e natural ao usuário pesquisador, existindo poucas exigências sintáticas diferentes do conhecimento do usuário.
- *LIP Scripts*: Chamados no programa MFSV de *LIPScripts*, onde o usuário poderá complementar, se desejar, a programação para a geração da visualização ao estilo clássico das linguagens tradicionais de programação.

O usuário pode utilizar tais *scripts* matemáticos, ou em LIP, como geradores dos dados a serem visualizados, ou trazer os dados previamente gerados, contidos em uma memória secundária. O programa MFSV sempre exigirá uma ferramenta de visualização

associada aos dados a serem visualizados. O ambiente do programa possui uma barra de ferramentas chamada `ToolsSV`[22], a qual possui muitos diferentes tipos de ferramentas de visualização. Cada uma destas ferramentas é implementada como um objeto de uma classe de visualização em uma biblioteca de classes. Tal classe possui um método transcritor e este método traduz, tal qual uma mini-ferramenta CASE (*Computer Aided Software Engineering*) de geração automática de código o texto da área de edição *MathScripts* para um código fonte em LIP, o qual será gerado na área de edição *LIPScripts*. Esta segunda área é controlada por outra classe específica, cujos métodos têm como finalidade a interpretação, ou seja a execução, do código em LIP. Na área do *LIPScripts* é possível a inclusão de outras linhas de código por parte do usuário. Neste caso o usuário deverá ter conhecimento das técnicas de programação de computadores. A execução do programa contido na *LIPScripts* avaliará, entre outros elementos, expressões matemáticas que conduzem à modelagem dos dados, os quais são, também, membros atributos da classe de visualização mencionada. Após encerrada a execução do programa em LIP a visualização já estará disponível em um *display* como sugerido pelo esquema da Figura 2:

FIGURA 2: ESQUEMA DA SEQÜÊNCIA DE OPERAÇÃO DO PROGRAMA MFSV



O Capítulo 6 apresenta um esquema completo do projeto e operação do MFSV.

[22] A barra *ToolsSV* (*Tools of Scientific Visualization*) possui, atualmente, apenas uma ferramenta, a qual foi necessária ao estudo de caso utilizado para a demonstração do MFSV.

Algumas outras rotinas auxiliares, escritas em LIP, para solução numérica de equações, de matrizes e de sistemas de equações lineares e não-lineares, entre outras, podem ser implementadas valendo-se dos recursos matemáticos do MFSV.

A interface do MFSV é primordialmente gráfica, não só no sentido das escolhas de objetos sobre uma janela de aplicativo, mas também na interatividade com o usuário. A tentativa é evitar ao máximo a codificação por parte do pesquisador. Este deverá apenas formular o fenômeno e suas condições iniciais e finais e o software o assistirá na obtenção da visualização. Muitas primitivas regulares como esferas, cones, entre outras, são apresentadas como opções de um item de menu do MFSV. Tais opções devem abrir janelas de diálogo parametrizadas para obtenção dos dados fornecidos pelo usuário, ou poder-se-ia obter dados, como as coordenadas de um ponto, com um mouse, a partir da área gráfica usada para a visualização. Uma ferramenta de visualização científica foi desenvolvida como exemplo de implementação. A visualização é gerada a partir de funções da biblioteca gráfica OpenGL[23] em um *framebuffer*[24] gráfico, o qual poderá ser apresentado em um *display* local ou em um terminal remoto, em processamentos que utilizem redes de computadores.

---

[23] O software proposto utiliza o Ambiente de Desenvolvimento Integrado do C++Builder (da Inprise Inc.) for Windows95/2000/NT combinado com um componente de acesso às funções gráficas da OpenGL. Este ambiente é uma poderosa ferramenta *RAD* (*Rapid Application Development*) para programação visual.

[24] *Framebuffer* é o nome dado ao conjunto de *bitplanes* que formam uma imagem. Os *bitplanes* são áreas da memória de acesso aleatório (chamada de memória RAM de um computador), destinadas ao armazenamento de informações de *bits* dos *pixels*, os quais são os elementos gráficos básicos de formação das imagens em um monitor de vídeo.

1.4 ORGANIZAÇÃO E DESENVOLVIMENTO DO PROJETO

O SINMEC, mencionado na seção 1.1, construiu um software (MALISKA e DIHLMANN, 1992) chamado ISO-3D, o qual realiza a visualização tridimensional para campos escalares e vetoriais, cuja aplicação é mais restrita à área de Dinâmica de Fluidos[25]. No referido artigo os autores afirmam:

> O objetivo principal que norteou o desenvolvimento do software foi a criação de um aplicativo robusto, de fácil uso pelos analistas numéricos e com a característica de ser "enxuto", isto é, sem carregar os excessos de determinadas rotinas gráficas estabelecidas, que ao executarem uma certa tarefa, muitas vezes trazem junto estruturas desnecessárias quando a aplicação é em mecânica dos fluidos computacional.

Com base nesta afirmação, o MFSV foi pensado e projetado tendo-se em vista a futura vinculação dinâmica[26] (*dynamic link*) e automática de classes visualizadoras. Desta forma a memória principal do computador não será sobrecarregada com ferramentas de visualização que não estejam em uso em um determinado momento. Entretanto, não são módulos externos e agregados manualmente pelo usuário, mas sim módulos indispensáveis à execução do MFSV.

Publicações, como a de KOCHHAR et alli (1991), relacionadas à tarefa de projetar arquiteturas para a construção de ambientes para programas científicos podem ser utilizadas para a criação de visualizadores. FONSECA et alli (1997) propõem a construção de ferramentas de manipulação para visualização interativa de dados volumétricos baseada em uma arquitetura orientada a objetos[27] utilizando *3D widgets*, os quais são definidos, em CONNER et al. (1992), como objetos que encapsulam geometria 3D e comportamento, utilizados para controlar ou exibir informações sobre objetos de aplicações 3D. As partes que

---

[25] Para maiores informações sobre aplicações computacionais na área da Dinâmica de Fluidos consulte KOTAKE e HIJAKATA (1993)

[26] Uma vinculação dinâmica ocorre durante a execução do programa e somente quando necessária.

[27] A programação orientada a objetos é um novo paradigma de modelagem e construção de programas, o qual substituiu em muitas áreas o antigo paradigma da programação estruturada.

compõem um *widget* reagem a eventos de interação e as restrições e relacionamentos à elas aplicados.

A programação do MFSV seguiu o paradigma de orientação a objetos e foi controlada por sistemas operacionais baseados em eventos[28], e o encapsulamento de comportamentos e a herança para classes foram priorizados, para garantir conceitos tais como a reutilização de código[29] (COHOON, 1999).

A organização proposta para a dissertação é conseqüência da própria seqüência de desenvolvimento do projeto e dos estudos realizados. A partir desta perspectiva foram subdivididos os capítulos em itens de acordo com os tópicos abordados para a construção do software MFSV bem como os aspectos de fundamentação teórica necessários.

O capítulo 1 faz uma introdução à área da Visualização Científica. Tal introdução inicia-se por um pequeno histórico do surgimento da Visualização Científica e sua importância na atualidade. Na seqüência foi realizada a revisão literária conjugada à análise necessária dos programas visualizadores existentes, na qual foi detectada a carência de certas características, as quais foram abordadas neste trabalho. No final da introdução do capítulo 1 é enunciada a proposição fundamental a demonstrar nesta dissertação.

O resultado deste trabalho de investigação científica foi a determinação da ausência, nos programas analisados, das formulações matemáticas como elementos filtrantes dos dados a serem visualizados, sendo esta uma condição inerente ao ambiente destes

---

[28] A programação baseada em eventos é uma característica fornecida por determinados sistemas operacionais, os quais enviam mensagens aos aplicativos dos acontecimentos, ou eventos, que estão ocorrendo em um determinado instante. Estes eventos podem ser acionamentos de teclas, o uso do *mouse*, entre outros.

[29] Encapsular significa esconder certas áreas do acesso de outras. A herança é a cópia de características de uma classe chamada base para classes chamadas derivadas, as quais podem aliar novas características. A reutilização é um conceito que está relacionado ao aprimoramento das versões de programas existentes. Os princípios teóricos da orientação a objetos facilitam o conceito da reutilização.

programas. Esta ausência foi o incentivo necessário e motivou a construção de um protótipo de software visualizador analítico, o MFSV. A construção deste demonstra adequadamente a proposição desta dissertação. Alguns trabalhos foram publicados, como em FONSECA (1997), com o sentido de apontar arquiteturas para a construção de visualizadores e auxiliaram no projeto do MFSV.

No capítulo 2 é realizado um resumo dos fundamentos da área de Transferência de Calor. Em tal área é escolhido um fenômeno que serviu para a demonstração da aplicação do MFSV.

O capítulo 3 mostra a taxionomia da Visualização Científica e o uso de uma metodologia seletiva, a qual permitiu a escolha de qual ferramenta de visualização adotar para a representação do fenômeno do estudo de caso. Nesse capítulo são apresentados os tópicos do OpenGL que foram estudados e por que foi ela a biblioteca de funções gráficas eleita para a programação do tratamento gráfico do MFSV.

A escolha de uma experiência envolvendo a fenomenologia da Transferência de Calor é apresentada no capítulo 4. Tal experiência envolve a condução de calor por corpos considerados aproximadamente lineares, ou de condução unidimensional, em regime térmico permanente. Descreve-se a opção por repetir a experiência em instalações próprias e as razões dessa decisão. Este capítulo apresenta, resumidamente, a experiência que forneceu os dados a serem visualizados e a série de considerações que foram levadas em conta para a construção do artefato que aqueceria a barra de aço, a qual fica suspensa horizontalmente por uma de suas extremidades. Outros fatores importantes como: Permitir o sistema atingir efetivamente o regime térmico permanente e a constância da temperatura ambiente foram cruciais no projeto e construção do artefato. Mais aspectos significativos foram considerados, como: precisão nos ajustes das resistências elétricas e a metodologia utilizada para realizar as medições de forma apropriada. Todos estes cuidados foram tomados para que os dados obtidos fossem os

mais condizentes possível com a realidade física do fenômeno. Esse capítulo apresenta, ainda, os resultados das medições e as formulações matemáticas do fenômeno, os quais foram necessários para a modelagem dos dados a serem visualizados pelo MFSV.

A essência da dissertação está abordada no capítulo 5 intitulado: Filtros Físico-Matemáticos em Visualização Científica. No referido capítulo é apresentada a consistência da proposição maior deste trabalho e quais foram os outros aspectos que corroboraram para a origem da idéia dos filtros formulados matematicamente. Tais filtros são conceituados e são relacionados alguns possíveis exemplos de aplicação de filtros físico-matemáticos. Por último, este capítulo mostra como são utilizados filtros formulados matematicamente no estudo de caso escolhido à demonstração do MFSV, e quais foram as respostas e visualizações obtidas através do programa MFSV.

Os aspectos do projeto do software e alguns detalhes de implementação foram apresentados no capítulo 6. Tal capítulo elucida as ferramentas de programação visual e gráficas utilizadas para o desenvolvimento do MFSV e são apresentadas algumas soluções algorítmicas criadas para o plexo de problemas adjacentes que surgiram, tais como: A necessidade de uma estrutura de dados para armazenamento de variáveis simples e arranjos matriciais, a construção dos *scripts* a partir das classes de objetos, os métodos transcritores e muitas outras situações que surgiram durante o desenvolvimento.

O último capítulo, o capítulo 7, aponta as conclusões finais do trabalho realizado, seu possível desenvolvimento posterior e outros trabalhos correlacionados que poderão ser investigados.

2 FUNDAMENTOS DE TRANSFERÊNCIA DE CALOR

Uma definição curta, embora ampla, fornece uma resposta inequívoca à pergunta: O que é a Transferência de Calor? "O trânsito de energia provocado por uma diferença de temperatura" (INCROPERA, 1990, p. 2). Toda vez que existir uma diferença de temperatura num meio, ou entre vários meios, ocorrerá a fenômeno da transferência obrigatoriamente.

A transferência de calor se dará, sensivelmente, em pelo menos um dos três seguintes modos: condução, convecção e radiação térmica.

A transferência por condução ocorrerá quando existir um gradiente de temperatura num meio estacionário, que pode ser um sólido ou um fluído. O estudo da termologia revela que os corpos com temperaturas altas possuem uma maior vibração de suas moléculas. A atividade molecular e atômica são os processos que sustêm a condução. O fornecimento ou transferência de energia por parte das moléculas mais enérgicas para as menos enérgicas é o fenômeno que retrata a condução. No estudo de caso, considerado neste trabalho, o modo de transferência mais presente é o da condução do calor considerado em um único sentido ao longo de uma barra de aço e a quantificação do processo de transferência de calor é dado pela equação da taxa baseada na Lei de Fourier:

$$q''_x = -k\frac{dT}{dx}, \qquad (1)$$

onde q''_x é o fluxo de calor, dado em W/m^2, que é a taxa de transferência de calor numa determinada direção chamada x por unidade de área ortogonal à direção da transferência e é proporcional ao gradiente de temperatura, dT/dx, nesta direção. É importante lembrar que a

condução não ocorre nas mesmas quantidades em diferentes materiais. Existem materiais mais isolantes e outros mais condutores e, via de regra, materiais que conduzem bem a eletricidade, também conduzem bem o calor. A constante de proporcionalidade k tem relação direta com esta característica do material e é conhecida como condutividade térmica. O sinal de menos à frente do k, na expressão *(1)*, indica que o calor se propaga na direção onde as temperaturas são menores.

A segunda forma de transferência de calor é a promovida pela convecção. O processo convectivo compreende dois mecanismos: A transferência de energia provocada pelo movimento molecular aleatório, conhecido por difusão, que é uma circunstância meramente microscópica e a transferência de energia provocada pela movimentação da massa do fluido, o que é uma circunstância macroscópica. Este último movimento está associado ao fato de, em qualquer instante, um grande número de moléculas estar se movendo coletivamente ou em agregados. Tal movimento, na presença de um gradiente de temperatura, provoca a transferência de calor. Mesmo assim as moléculas continuam a apresentar seus respectivos movimentos aleatórios e a superposição dos dois mecanismos denota a transferência total de energia. Quando é referido somente o movimento de massa é utilizado o termo advecção. A convecção de interesse imediato é a que ocorre entre um fluido em movimento e uma superfície limitante, quando existe uma diferença de temperatura entre eles.

A região desenvolvida desde a superfície, onde a velocidade é zero, até uma região onde a velocidade do fluido tende a um valor finito u_∞ é conhecida como camada hidrodinâmica, ou a camada de velocidade, ou ainda como a camada limite. Além deste fato, se existir uma diferença entre a temperatura da superfície e o fluido escoante, haverá uma faixa do fluido onde a temperatura variará desde T_s, em y=0, até T_∞, na região externa. A esta região é dado o nome de camada limite térmica, que poderá ser maior, menor ou mesmo igual à camada de velocidade. Em qualquer caso, sendo $T_s > T_\infty$, a transferência convectiva de calor se processará da superfície para o escoamento externo. Nas vizinhanças da superfície do

objeto existe o predomínio da transferência devida ao movimento aleatório das moléculas, porque a velocidade do fluido é baixa e junto à superfície ela é nula. A medida em que se afasta da superfície e até o limite da camada de velocidade, é percebida uma maior contribuição por parte do movimento macroscópico do fluido. Portanto, para entender os processos convectivos é preciso entender os fenômenos que ocorrem nesta camada limite, sendo este o papel da mecânica dos fluidos. Indiferente à forma específica que a convecção possa ter, a equação da taxa apropriada tem a seguinte configuração:

$$q'' = h(T_S - T_\infty), \qquad (2)$$

onde q'' é o fluxo de calor convectivo, dado em W/m^2. Tal equação é conhecida como a lei de Newton do resfriamento e a constante de proporcionalidade h, dada em $W/m^2.K$, é o coeficiente de transferência convectiva de calor, ou a condutância da película ou, ainda, o coeficiente de película. Esta situação ocorre no estudo de caso, escolhido neste trabalho, onde a barra de aço estará suspensa no ar, ou mais apropriadamente em um fluido, apoiada apenas por uma de suas extremidades. Deve-se notar que o processo convectivo poderia ser do tipo latente, se existisse a mudança de estado de líquido para vapor ou vice-versa do fluido. Porém, este tipo de mudança não ocorre no estudo de caso.

O terceiro e último modo de transferência de calor é a radiação térmica. A radiação térmica é a energia emitida pela matéria que estiver em uma temperatura finita, a qual será obviamente diferente de zero absoluto[30]. Esta emissão poderá ocorrer, também, a partir de líquidos e gases, embora quase sempre estejamos interessados na emissão de superfícies sólidas. Qualquer que seja o estado da matéria, entende-se que a emissão ocorrerá devido às modificações das configurações eletrônicas dos átomos ou das moléculas que a constituem. O transporte da energia, em processos térmicos emissivos, é feito pela propagação de ondas

[30] O valor do zero absoluto é de -273,15°C e é impossível chegar a este extremo de temperatura.

eletromagnéticas[31]. Logicamente a propagação de ondas eletromagnéticas prescinde de um meio material e, ao contrário do que se possa pensar, se realizará de forma melhor na ausência de um sustentáculo. Assim, a melhor eficiência na propagação de uma onda eletromagnética é a que ocorre no vácuo. O fluxo máximo de calor, dado em W/m^2, em processos de radiação térmica é expresso pela lei de Stefan-Boltzmann:

$$q'' = \sigma T_s^4, \qquad (3)$$

onde T_s é a temperatura absoluta, dada em K, da superfície e σ é a constante de Stefan-Boltzmann ($\sigma = 5{,}67x10^{-8} W/m^2.K^4$). Nestas condições tem-se o que é chamado de radiador ideal ou um corpo negro. Entretanto, em condições reais, a emissão de um corpo é menor que a ideal e a fórmula utilizada será a seguinte:

$$q'' = \varepsilon \sigma T_s^4, \qquad (4)$$

onde ε é uma propriedade radiativa da superfície, a emissividade. Esta propriedade, cujo valor está sempre no intervalo $0 \leq \varepsilon \leq 1$, indica a eficiência de emissão da superfície em relação a um radiador ideal. Lembrando que um corpo poderá receber energia térmica radiativa por unidade de área e absorvê-la. Neste caso, tem-se uma propriedade da matéria chamada absortividade α, a qual estará sempre no intervalo $0 \leq \alpha \leq 1$. No estudo realizado nesta dissertação não foram consideradas as situações de absorção oriundas de radiações térmicas. O processo de transferência de calor pela via radiativa é, em geral, bastante complexo. Entretanto, pode-se supor algumas simplificações na forma de seu tratamento. Uma das simplificações é considerar $\varepsilon = \alpha$, o que acarretará na propriedade de um determinado material, o qual emite e recebe energia com a mesma taxa líquida, sendo este material denominado como de superfície cinzenta. Supondo que uma das superfícies é bem

[31] Na Física Quântica é usada a seguinte designação equivalente: Emissão fotônica.

pequena em relação a outra e está completamente envolta pela maior e, ainda, que o espaço entre elas está ocupado por um gás, que não interfere no processo de transferência. Nestas condições, ter-se-á uma situação onde valerá a seguinte fórmula da taxa líquida de transferência de calor:

$$q'' = \frac{q}{A} = \varepsilon\sigma(T_s^4 - T_{viz}^4), \qquad (5)$$

onde A é área da superfície, dada em m^2, e ε é a sua emissividade, enquanto T_{viz} é a temperatura absoluta, dada em K, das vizinhanças. Neste caso particular, a emissividade das vizinhanças não interfere na taxa líquida da troca térmica. Algumas vezes constuma-se expressar a troca líquida pela seguinte fórmula:

$$q_{rad} = h_r A(T_s - T_{viz}), \qquad (6)$$

onde h_r é o coeficiente de transferência radiativa de calor e é dado pela seguinte fórmula:

$$h_r = \varepsilon\sigma(T_s + T_{viz})(T_s^2 + T_{viz}^2). \qquad (7)$$

Uma utilidade imediata dessa atitude é a linearização da equação da taxa líquida de transferência de calor. Entretanto, o coeficiente de transferência radiativa tem uma conformidade não-linear e esta está relacionada à temperatura. Esta dependência é muito forte quando comparada com o coeficiente de transferência convectivo.

Muitos aspectos e detalhes da fenomenologia da área de Transferência de Calor foram aqui omitidos ou serão abordados em itens de capítulos posteriores, mas a literatura a respeito destes fenômenos é ampla. Contudo, ainda é necessário explicar a condição geométrica de propagação do calor por condução de acordo com as características do material e se existe ou não variação da potência da fonte fornecedora de energia. Na transferência de

calor tem-se os conceitos de regime permanente, ou estacionário, e o regime transiente, ou variável. No regime permanente, normalmente mais simples, são considerados todos os eventos que ocorrem após o equilíbrio térmico ter sido atingido, o qual é o regime escolhido para o estudo de caso deste trabalho. O regime transiente é toda a seqüência de alterações que ocorre quando inicia-se o fornecimento de energia a um sistema, ou quando é deixado de fornecê-la. Todo sistema que não sofrer alterações continuadas no fornecimento de energia tenderá a atingir o equilíbrio térmico. A constância da potência da fonte conduz ao equilíbrio térmico. A forma como o calor é conduzido através de um material é sempre tridimensional. Porém, em situações específicas podemos considerá-la como bidimensional, ou mesmo unidimensional, como é a consideração feita no presente trabalho para o estudo de caso escolhido. Para que se possa considerar a condução de calor realizada de forma unidimensional, existe a necessidade das dimensões da seção transversal do objeto serem bem menores, quando comparadas ao comprimento da peça. Barras de aço atendem a este quesito e, portanto, pode-se supor que a condução acontecerá em um único sentido.

Após esta breve introdução aos fenômenos de transporte de calor, deve-se enfatizar a importância deste estudo, não somente em relação às bases do tratamento do estudo de caso abordado nesta dissertação, mas também no cotidiano. Todas as engenharias têm interesse imediato nos processos de transferência de calor e a própria conservação da vida está relacionada a eles. Todas as formas de transformação de energia promovem, de uma maneira ou outra, a transferência de calor.

Um exemplo de trabalho onde o fenômeno de transporte de calor é muito importante é apresentado em COLAÇO e ORLANDE (1996). Tal trabalho mostra uma análise da transferência de calor em regime transiente em um pistão de motor a diesel e apresenta visualizações dos gradientes das distribuições de temperaturas na seção do pistão.

3 FUNDAMENTOS DE VISUALIZAÇÃO EM COMPUTAÇÃO CIENTÍFICA

Para se obter uma visualização dos dados de algum fenômeno e/ou entidade é necessário, antes, fazer um estudo dos diferentes tipos de técnicas existentes. Neste capítulo são discutidos quais são os tipos de ferramentas de visualização disponíveis, em quais circunstâncias são aplicadas e porque foi utilizada a biblioteca OpenGL como biblioteca gráfica na programação do software visualizador MFSV.

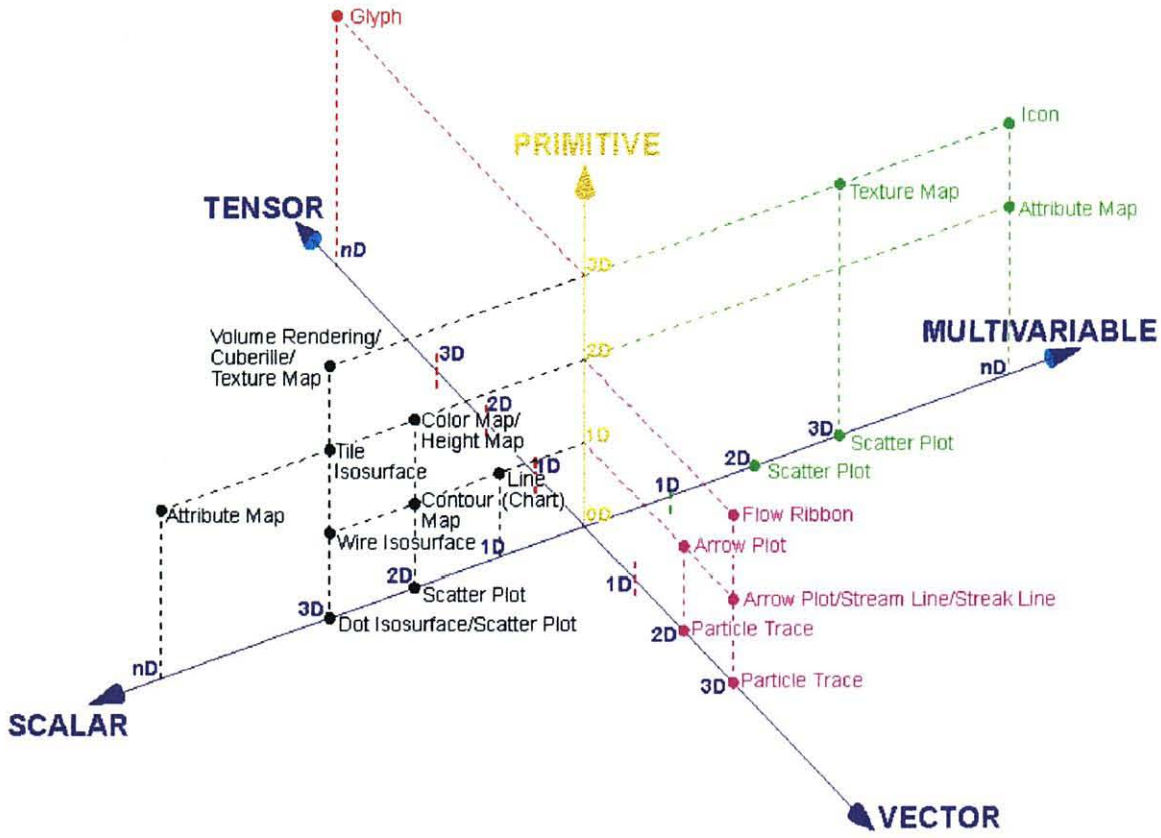
3.1 TAXIONOMIA

Muitos programas de Visualização Científica foram criados desde o final dos anos 80, sendo alguns bem completos quanto as várias ferramentas de visualização. Estas ferramentas estão classificadas segundo a dimensão do fenômeno ou entidade, as quais representam, e a dimensão da primitiva que forma o determinado tipo de ferramenta de representação. A Figura 3 apresenta a dimensão do fenômeno a visualizar segundo um dos eixos nomeados como *Scalar*, *Vector*, *Multivariable* e *Tensor*. O eixo chamado de *Primitive* possui a dimensão da primitiva utilizada na ferramenta escolhida para representar o fenômeno ou entidade. A escala da abstração das primitivas pode ser entendida pelos seguintes elementos geométricos: ponto (0D), linha (1D), superfície (2D) e volume (3D).

A pesquisa, descrita neste trabalho, começou pela análise dos visualizadores existentes e de publicações da área de Visualização Científica. Porém, também foram necessários estudos em outras áreas do conhecimento, como a Física da Transferência de Calor. Afinal, era necessário demonstrar a aplicação do MFSV em um estudo de caso adequado. Este enfoque traz mais um aspecto a ser considerado neste trabalho: A seleção,

dentre as ferramentas disponíveis, uma ferramenta a ser utilizada na representação do fenômeno escolhido para uso do MFSV.

FIGURA 3: FERRAMENTAS DE VISUALIZAÇÃO DISPONÍVEIS SEGUNDO A DIMENSÃO DO FENÔMENO



É proposto um resumo das ferramentas disponíveis para a visualização de dados como apresentado em EARNSHAW e WATSON (1993, p. 20). A Tabela 1 mostra as primitivas e suas dimensões.

TABELA 1: PRIMITIVAS E AS SUAS DIMENSÕES: [*(S)-SCALAR, (V)-VECTOR, (T)-TENSOR, (MV)-MULTIVARIABLE*]

DIMENSÃO DA PRIMITIVA	**RELAÇÃO DAS PRIMITIVAS DISPONÍVEIS**
0D - Pontos (*Points*)	*Scatter Plot(S, MV), Dot Isosurface(S), Particle Trace(V)*
1D - Linhas (*Lines*)	*Line Chart(S), Contour Map(S), Wire Isosurface(S), Arrow Plot(V), Stream/Streak Line(V)*
2D - Polígonos / Elemento de Imagem (*Polygons / Pixels*)	*Polygons: Height Map(S), Tiled Isosurface(S), Flow Ribbon(V)* *Pixels: Color Map(S), Attribute Map(S, MV)*
3D - Elementos Volumétricos (*Voxels/ Others: Glyph, Icon*)	*Voxels: Volume Rendering(S),Cuberille(S), Texture Map(S, MV)* *Others: Glyph(T), Icon(MV)*

A Tabela 2 apresenta as ferramentas de visualização segundo a dimensão dos dados a serem visualizados. Esta dimensão quase sempre representa um fenômeno da natureza em geral.

TABELA 2: FERRAMENTAS DISPONÍVEIS SEGUNDO A DIMENSÃO DE UM FENÔMENO E/OU ENTIDADE

TIPO DO FENÔMENO E/OU ENTIDADE	DIMENSÃO DO FENÔMENO A VISUALIZAR E FERRAMENTAS DISPONÍVEIS			
	1D	2D	3D	ND
Escalar (*Scalar*)	*Line Chart*	*Contour Map* *Height Map* *Color Map* *Scatter Plot*	*Scatter Plot* *Dot Isosurface* *Wire Isosurface* *Tiled Isosurface* *Texture Map* *Cuberille* *Volume Rendering*	
Vetorial (*Vector*)		*Particle Trace* *Arrow Plot*	*Particle Trace* *Arrow Plot* *Stream Line* *Streak Line* *Flow Ribbon*	
Tensorial (*Tensor*)				*Glyph*
Agregação (*Multivariable*)		*Scatter Plot*	*Scatter Plot* *Texture Map*	*Attribute Map* *Icon*

Para completar a classificação é necessário um resumo explicativo sobre cada ferramenta de visualização (EARNSHAW e WATSON, 1993, p. 20):

- A ferramenta *Line Chart* expressa o gráfico de uma propriedade na forma tradicional da matemática, onde freqüentemente a sua representação é uma função do tempo.

- *Contour Map* é o conjunto de isolinhas que conecta pontos de mesma cota. O resultado são como curvas de nível.

- Na *Height Map*, os valores bidimensionais *(x,y)*, segundo uma grade cúbica, são mapeados por uma coordenada *z*, dando a aparência de um terreno com montanhas e vales, embora nem sempre tais dados retratem de fato alturas.

• Similarmente, o *Color Map* utiliza valores bidimensionais *(x,y)*, em uma grade cúbica, mas tais valores são mapeados para uma faixa de cores, a qual é normalmente de 0 a 255. Os dados são apresentados em uma imagem de 8-bits usando uma tabela de pseudocores, a qual é projetada para realçar alguma característica dos dados. Outros mapeamentos podem ser realizados usando-se as faixas de 0 a 4095, em uma imagem de 12-bits ou de 0 a 16.777.216, em uma imagem *True Color*[32] de 24 bits. A tabela de cores é freqüentemente escolhida para relacionar-se intuitivamente com alguma percepção humana do fenômeno tratado, por exemplo: a seqüência de cores branca, vermelha, laranja, amarelo... pode representar um decréscimo da temperatura. O método tradicional é o mapeamento da luminância, onde as cores brilhantes estão relacionadas com os valores altos dos dados. Na tomografia computadorizada (abreviada por CT) é utilizada uma escala de cinza para registradores, onde o preto vale 0 e o branco tem o valor 255.

• Com a técnica *Scatter Plot* tem-se uma situação particular, onde não se associa, normalmente, as dimensões dos dados com os eixos de coordenadas. Ao invés disto, propriedades são associadas a eles. Assim, em um diagrama de fase de temperatura *versus* pressão mostraria as regiões onde um sistema químico estaria no estado sólido, líquido ou gasoso, e a técnica *scatter plot* poderia destacar as aglutinações e as tendências estatísticas. Esta técnica também é aplicável para dados agregados. Gráficos com pontos identificam as coordenadas, as quais são discretas.

• Várias *Dot Isosurfaces*[33] podem ser formadas usando-se espaçamentos regulares, com pontos em *depth-shaded*[34] ajustados na superfície. Por esta representação utilizar pontos, permite que outras *dot surfaces* sejam visíveis mais no interior ou exterior do volume.

---

[32] *True Color* indica uma distribuição de cores onde a visão do ser humano não consegue mais perceber a diferença entre duas cores subseqüentes da faixa. Tal característica proporciona a capacidade da apresentação de imagens fotorealísticas.

[33] *Isosurface* é uma superfície tridimensional dentro de um volume sobre o qual uma propriedade tem um valor em particular.

• Uma *Wire Isosurface* é similar a uma *dot isosurface*, porém é uma isosuperfície representada por linhas em *depth-shaded*, as quais são usualmente conectadas como lados de polígonos.

• As *Tiled Isosurfaces* são isosuperfícies representadas por polígonos sombreados por um dos métodos de *shading*[35], a saber: *Flat*, *Gouraud* ou *Phong*.

• A *Texture Map* é a ferramenta tridimensional equivalente à bidimensional *color map*. A *Texture Map* é uma grade tridimensional para uma faixa de cores e utiliza uma tabela de pseudocores para gerar um sólido maciço e que por projeção cria uma imagem bidimensional. A *Texture Map* é por definição uma textura tridimensionalmente sólida.

• Utilizada quase que exclusivamente em imagens de medicina, a *Cuberille* é a técnica onde a superfície de um objeto é aproximada por um conjunto de faces de cubos, os quais são sombreados para dar a aparência de uma superfície real de objeto.

• Diferentemente da *Texture Map*, na técnica de *Volume Rendering* a imagem final resultante tem uma aparência de semitransparência, como um "gel". Isto ocorre porque os valores são mapeados com cores e opacidades diferentes e todos os *voxels*[36] contribuem para a formação da imagem final.

---

[34] *Depth-shaded* é uma técnica onde a profundidade de um dado ponto gera diferenças na intensidade de cor produzida para um *pixel* em relação a outros *pixels* produzidos a partir de pontos mais próximos ou mais distantes.

[35] *Shading* é o processo de interpolação de cores no interior de um polígono, ou entre vértices de um linha, durante a rasterização.

[36] *Voxel* é o equivalente tridimensional de um *pixel*. A palavra *Voxel* vem da contração das palavras *Volume* e *Element*. A palavra *Pixel* vem da contração das palavras *Picture* e *Element* e representa a menor informação gráfica com significado.

• Na representação de fenômenos vetoriais a técnica *Particle Trace* é o rastro deixado por uma partícula[37], quando inserida em um campo vetorial, como por exemplo o das velocidades. As características de um campo vetorial são usadas para mover as partículas no tempo e no espaço. Isto requer a remontagem do campo vetorial em localizações que não correspondem aos valores discretos dos dados. Esta variação é chamada de rastro de partícula. As partículas antigas são mostradas por um certo número de passos de incremento no tempo.

• A técnica *Stream Line* é o mesmo que *particle trace* exceto que linhas tangentes, ou vetores, são acrescidas às localizações da partícula no decorrer do tempo.

• *Streak Lines* são todas as *stream lines* que passaram através de um ponto, mas originárias de localizações diferentes.

• Na ferramenta *Arrow Plot* cada vetor é mapeado sobre uma linha univocamente referenciada com a direção apontada por uma seta. São bem apropriados para representações em duas dimensões, sendo que em três dimensões, tais setas podem ser confusas dependendo da posição de onde é realizada a observação e em virtude dos efeitos da perspectiva. O valor escalar reflete o módulo do vetor.

• As *Flow Ribbons* são as *stream lines* representadas como uma fita sem espessura. A *Flow Ribbon* pode ser torcida para mostrar vórtices[38] locais em um campo vetorial.

---

[37] Partículas (ou *Path Line*, ou *Particle Path*) representam um caminho que é o lugar geométrico dos pontos ocupados no tempo por um único elemento infinitesimal do fluído ou, resumidamente, o rastro deixado por uma partícula em movimento.

[38] Vórtices são movimentos rotacionais, normalmente de um fluido, encontrados em muitos fenômenos da natureza como, por exemplo, os tornados.

• Os *Glyphs*, ou glifos, são objetos gráficos cujos elementos formadores, como posição, tamanho, forma, cor e orientação, estão relacionados aos dados. São muito úteis para descrição de situações espaciais complexas (ROSENBLUM et al., 1994, p. 106). Alguns glifos, para tensores, possuem nomes próprios como: *Jacks*, *Leme's Ellipsoid* e *Shaft-and-disk*. Estas e outras referências sobre tais glifos podem ser encontradas em EARNSHAW e DAVID (1993, p. 24).

• O *Icon* é como um glifo em 2D e possui, também, representação complexa composta de um número de conexões de segmentos de linhas. Cada um dos elementos que o forma, como comprimentos, larguras e ângulos, representam uma variável no sistema.

• A ferramenta *Attribute Map* pode ser entendida como um *color map* de uma propriedade escalar aplicado sobre uma superfície sobre a qual está outra isosuperfície derivada de uma segunda propriedade escalar, ou a superfície geométrica de um objeto como, por exemplo, a fuselagem de um aeroplano ou um reservatório de óleo.

A Visualização Científica, desde a aquisição e a modelagem dos dados até a efetiva representação, pode ser classificada sob dois pontos de vista (FREITAS et alli, 1994):

- Visualização de dados científicos
- Visualização em computação científica (ViSC)

O MFSV, proposto neste trabalho, se enquadra na segunda categoria, ViSC. O software procura, ainda que parcialmente, dar suporte ao processo de análise científica no estudo de caso escolhido, especificamente às enteléquias relacionadas às Distribuições de Temperaturas em Regime Permanente nos Fenômenos de Transporte de Calor. A escolha de uma ferramenta que venha fornecer o suporte à tarefa da visualização, mesmo que superficialmente, é uma ação trabalhosa. Também o volume e complexidade dos

dados e o tipo da representação visual a ser empregada, de acordo com a entidade e/ou fenômeno escolhido, devem seguir os critérios e metodologias previamente estabelecidos (BRODLIE, 1992).

A visualização, pelo MFSV, do estudo de caso escolhido exige, obviamente, pelo menos uma ferramenta. Para se determinar qual ferramenta seria desenvolvida para o MFSV, era necessária não somente uma descrição da organização dos resultados da experiência realizada, descrita no capítulo 4, e a natureza das grandezas físicas envolvidas, mas também uma visualização que permitisse maior destaque da proposição deste trabalho.

A metodologia empregada para a seleção de representações visuais está descrita em FREITAS (1993, p. 93) e consiste dos seguintes passos:

- Passo 1 - conhecimento das entidades e/ou fenômenos.
- Passo 2 - determinação das questões relativas aos objetivos do estudo
- Passo 3 - refinamento das questões
- Passo 4 - escolha das representações visuais
- Passo 5 - determinação das possibilidades de combinação de representações

Embora outros aspectos menos relevantes possam ter influenciado a escolha, procurou-se seguir a metodologia supra citada e o resumo da sua utilização é o seguinte:

- Passo 1 - a área Transferência de Calor envolve a propagação do calor através de meios e sob certas condições conhecidas, como descritas nos capítulos 2 e 4. As três grandezas físicas relacionadas ao estudo de caso são: a temperatura, a potência e a distância, as quais possuem o tipo escalar, a dimensão 2D, a natureza do domínio contínuo e uma distribuição contínua sob tal domínio.

- Passo 2 - a experiência descrita no capítulo 4 proporciona resultados de distribuições de temperaturas, os quais reunidos pertencem a uma superfície. O objetivo do estudo é representar esta superfície.
- Passo 3 - a superfície gerada é de característica matematicamente pertencente ao campo das funções, ou seja, não existem dois pontos sob a mesma coordenada z em relação a um plano xy, onde a potência, a distância e a temperatura são as grandezas relacionadas aos eixos x, y e z, respectivamente. Outro aspecto é o uso de filtros formulados matematicamente aplicados na superfície dos dados a serem visualizados. Por se tratar de apresentação de gradientes de temperatura, estes são melhor representados pela variação das tonalidades de cores.
- Passo 4 - após todas estas considerações, a escolha da ferramenta *Color Map* foi a mais adequada.
- Passo 5 - não foram encontradas razões para combinações de representações.

3.2 BIBLIOTECA GRÁFICA OPENGL COMO FRAMEWORK DO VISUALIZADOR

Inicialmente pensou-se em desenvolver todas as rotinas de suporte gráfico em C/C++ padrão, mas a quantidade de rotinas a serem desenvolvidas seria muito grande e haveria esforço excessivo em aspectos de programação gráfica básica, os quais não estão diretamente relacionados com a proposta deste trabalho. Então, após a análise de plataformas gráficas de desenvolvimento existentes, entre elas o *Direct 3D*[39] (TRUJILLO, 1997) do pacote *DirectX* da empresa *Microsoft*, a biblioteca gráfica escolhida foi a biblioteca multiplataforma OpenGL[40](HILL, 2000) na forma de um componente programável para um compilador de programação visual.

---

[39] Um resumo da análise da biblioteca *Direct3D* está no Anexo 1 desta dissertação.

[40] No processo de análise foi incluída uma versão de domínio público do OpenGL chamada *MESA* e informações sobre tal biblioteca podem ser encontradas no seguinte endereço: http://www.mesa3d.org/

O OpenGL é um *software* para *interface* com *hardware* gráfico e permite a criação de programas interativos que produzem imagens de alta qualidade e movimentação de objetos tridimensionais. A facilidade de uso desta biblioteca em codificação para produção de programas gráficos aliada a sua portabilidade faz com que ela seja uma das mais difundidas no mundo em seu segmento. Todavia esta biblioteca não provê funções que controlem as tarefas de controle das janelas de apresentação ou entradas do usuário no sistema. Tais tarefas são características particulares de cada sistema operacional e os recursos gráficos destes.

Primordialmente a biblioteca do OpenGL é um conjunto de aproximadamente 250 funções, das quais 200 funções são efetivamente a base da biblioteca e outras 50 funções da *Utility Library*, a qual é utilizada, especificamente, para objetos e operações necessárias à produção interativa de aplicações tridimensionais.

A biblioteca OpenGL não fornece funções prontas para a criação de objetos mais complexos como automóveis, moléculas, entre outros. Estes objetos complexos podem e são criados a partir de primitivas geométricas mais simples disponíveis na biblioteca, como pontos, linhas e polígonos. Contudo, o OpenGL fornece condições para o manuseio mais elaborado de superfícies de múltiplas faces através de recursos como o uso das curvas e superfícies *NURBS*[41] e superfícies quadráticas.

Entre outros recursos que o OpenGL inclui estão: Modelagem *Wireframe*[42], *depth-cuing*[43] através de efeitos *fog* que imitam uma neblina, *antialiasing* que é uma técnica para a remoção do indesejável efeito de serrilhado presente nas bordas das arestas de primitivas

---

[41] *NURBS (Non-Uniform Rational B-Splines)* é um tratamento matemático para malhas de superfícies.

[42] *Wireframe* é a representação, em computação gráfica, de objetos somente pelas suas linhas formadoras, sem o uso de faces sombreadas.

[43] Na técnica *depth-cuing* as cores dos *pixels* são atribuídas segundo a distância dos pontos originadores deste em relação ao ponto de visão.

gráficas quando apresentadas na tela, *rendering* nos modos *flat* e *smooth*[44], aplicação de texturas e sombras, obtenção do efeito *motion-blurred* que é efeito de imagens sobrepostas associado ao movimento de um objeto, efeito de profundidade de campo chamado de *depth-of-field* para a simulação do foco obtido por câmeras fotográficas, e muitos outros.

Há implementações de OpenGL, como a existente para o *X Window System*, que utilizam redes de computadores empregando a arquitetura cliente-servidor, onde um computador é responsável pelo processamento, enquanto os outros computadores desta rede podem apresentar os gráficos gerados. O formato ou protocolo que o OpenGL usa para transferência dos dados é sempre o mesmo, indiferente aos tipos de computadores ligados à rede. Quando um único computador utiliza o OpenGL, este será o cliente e o próprio servidor.

Não foram estudados todos os aspectos da biblioteca de rotinas gráficas do OpenGL, mas o suficiente para os propósitos deste trabalho. Dentre os estudos foram considerados os seguintes tópicos[45], encontrados em WOO (1997):

- Fundamentos da biblioteca OpenGL
- Gerenciamento de Estados e Objetos Geométricos
- Posição do Observador e dos Objetos
- Uso de Cores
- Canal Alfa[46]
- Uso de *Bitmaps*[47]

---

[44] *Flat* e *Smooth*, esta última também conhecida por *Gouraud* são algumas das técnicas de *rendering* disponíveis para os programas de computação gráfica.

[45] Para uma descrição dos tópicos sobre o OpenGL vide Anexo 1.

[46] Uma cor é coposta pelas 3 componentes das cores primárias mais a informação do canal alfa. Esta informação informa o quão transparente um *pixel* é em relação aos *pixel* calculados para a mesma posição.

[47] *Bitmap* é uma imagem *raster*, ou seja formada por *pixels*, a qual é armazenada na memória *RAM* ou secundária.

4 EXPERIMENTO

Para demonstrar o uso do software protótipo MFSV, e a proposição desta dissertação, foi usado um fenômeno ligado à área de Transferência de Calor. Entretanto, também foi necessária a realização de uma experiência para obtenção de dados sobre este fenômeno escolhido. Todas as fases da experiência, a metodologia empregada para a obtenção de resultados coerentes com a realidade, a formulação matemática e a listagem dos resultados obtidos estão apresentados neste capítulo da dissertação.

4.1 DISTRIBUIÇÃO DE TEMPERATURAS AO LONGO DE UMA BARRA DE AÇO EM REGIME PERMANENTE POR CONDUÇÃO UNIDIMENSIONAL DE CALOR

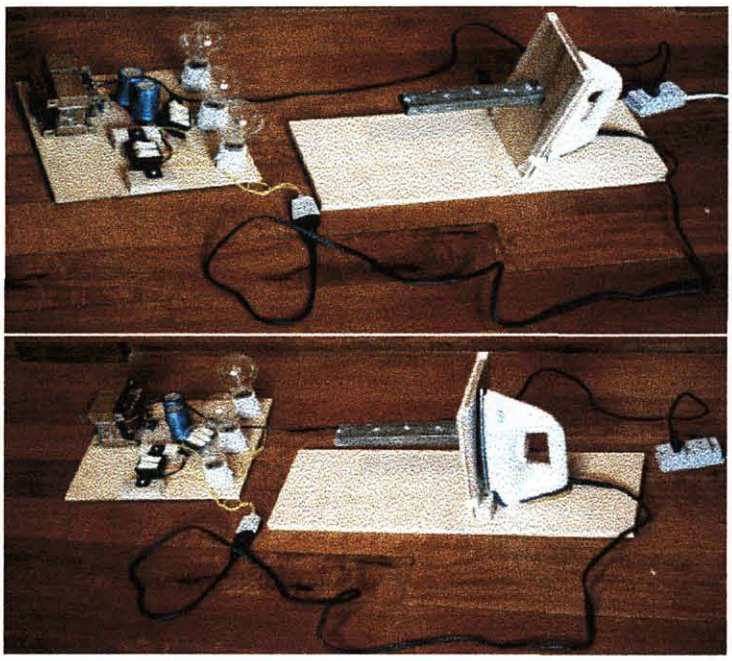
O propósito da experiência é mostrar como se dá a distribuição de temperaturas, em regime permanente e condução de calor considerada como uma propagação unidimensional[48], ao longo de uma barra de aço disposta horizontalmente em relação ao solo, a qual é aquecida pela extremidade que a sustenta no sistema que fornece a energia térmica. A barra está apoiada em balanço e tem seção retangular. O objetivo da experiência para o presente trabalho foi obter várias medições de várias distribuições de temperaturas e representá-las em uma mesma visualização. Para obter todos os dados referentes à construção de um artefato, o qual era necessário à realização da experiência, foram consultados pesquisadores de laboratórios ligados a instituições de pesquisa na área de Transferência de Calor. As medições não foram realizadas nos referidos laboratórios, porque a obtenção delas é um processo muito lento e, portanto, foi optado pela reprodução da experiência em dependências particulares, considerados todos os cuidados que tal experiência exige.

---

[48] Uma publicação para a propagação bidimensional está descrita no trabalho de JUCÁ & PRATA (1991)

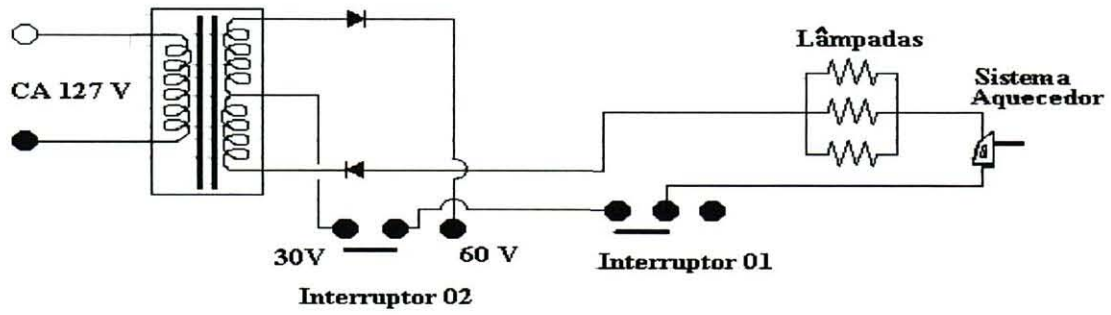
O artefato foi idealizado e construído com base nos dados obtidos. Para o uso deste equipamento na realização da experiência foi utilizada uma sala termicamente climatizada para que a temperatura ambiente permanecesse constante de tal forma que permitisse a reprodução fiel das condições utilizadas nos laboratórios de máquinas térmicas. A Figura 4 apresenta fotos do artefato empregado para realizar a experiência.

FIGURA 4: ARTEFATO UTILIZADO PARA A REALIZAÇÃO DA EXPERIÊNCIA



Este artefato pode ser representado na forma de um diagrama elétrico como o sugerido pela Figura 5, a qual apresenta um diagrama das partes mais importantes do artefato.

FIGURA 5: DIAGRAMA ELÉTRICO DO ARTEFATO UTILIZADO NA EXPERIÊNCIA



Houve muitos ajustes até que resultados condizentes pudessem ser obtidos. A tensão e a corrente alternada provenientes da rede foram inapropriadas para a ligação direta

do aquecedor. Foi necessária a transformação da corrente alternada para contínua e baixar a diferença de potencial de 127V para 60V ou 30V, dependendo das necessidades.

Para fazer com que a potência fornecida ao aquecedor permitisse um ajuste variável no valor, mas que se mantivesse com flutuações mínimas para o valor escolhido, foram utilizadas lâmpadas com potências diferenciadas, as quais funcionam como resistências no circuito. A combinação de lâmpadas com potências variadas e sob tensão de 30V ou 60V permitiu obter distribuições de temperaturas dentro da faixa esperada.

O aquecedor utilizado foi um ferro elétrico de passar roupas modificado para atender as condições de regime permanente na transferência de calor. Uma barra de aço com base de 37mm e altura de 18,8mm na seção transversal, medidas aferidas com uso de paquímetro, e com um comprimento de 184mm foi fixada com rebites no centro da base do ferro elétrico. O ferro elétrico, por sua vez, foi fixado verticalmente, em uma base de madeira, para que a barra de aço ficasse perfeitamente na horizontal. Isto é necessário, porque os resultados seriam diferentes com a barra de aço na vertical ou em outra angulação intermediária. Existe a influência nos resultados da experiência pelo fluxo de ar que sobe após ser aquecido. A área excedente da base aquecedora do ferro elétrico também poderia interferir na camada de ar ao redor da barra de aço. Para evitar tal interferência foi colocada sobre esta área uma placa de cerâmica para o isolamento térmico. Foram feitos 6 furos em uma linha central, em posições aferidas, ao longo do comprimento da barra de aço. Estes furos foram utilizados como pontos[49] de leitura das temperaturas, utilizando-se um termômetro digital[50]

Outro termômetro[51] foi utilizado dentro do ambiente, mas suficientemente longe do artefato para verificar a temperatura nas vizinhanças do artefato. A temperatura ambiente deveria permanecer constante. Foram consideradas as propriedades físicas do metal, o aço, e outras condições para a realização da experiência. Finalmente, os resultados obtidos foram confrontados com resultados obtidos em relatórios de experiências similares realizadas em

---

[49] Será utilizado o termo ponto, doravante, para a localização destes furos sobre a barra de aço.

[50] Termômetro digital de haste com resolução de 1 casa decimal (0,1°C).

[51] Termômetro clássico de bulbo.

laboratórios. A conclusão foi que as divergências de valores ficaram muito pequenas e dentro dos limites de variação tolerados para a experiência.

A experiência foi realizada com o intuito de obter pelo menos 5 distribuições de temperaturas. Cada distribuição de temperaturas era constituída por um conjunto de 6 medições, uma para cada ponto existente na barra de aço, em relação a um valor constante de potência fornecida ao sistema aquecedor. Para cada uma das 5 distribuições foi fornecido um valor diferente de potência. Tal quantidade de dados obtidos da experiência foi suficiente para a dedução das fórmulas matemáticas que dariam a continuidade no domínio de valores dos dados a serem visualizados.

Um dos aspectos mais importantes foi o método adotado para realizar as medições em cada um dos pontos na barra de aço. Tal método compõe-se dos seguintes passos para cada distribuição de temperaturas realizada:

- Estabelecer uma potência constante ao sistema aquecedor.
- Colocar o termômetro digital de haste no ponto da barra de aço mais próximo à base do sistema aquecedor, ponto rotulado como *d1* na Tabela 3.
- Após atingido um valor constante para a temperatura, transferir o termômetro para o ponto mais distante da base do sistema aquecedor, na extremidade oposta da barra de aço e verificar o valor constante final para temperatura neste ponto.
- Caracterizado o regime permanente deve-se começar as medições para cada ponto em um sentido e no sentido oposto, conforme a ordem da seqüência dos pontos sobre a barra de aço. Em cada ponto o termômetro deve permanecer até ser atingido o valor final de temperatura.
- O passo anterior deve ser repetido pelo menos 4 vezes para cada distribuição de temperaturas desejada. Isto garante uma uniformidade estatística para os valores apurados. A convenção assumida foi considerar os maiores valores de temperatura registrados em cada ponto.

Uma observação importante é que a diferença entre a maior e a menor temperatura, em um determinado ponto, nunca excedeu 0,1°C. Valor que, *a priori*, pode ser considerado desprezível para os propósitos deste trabalho.

Com base neste método foram obtidos os valores necessários ao presente trabalho e a Tabela 3 mostra tais valores e as condições iniciais para cada distribuição de temperaturas:

TABELA 3: CONDIÇÕES INICIAIS PARA CADA DISTRIBUIÇÃO DE TEMPERATURAS E OS VALORES OBTIDOS

DISTÂNCIAS DOS PONTOS EM RELAÇÃO A BASE DO SISTEMA AQUECEDOR	MEDIÇÕES DISTRIBUIÇÃO Nº 1 EM °C	MEDIÇÕES DISTRIBUIÇÃO Nº 2 EM °C	MEDIÇÕES DISTRIBUIÇÃO Nº 3 EM °C	MEDIÇÕES DISTRIBUIÇÃO Nº 4 EM °C	MEDIÇÕES DISTRIBUIÇÃO Nº 5 EM °C
D1 = 0 mm	41,0 °C	50,8 °C	59,1 °C	66,3 °C	81,3 °C
D2 = 17 mm	40,2 °C	49,7 °C	57,2 °C	64,3 °C	77,9 °C
D3 = 47 mm	38,9 °C	47,8 °C	54,2 °C	61,0 °C	72,4 °C
D4 = 89 mm	37,6 °C	46,0 °C	51,3 °C	56,8 °C	67,1 °C
D5 = 130 mm	36,8 °C	44,9 °C	49,4 °C	54,3 °C	63,7 °C
D6 = 184 mm	36,4 °C	44,2 °C	48,3 °C	52,8 °C	61,7 °C
TENSÃO (V) NOMINAL ESCOLHIDA PARA CADA DISTRIBUIÇÃO.	30,00 V	60,00 V	30,00 V	60,00 V	60,00 V
TENSÃO (V) EFETIVA	7,90 V	11,10 V	13,30 V	15,50 V	19,30 V
CORRENTE CC (A) EFETIVA	0,5850 A	0,8250 A	0,9830 A	1,1480 A	1,4300 A
RESISTÊNCIA (Ω) EFETIVA	13,50 Ω	13,50 Ω	13,50 Ω	13,50 Ω	13,50 Ω
POTÊNCIA (W) EFETIVA	4,62 W	9,13 W	13,07 W	17,80 W	27,60 W

Os valores das grandezas efetivas da Tabela 3 são os utilizados para efeito de cálculo da potência chamada de efetiva no sistema aquecedor. Deve-se notar, que o cálculo desta potência pode ser realizado por qualquer uma das duas fórmulas da Eletricidade Clássica:

$$P = R\,i^2 \qquad (8)$$

ou

$$P = U\,i, \qquad (9)$$

onde P é a potência (*Watts*), R é a resistência elétrica (*Ohms*), U é a tensão (*Volts*) e i é o valor da corrente elétrica (*Amperes*). As formulações matemáticas para a visualização estão em função da potência efetiva e temperatura inicial dada no ponto rotulado como *d1* na Tabela 3.

Foram realizadas medições de outras grandezas físicas, além da temperatura, a saber: Tensão, corrente e resistência no circuito paralelo das resistências, ou seja as lâmpadas. Estes valores foram utilizados como referências para a confirmação da coerência dos valores obtidos. Entretanto, estes valores não têm grande significado para o presente trabalho e muitos destes valores estão disponíveis no Anexo 2, estando presente, também, os valores das potências das lâmpadas e os valores de resistências elétricas equivalentes a estas lâmpadas.

Um ponto importante a observar é que devido as oscilações naturais da rede elétrica dificilmente duas medições serão iguais em um intervalo de tempo relativamente grande, porque afetam o valor efetivo da potência na fonte do circuito. Portanto, as medições de uma mesma distribuição devem ser realizadas, se possível, em um período do dia com menos picos de tensão, mesmo considerando o fato do uso de estabilizadores de tensão, como foi empregado na experiência descrita neste trabalho.

4.2 FORMULAÇÃO MATEMÁTICA

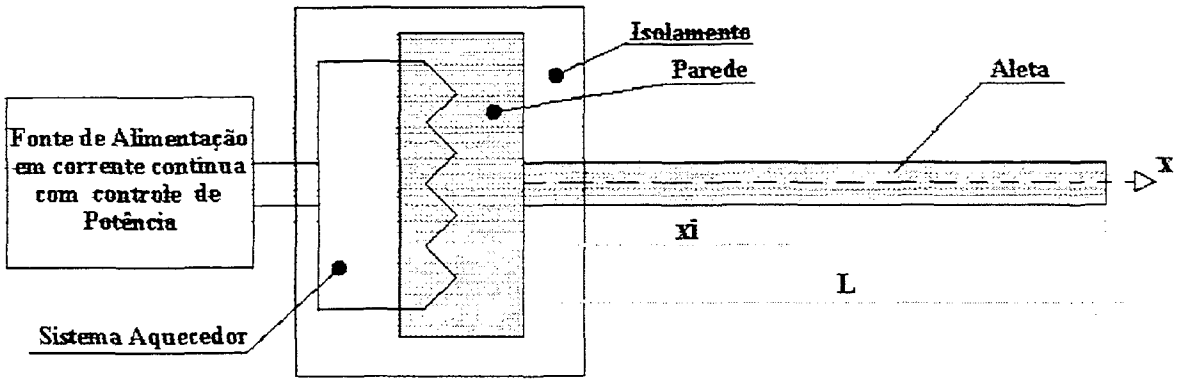
O objetivo desse item é explicar todos os detalhes que conduzirão à formulação matemática de um modo geral, começando pelas questões teóricas relacionadas ao fenômeno e sua abordagem pela Física e passando para a formulação matemática de dados destinados à visualização.

4.2.1 Análise Física do Fenômeno de Transferência de Calor em Aletas

Aletas são superfícies estendidas usadas em referência a um sólido onde há transferência de energia por condução no interior de suas fronteiras e transferência de energia por convecção e/ou radiação entre suas fronteiras e a vizinhança. A aplicação mais freqüente compreende a utilização de uma aleta para, especificamente, aumentar a taxa de transferência de calor entre um sólido e um fluido adjacente (INCROPERA, 1990, p. 55).

Considerando o artefato destinado à realização da experiência como descrita anteriormente neste trabalho, pode-se representar o sistema aquecedor segundo um esquema. A Figura 6 apresenta uma sugestão de esquema do sistema aquecedor.

FIGURA 6: ESQUEMA DO SISTEMA AQUECEDOR



Para efeito da apresentação da formulação matemática do fenômeno segundo à Física, será efetuado cálculo dos valores teóricos para comparação com os valores das medições realizadas para uma distribuição de temperaturas qualquer ao longo dos 6 pontos sobre a barra de aço, ou aleta[52]. Pela realização das medições foram obtidos os valores de temperaturas para cada um dos pontos como estão apresentados na Tabela 4.

TABELA 4: VALORES DAS TEMPERATURAS PARA OS 6 PONTOS SOBRE A ALETA

PONTO DE MEDIÇÃO	1	2	3	4	5	6
DISTÂNCIAS NO EIXO X DADA EM MILÍMETROS	0	17,0	47,0	89,0	130,0	184,0
TEMPERATURA MEDIDA °C	41,0	40,2	38,9	37,6	36,8	36,4

Todos os valores de propriedades físicas foram registrados para a realização das medições. Estes dados foram registrados para posterior utilização nos cálculos teóricos e relacionados na Tabela 5.

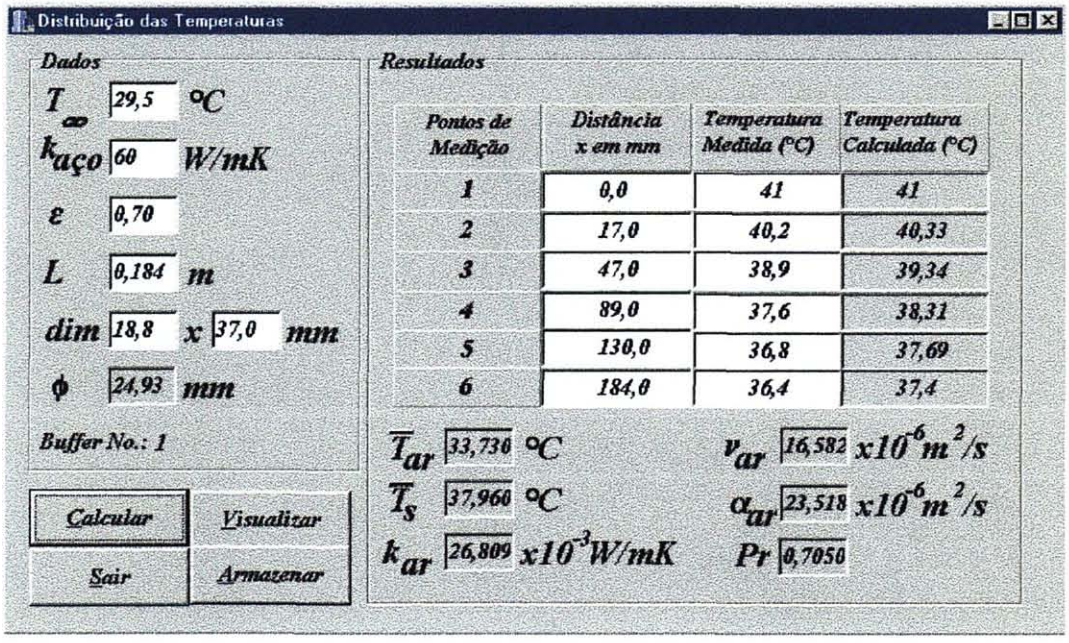
[52] O termo aleta não havia sido utilizado em virtude da sua definição ser realiazada no texto e contexto deste capítulo, e será utilizado a partir deste capítulo por ser mais técnico.

TABELA 5: VALORES DAS PROPRIEDADES AVALIADAS PARA A REALIZAÇÃO DAS MEDIÇÕES

DESCRIÇÃO DA PROPRIEDADE FÍSICA	VALOR OU NATUREZA DA PROPRIEDADE
Temperatura do ar ambiente	$t_\infty = 29{,}5^o C = 302{,}65K$
Número pontos sobre a aleta	$N=6$
Material da aleta	*Aço*
Condutividade térmica da aleta	$k \cong 60W/mK$ (valor arbitrário)
Estado superficial da aleta	*Jateado*
Emissividade da superfície da aleta	$\varepsilon \cong 0{,}70$ (valor arbitrário)
Comprimento da aleta	$L = 0{,}184m$
Dimensões da seção transversal	$(18{,}8x37{,}0)mm^2$

Para a seqüência de cálculos a serem realizados foi criado um pequeno programa chamado DTemp[53], cuja janela do ambiente está apresentada na Figura 7. O programa DTemp auxiliou enormemente o desenvolvimento do programa final de visualização, o MFSV. Entretanto, para fins didáticos, será mostrada a seqüência manual das passagens de cálculo[54].

FIGURA 7: PROGRAMA DTEMP E UMA DISTRIBUIÇÃO DE TEMPERATURAS CALCULADA



[53] O nome deste programa auxiliar vem das palavras Distribuição de Temperaturas.

[54] Esta seqüência é encontrada em relatórios de experiências similares realizadas em laboratórios de instituições de ensino e pesquisa na área de Engenharia Mecânica.

a) O primeiro cálculo a ser realizado é o da temperatura média t_s da aleta e será utilizado o método clássico do ponto médio para integração dos dados discretos obtidos na medição.

$$t_s = \frac{1}{L}\int_0^L f(x_i)dx. \qquad (10)$$

$$t_s = [\,(41+40{,}2)\times 17+(40{,}2+38{,}9)\times(47-17)+ (38{,}9+37{,}6)\times(89-47)+(37{,}6+38{,}9)\times(130-89)+ (36{,}8-36{,}4)\times(184-130)]/(2\times 184),$$

$$\therefore \quad t_s = 37{,}96^{o}C = 311{,}11K.$$

b) Cálculo da temperatura média do ar:

$$t_{mar} = \frac{t_s + t_\infty}{2} = \frac{37{,}96+29{,}5}{2}, \quad \therefore t_{mar} = 33{,}73^{o}C = 306{,}88K \qquad (11)$$

c) Propriedades físicas do ar levando-se em conta: α, υ, Pr e K. Dados obtidos por interpolação linear dos valores em INCROPERA (1990, p. 432) e na temperatura média do ar t_{mar} e que estão relacionados na Tabela 6:

TABELA 6: PROPRIEDADES FÍSICAS DO AR OBTIDAS POR INTERPOLAÇÃO LINEAR

DESCRIÇÃO DA PROPRIEDADE FÍSICA DO AR	**VALOR DA PROPRIEDADE FÍSICA**
Viscosidade cinemática do ar (m^2/s)	υ = 16,582x10^{-6}m^2/s
Difusidade térmica do ar (m^2/s)	α = 23,518x10^{-6}m^2/s
Número de Prandt	Pr = 0,7050
Condutividade térmica do ar	K = 26,809x10^{-3}W/mK

FONTE: (INCROPERA, 1990, P. 432)

d) Cálculo do diâmetro equivalente da aleta de seção retangular como sendo de seção circular:

$$D = \frac{4A}{P} = \frac{4 \times (18{,}8 \times 37)}{2 \times 18{,}8 + 2 \times 37}, \qquad (12)$$

$$\therefore \quad D = 24{,}93mm = 2{,}493x10^{-2}m.$$

e) Cálculo do coeficiente de expansão térmica do ar:

$$\beta = \frac{1}{273{,}15 + t_{mar}}, \qquad (13)$$

$$\therefore \quad \beta = 3{,}2686x10^{-3}.$$

f) Cálculo do número de Rayleigh:

$$R_{aD} = \frac{g\beta(t_s - t_\infty)D^3}{\upsilon\alpha}. \qquad (14)$$

$$R_{aD} = \frac{\left[9{,}81 \times 3{,}2586x10^{-3} \times (37{,}96 - 29{,}5) \times \left(24{,}93x10^{-3}\right)^3\right]}{16{,}582x10^{-6} \times 23{,}518x10^{-6}},$$

$$\therefore \quad R_{aD} = 10744{,}84.$$

g) Cálculo do número de Nusselt:

$$N_{uD} = \left\{ 0{,}6 + \frac{0{,}387 \times R_{aD}^{1/6}}{\left[1 + \left(\frac{0{,}559}{\Pr} \right)^{9/16} \right]^{8/27}} \right\}^2 \qquad (15)$$

$$N_{uD} = \left\{ 0{,}6 + \frac{0{,}387 \times 10744{,}84^{1/6}}{\left[1 + \left(\frac{0{,}559}{0{,}7050} \right)^{9/16} \right]^{8/27}} \right\}^2, \quad \therefore \quad N_{uD} = 4{,}445.$$

h) Cálculo do coeficiente de convecção:

$$h_{con} = \frac{N_{uD} \times k}{D} = \frac{4{,}445 \times 26{,}809x10^{-3}}{24{,}93x10^{-3}}, \qquad (16)$$

$$\therefore \quad h_{con} = 4{,}780 W / m^2 K.$$

i) Cálculo do coeficiente de radiação:

$$q_{rad} = \varepsilon\sigma\left(T_S^4 - T_\infty^4\right) = 0{,}7 \times 5{,}67x10^{-8} \times \left(311{,}11^4 - 302{,}65^4\right), \qquad (17)$$

$$\therefore \quad q_{rad} = 38{,}82 W / m^2, \quad \text{e}$$

$$h_{rad} = \frac{q_{rad}}{(t_S - t_\infty)} = \frac{38{,}82}{37{,}96 - 29{,}5}, \qquad (18)$$

$$\therefore \quad h_{rad} = 4{,}589 W / m^2 K.$$

j) Coeficiente de transferência de calor *h*:

$$h = h_{con} + h_{rad} = 4{,}780 + 4{,}589, \qquad (19)$$

$$\therefore \quad h = 9{,}369 W / m^2 K.$$

k) Cálculo do fator m:

$$m = \sqrt{\frac{hp}{kA}} = \sqrt{\frac{9{,}369 \times \left(2 \times 18{,}8x10^{-3} + 2 \times 37x10^{-3}\right)}{60 \times 18{,}8x10^{-3} \times 37x10^{-3}}}, \qquad (20)$$

$$\therefore \quad m = 5{,}0052 m^{-1}.$$

Após todos estes cálculos, os quais têm como objetivo obter o fator *m*, obtém-se o cálculo da distribuição das temperaturas ao longo da aleta, que são dadas pela seguinte equação (INCROPERA, 1990, p. 58):

$$t_i = \frac{\left(t_o - t_\infty\right) \times \cosh\left(m\left(L - x_i\right)\right)}{\cosh(mL)} + t_\infty. \qquad (21)$$

Com base na equação *(21)* pode-se calcular todas as temperaturas ao longo da aleta bastando, conhecer a temperatura inicial na base da aleta. A Tabela 7 apresenta os valores obtidos pelas medições comparadas com os valores calculados pela equação *(21)*.

TABELA 7: UMA COMPARAÇÃO ENTRE OS VALORES DAS MEDIÇÕES E OS VALORES CALCULADOS.

PONTO DE MEDIÇÃO	**1**	**2**	**3**	**4**	**5**	**6**
DISTÂNCIA NO EIXO X DADA EM MILÍMETROS	0	17,0	47,0	89,0	130,0	184,0
TEMPERATURA MEDIDA EM °C	41,0	40,2	38,9	37,6	36,8	36,4
TEMPERATURA CALCULADA EM °C	41,00	40,33	39,34	38,31	37,69	37,40

É importante notar que o desvio entre a temperatura calculada e medida é pequeno, em torno de 1°C. Obviamente, estas discrepâncias são devidas as oscilações da fonte, pequenas variações na temperatura ambiente, erros de arredondamentos, valores arbitrários estabelecidos, o fato da equação do número de Nusselt ser empírica, entre outras.

Foram calculados os valores teóricas, através do programa DTemp, e estes valores estão relacionados na Tabela 8. Esta tabela apresenta os valores obtidos pelas medições, os quais podem ser comparados com os valores calculados.

TABELA 8: VALORES CALCULADOS E DAS 5 DISTRIBUIÇÕES DE TEMPERATURAS DAS MEDIÇÕES

PONTO DE MEDIÇÃO	**1**	**2**	**3**	**4**	**5**	**6**
DISTÂNCIA NO EIXOX DADA EM MILÍMETROS	**0**	**17,0**	**47,0**	**89,0**	**130,0**	**184,0**
1ª DISTRIBUIÇÃO DE TEMPERATURAS MEDIÇÕES EM °C	41,0	40,2	38,9	37,6	36,8	36,4
1ª DISTRIBUIÇÃO DE TEMPERATURAS VALORES CALCULADOS EM °C	41,00	40,33	39,34	38,31	37,69	37,40
2ª DISTRIBUIÇÃO DE TEMPERATURAS MEDIÇÕES EM °C	50,8	49,7	47,8	46,0	44,9	44,2
2ª DISTRIBUIÇÃO DE TEMPERATURAS VALORES CALCULADOS EM °C	50,8	49,46	47,48	45,44	44,21	43,64
3ª DISTRIBUIÇÃO DE TEMPERATURAS MEDIÇÕES EM °C	59,1	57,2	54,2	51,3	49,4	48,3
3ª DISTRIBUIÇÃO DE TEMPERATURAS VALORES CALCULADOS EM °C	59,10	57,15	54,28	51,35	49,57	48,75
4ª DISTRIBUIÇÃO DE TEMPERATURAS MEDIÇÕES EM °C	66,3	64,3	61,0	56,8	54,3	52,8
4ª DISTRIBUIÇÃO DE TEMPERATURAS VALORES CALCULADOS EM °C	66,30	63,80	60,12	56,37	54,11	53,05
5ª DISTRIBUIÇÃO DE TEMPERATURAS MEDIÇÕES EM °C	81,3	77,9	72,4	67,1	63,7	61,7
5ª DISTRIBUIÇÃO DE TEMPERATURAS VALORES CALCULADOS EM °C	81,30	77,61	72,18	66,66	63,34	61,8

4.2.2 Formulação Matemática para a Visualização

Logicamente não seria viável utilizar o mesmo procedimento de medições e cálculos para se estabelecer uma massa suficientemente grande de dados para a visualização. Portanto, procurou-se obter uma equação para o fator *m*. Para tal equação, a ser determinada, os valores de temperatura ambiente e características físicas de materiais deveriam se manter

constantes. Como foram obtidas 5 distribuições de temperaturas procurou-se, a partir destes valores, deduzir as equações que dessem a continuidade matemática, para os dados, necessária ao tipo de representação pretendido à visualização, a saber: Uma visualização utilizando a ferramenta *Color Map*, onde a superficie visualizada representa todas as temperaturas em um domínio finito e contínuo de valores.

A primeira equação é para a temperatura incial da aleta, normalmente a mais alta. Tal equação está em função da potência da fonte. Arbitrando, de acordo com as características físicas do fenômeno, que tal relação tivesse um formato logarítmico, foram determinados os coeficientes constantes e a seguinte equação para a temperatura inicial da aleta:

$$t_{oi} = 72{,}0122 \ln\left(3{,}815x10^{-2} \times P_i + 1\right) + 29{,}5. \qquad (22)$$

A variável P_i é a potência ajustada na fonte de energia para o sistema aquecedor.

Após a dedução da equação *(22)*, foi necessária a dedução de uma equação que descrevesse qualquer fator m em função da temperatura inicial T_{oi} calculada pela equação *(22)*, sempre considerando a temperatura ambiente igual a 29,5°C. Da mesma forma, foi suposta uma descrição logarítmica para os valores calculados de m e foram determinados coeficientes constantes e a seguinte equação:

$$m = \frac{0{,}5474 \ln\left(0{,}9262 \times \theta_i + 1\right) + \theta_i^{\,2}}{574{,}0444 \times e^{\left(\theta_i^{\,2}/132{,}25\right)}} + 3{,}5765 \quad , \qquad (23)$$

onde $\theta_i = t_{oi} - t_{\infty}$. (24)

A variável θ_i é chamada de excesso de temperatura (INCROPERA, 1990, p. 57).

Estas duas equações permitiram, com bastante precisão, obter dados de forma contínua num determinado intervalo de temperaturas iniciais e potências para a fonte. Além destas duas equações *(22)* e *(23)*, foi necessário obter mais uma equação que servisse como um exemplo de filtro físico-matemático. A relação deduzida a partir da equação *(21)* foi:

$$x_i = L - \frac{1}{m}\left\{\arccos h\left[\frac{(t - t_\infty)}{t_{oi} - t_\infty} \times \cosh(Lm)\right]\right\}, \qquad (25)$$

onde x_i é a distância na aleta onde ocorre a temperatura t dada. O valor de x_i calculado deverá estar compreendido no intervalo: $0 \leq x_i \leq L$ e a temperatura escolhida para o cálculo de x_i deverá estar no intervalo desde T_o até a temperatura mais baixa na extremidade oposta da aleta.

Todas estas formulações deram ao software de visualização desenvolvido um exemplo de aplicação de filtros físico-matemáticos.

5 FILTROS FÍSICO-MATEMÁTICOS EM VISUALIZAÇÃO CIENTÍFICA

Este capítulo apresenta a fundamentação teórica dos filtros formulados matematicamente, a sua axiomatização e como foram aplicados filtros no estudo de caso deste trabalho, o qual envolve fenômenos da área de Transferência de Calor.

5.1 IDEALIZAÇÃO

Muitos problemas se apresentam, ao usuário, com uma configuração difícil para a escolha por seções puramente geométricas ou contornadas por valores de fronteira da região a ser representada em programas de visualização, como mostrado no exemplo a seguir:

O problema de modelagem e compreensão do fenômeno da convecção é um dos maiores problemas na compreensão da evolução das estrelas.

Durante a evolução de uma estrela qualquer, a fonte de energia que a faz brilhar tem origem na fusão termonuclear de elementos químicos mais leves em mais pesados. A partir da região de produção de energia, que pode ser junto ao centro da estrela e/ou em uma, ou mais camadas esféricas que não estão no núcleo, onde as temperaturas atingem dezenas de milhões de Kelvin, até a superfície da estrela, onde a temperatura cai para alguns milhares de Kelvin e que efetivamente brilha no visível, existe um grande percurso que pode levar centenas de milhares de anos entre absorções, espalhamentos e reemissões de fótons. Estes processos de transporte de energia podem ser modelados analogamente aos processos de convecção, condução e de transporte radiativo. Nas diversas fases evolutivas de uma estrela, em função de sua massa, composição, entre outras características, apresentam-se diversas estruturas nas zonas onde determinado processo domina o transporte de energia. O estudo da

evolução estelar não está somente intimamente ligado a compreender e visualizar a física da produção de energia mas também pelos seus processos de transporte. Assim, um visualizador adequado se faz necessário, capaz de tratar intensamente dados obtidos por simulações numéricas de evolução estelar. Estes tratamentos não se restringiram somente a diversidade dos valores em si, mas em relações destes dentro de um determinado instante, estendendo-se a relações entre os dados anteriores e os atuais. Pode-se citar, por exemplo, a região onde ocorre a convecção que pode ser caracterizada pela expressão:

$$\frac{P}{T}\frac{dT}{dP} > \frac{Gamma - 1}{Gamma}, \qquad (26)$$

onde T e P são respectivamente a temperatura e a pressão em uma posição r (vetor) dentro da estrela, e *Gamma* a razão dos calores específicos principais. As variação dT e dP podem ser tanto tomadas posicionais quanto temporais.

Neste exemplo percebe-se que as alterações dos dados, os quais seriam candidatos à visualização, são fortemente condicionadas a equacionamentos físicos e altamente dependentes do tempo. Nesta situação, um visualizador com ferramentas adequadas de visualização e que empregue filtros físico-matemáticos poderia criar visualizações que não somente filtrariam regiões de dados, como a região convectiva citada no exemplo, mas permitiria fazer um acompanhamento, em função do tempo, dos diversos estágios da evolução da estrela, o que permitiria, também, uma comunicação mais dinâmica com os programas de simulação.

Este é apenas um exemplo dentre vários que existem não somente no campo da Física, mas em outras áreas do conhecimento. A evidência que as filtragens formuladas matematicamente possam ser úteis em várias situações é, portanto, confirmada.

5.2 AXIOMATIZAÇÃO

Postulou-se os Filtros Formulados Matematicamente como sendo uma representação matemática tal qual uma equação ou função física de um fenômeno, uma matriz ou sistemas de equações lineares ou não-lineares, além de outras representações, cuja utilização em um programa de visualização é para a obtenção de uma seção dos dados, a qual deve ou não ser visualizada ou, ainda, entendida como uma amostragem do universo total possível dos dados gerados a partir de um fenômeno.

A hipótese admitida à proposição fundamental enunciada no capítulo 1 deste trabalho é: Programas de visualização existentes ou desenvolvidos poderiam realizar interativamente a filtragem de dados através de formulações matemáticas como uma opção natural do programa visualizador e vinculada, portanto, às características das ferramentas de visualização deste software.

Uma das formas de demonstrar tal premissa era desenvolver um programa visualizador que utilizasse os filtros formulados matematicamente. Desta forma, foi desenvolvido o *Mathematical Filters in Scientific Visualization - MFSV 1.0* e aplicado ao estudo de caso proposto para este trabalho.

5.3 FILTRO FÍSICO-MATEMÁTICO APLICADO AO ESTUDO DE CASO

O fenômeno Físico da Transferência de Calor por condução, descrito pelas equações anteriormente apresentadas, será visualizado como uma superfície no espaço tridimensional dentro dos limites estabelecidos e medições realizadas. Pode-se supor a necessidade da visualização de uma região desta superfície, a qual estivesse compreendida ou limitada por uma ou mais equações, chamadas, aqui, de equações filtrantes do fenômeno.

5.3.1 Um Filtro Formulado Matematicamente Aplicado ao Estudo de Caso Escolhido

No capítulo 4 foram formuladas matematicamente algumas equações e a equação *(25)* foi desenvolvida com o propósito de ser um exemplo de filtro aplicado ao estudo de caso escolhido. Tal equação será, por questão de praticidade, exposta novamente aqui:

$$x_i = L - \frac{1}{m}\left\{\arccos h\left[\frac{(t - t_\infty)}{t_{oi} - t_\infty} \times \cosh(Lm)\right]\right\}. \qquad (25)$$

A equação *(25)* será entendida como um filtro físico-matemático aplicado ao estudo de caso escolhido e, tal equação, significa em termos físicos: Para uma determinada temperatura t, informada, uma distância x_i, a partir da origem da aleta, é calculada. Desta forma, pode-se utilizar esta equação como filtrante ou limitante da região dos dados a ser visualizada. Deve-se ressaltar que os dados gerados, para qualquer temperatura, são obtidos a partir da equação *(21)*, a qual é também reapresentada para fins de praticidade:

$$t_i = \frac{(t_o - t_\infty) \times \cosh(m(L - x_i))}{\cosh(mL)} + t_\infty. \qquad (21)$$

Suposição proposta: Calcular todos os valores de x_i onde a temperatura t é maior que t_1 e menor que t_2. Obviamente, existem valores que são impróprios dependendo da parametrização imposta e tal impropriedade está relacionada a condições de dimensões finitas e restrições físicas do fenômeno. Muitas destas restrições são reveladas por valores impróprios ou fora do domínio de funções matemáticas calculadas a exemplo da função *arccosh()*. Todas estas considerações podem ser traduzidas em implementações de programas e são do conhecimento do usuário e pesquisador do fenômeno em questão. Pode-se verificar facilmente que existem valores de t para os quais não existe um x. A aleta tem um comprimento finito e determinado, qualquer valor de x que seja menor que *0* ou superior a L não tem significado físico prático ($0 \leq x_i \leq L$).

5.3.2 Resultados Obtidos com o Programa MFSV

Sabendo-se que a temperatura ambiente, durante a experiência descrita no capítulo 4, foi de 29,5°C, o comprimento L da aleta estava fixado em 0,184m, estabelecendo-se os limites de contagem da potência P da fonte de energia entre 4,55W e 27,6W, com uma segmentação deste intervalo de 30 pontos, e sabendo-se que as medições devem ser ao longo do comprimento da aleta desde da sua posição inicial, junto ao aquecedor, onde x vale *0*, até o comprimento total desta aleta, o qual vale L, e com uma segmentação também estabelecida em 30 pontos para este intervalo, pode-se propor a seguinte seqüência de ações para a criação da visualização:

- Fixar o valor da temperatura ambiente em 29,5°C
- Fixar o comprimento L da aleta em 0,184m
- Variar o valor da potência desde 4,55W até 27,6W, com 30 pontos calculados
- Em função do valor da potência calcular o valor da temperatura inicial da aleta
- Em função do valor da temperatura calcular o valor do fator m
- Variar o valor da distância ao longo da aleta desde *0* até L, com 30 pontos calculados
- Utilizar a equação *(25)* como filtro e fornecer valores de contorno para ela
- Calcular a temperatura de um ponto pela equação *(21)*, subordinando este cálculo ao filtro
- Visualizar a região filtrada

O filtro deve vir imediatamente antes da linha que calcula a variável principal, no caso a equação *(21)*. O filtro utilizado, na suposição proposta, pode criar intervalos de temperaturas que restrinjam a área de dados a ser visualizada através de um caminho equacionado. Pode-se completar a suposição proposta dando valores aos contornos da

equação *(25)* filtrante. Como esta equação depende de valores de temperaturas, pode-se supor, por exemplo, valores não menores que 45°C e não maiores que 60°C. No Capítulo 6, item 6.3.2, é proposto um *MathScript*, o qual representa a seqüência de ações descrita anteriormente e utiliza os contornos sugeridos neste parágrafo à equação *(25)*.

O programa MFSV pode prodir uma visualização de toda a superfície, que representa as distribuições de temperaturas, conforme a Figura 8. Entretanto, o resultado da utilização do MFSV, após a execução do *LIPScript* convertido a partir do *MathScript* referido anteriormente, é a visualização de uma faixa, segmento de uma superfície maior, no espaço tridimensional como apresentada na Figura 9.

FIGURA 8: VISUALIZAÇÃO DE TODA A SUPERFÍCIE QUE REPRESENTA TODAS AS DISTRIBUIÇÕES DE TEMPERATURAS

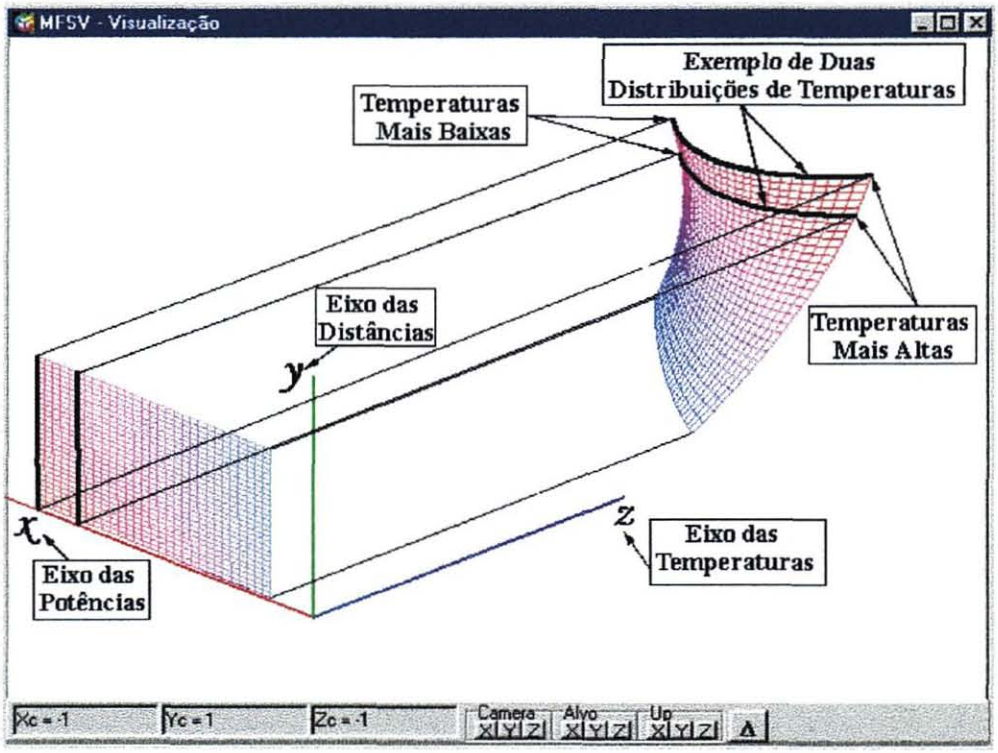
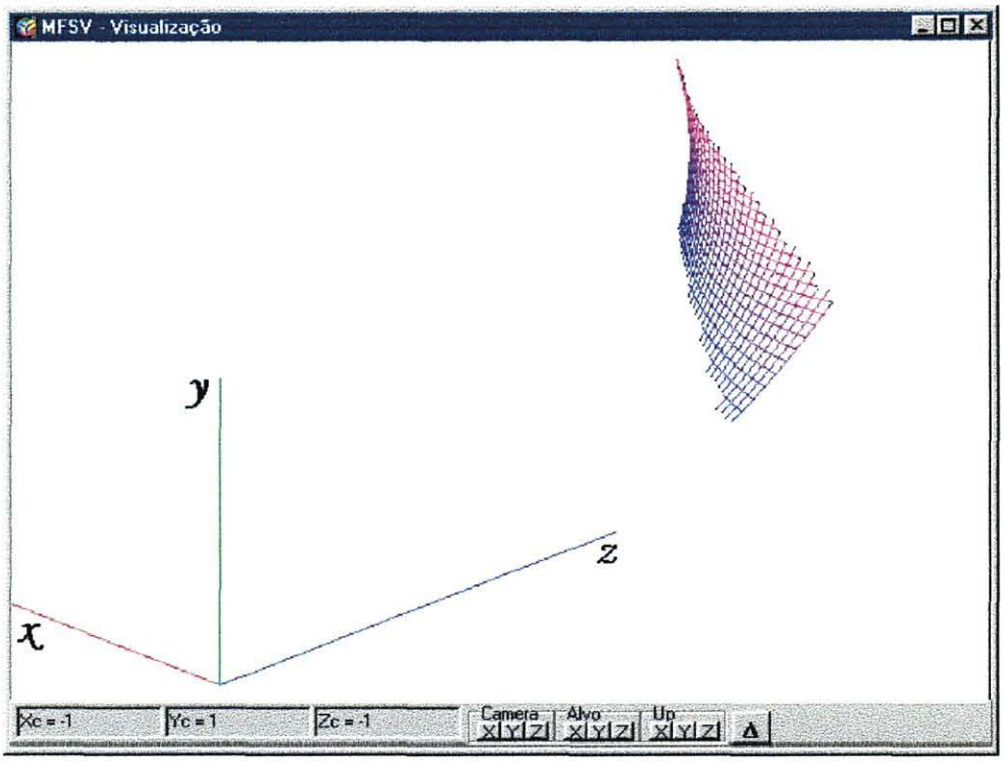


FIGURA 9: VISUALIZAÇÃO DA REGIÃO FILTRADA DA SUPERFÍCIE QUE CONTÉM AS DISTRIBUIÇÕES DE TEMPERATURAS



As Figuras 8 e 9 mostram a malha que representa as distribuições de temperaturas, sendo que a Figura 8 apresenta uma faixa filtrada da superfície pela equação e contornos mencionados anteriormente. Nestas figuras, o eixo x representa os valores de potência, o eixo y os valores de distância ao longo da aleta e o eixo z as temperaturas, sendo que as mais altas são as mais vermelhas, no espectro de cores, e as mais baixas são mais azuis. A visualização foi gerada em uma vista isométrica. Para cada valor de potência do eixo x existe uma linha de distribuição de temperaturas pelo eixo y das distâncias. A linha que mais se aproxima de um segmento de reta, na figura 9, tem sua projeção no plano xy exatamente sobre o eixo x e revela o local onde estão as temperaturas iniciais, as quais são as mais altas nas distribuições. Estas temperaturas iniciais aumentam para valores maiores de potência ao longo do eixo x. Percorrendo-se o eixo y, sobre uma mesma linha de distribuição de temperaturas, as temperaturas vão decrescendo. Em qualquer uma das duas direções, ao longo de x ou de y,

são percebidas curvas exponenciais para as temperaturas, sendo que para valores altos de potência os valores de temperatura aumentam e para distâncias maiores os valores de temperatura diminuem.

Todas estas descrições de posicionamento devem ser consideradas para as demais figuras referentes às distribuições de temperaturas apresentadas na seqüência deste trabalho. Para obter a visualização, mostrada na Figura 9, são estabelecidos os seguintes valores de opção no programa MFSV:

- Vista isométrica orientada no sentido NW->SE (de Noroeste para Sudeste).
- Modelagem *Wireframe* com uso de primitivas linhas
- *Zoom*[55] de ampliação com o fator de 3,82 vezes o tamanho atual da vista.
- *Pan*[56] horizontal de -15 unidades e vertical de -25 unidades
- *Scale*[57] em *y* de 100 vezes o tamanho atual

Como os valores das distâncias são muito baixos, quando comparados com os valores das potências e das temperaturas, há a necessidade de mudança na proporção da escala unitária representada no eixo *y* em relação aos eixos *x* e *z*. Esta é a razão do escalonamento em 100 vezes citado para o eixo *y*.

---

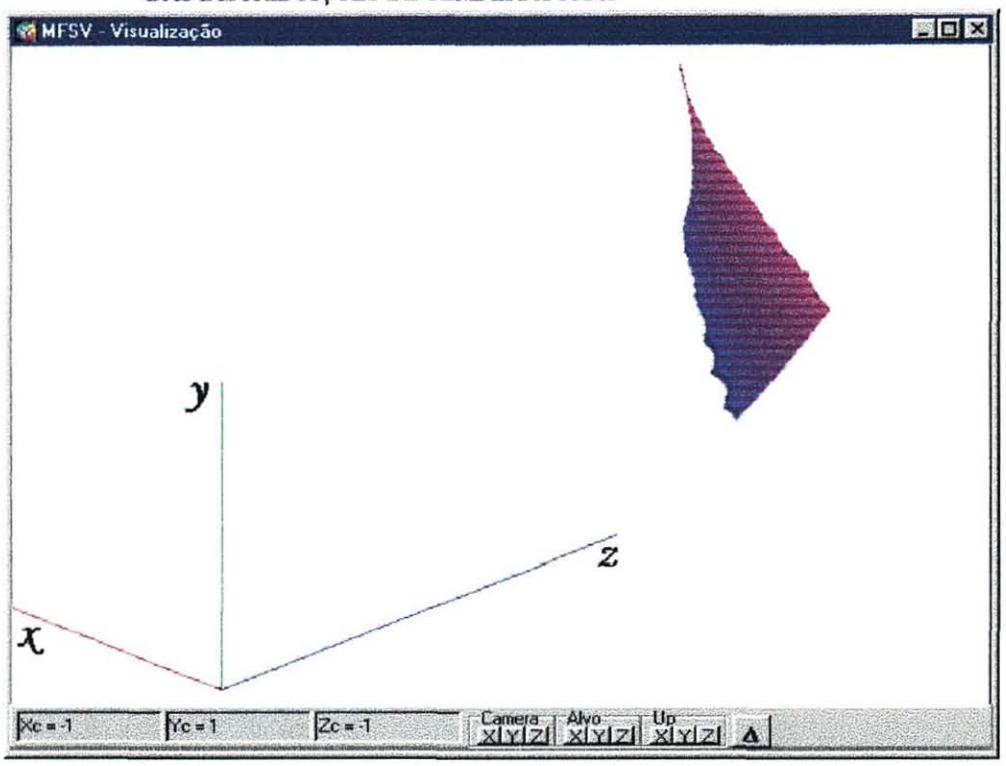
[55] *Zoom* é a operação que faz a aproximação ou o afastamento da vista atual dos objetos. Os objetos não sofrem alterações de valores em relação ao sistema de coordenadas.

[56] *Pan* é a operação que promove deslocamentos laterais da vista atual dos objetos. Semelhante ao *Zoom* os objetos não sofrem alterações de coordenadas.

[57] *Scale* é a operação que altera o tamanho unitário de um eixo em relação aos demais. Esta operação não muda necessariamnte os valores de coordenadas dos objetos, mas deforma a sua aparência em relação às suas proporções iniciais.

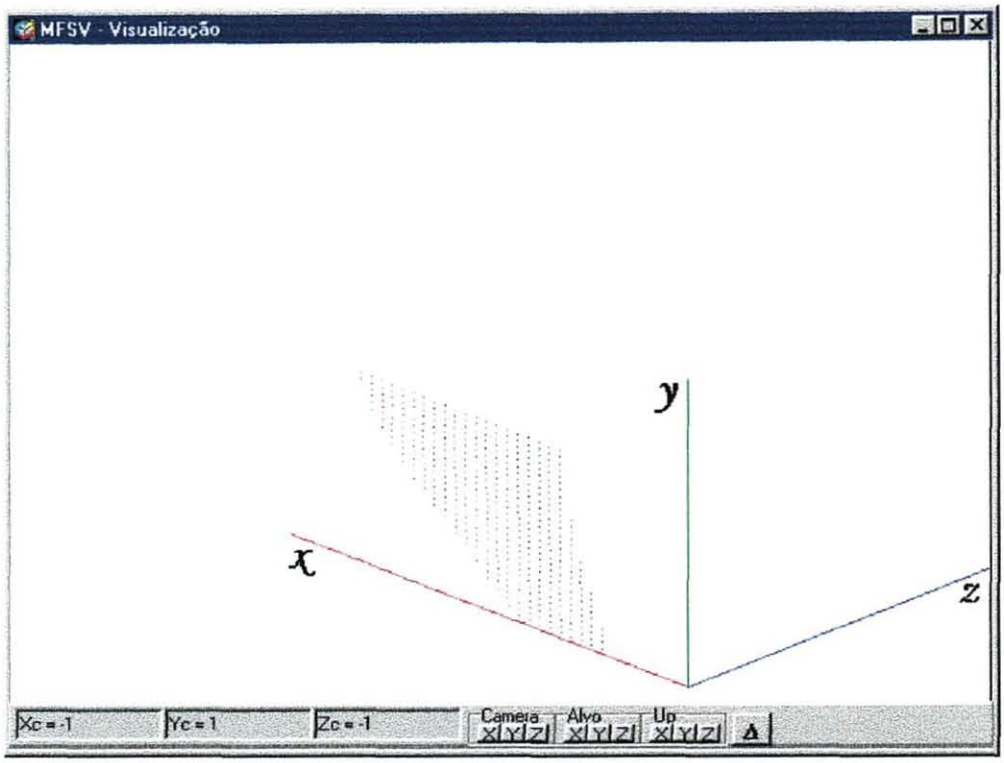
A Figura 10 mostra a mesma configuração da figura gerada para a Figura 9, porém utilizando faces para a visualização.

FIGURA 10: VISUALIZAÇÃO COM REPRESENTAÇÃO USANDO FACES DA REGIÃO FILTRADA DAS DISTRIBUIÇÕES DE TEMPERATURAS



A Figura 11 mostra a mesma faixa de temperaturas, porém com a visualização reapresentada com pontos e os valores de *z* projetados sobre o plano *xy*. Esta figura mostra a projeção no plano *xy* da faixa de valores para as temperaturas. Poder-se-ia, também, fazer o mesmo tipo de projeção para representações com linhas e faces. Todas as figuras apresentam um serrilhado na região de corte da superfície, porque a malha utilizada possui, relativamente, poucos pontos, 30 por 30 pontos. Aumentando-se a quantidade de pontos, nesta malha, melhora-se a definição destes contornos geométricos. Nas figuras mencionadas, até o momento, é vista apenas a faixa de temperaturas dentro das condições estabelecidas pela equação e contornos filtrantes da superfície de temperaturas.

FIGURA 11: VISUALIZAÇÃO COM REPRESENTAÇÃO USANDO PONTOS DA REGIÃO FILTRADA



Para obter a vista da Figura 11 no programa MFSV deve-se fazer as seguintes ações a partir da posição utilizada para as Figuras 9 e 10:

- *Pan* horizontal de +25 unidades
- *Scale*[58] em *z* de 0.01 vezes o tamanho atual

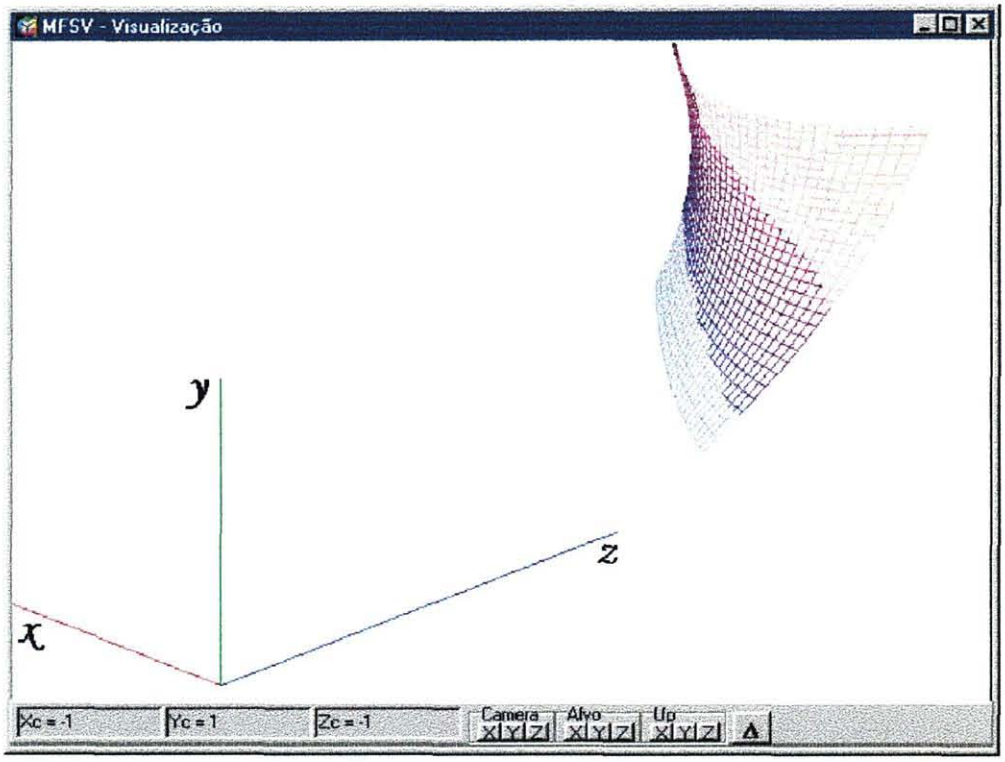
A Figura 12 mostra, novamente, a faixa de temperaturas, mas apresenta o restante dos valores calculados para a superfície de temperaturas. Nesta figura os valores que não pertencem a faixa filtrada estão em menor destaque em relação aos que pertencem, porque o MFSV permite a utilização do canal alfa para o ajuste de transparência dos *pixels*[59] desta

[58] De fato os valores de *z* continuam os mesmos não sendo 0, porém são suficientemente escalonados para se obter uma visualização da projeção no plano *xy* da região filtrada da superfície. Isto garante que o processo de cálculo das cores para os pontos na malha continue válido.

[59] A palavra *Pixel* vem da contração das palavras *Picture* e *Element* e representa a menor informação gráfica com significado.

região. Os valores de alfa são normalizados no intervalo de 0 a 1, sendo 0 o valor de opacidade mínima (0%) ou de transparência máxima (100%) e 1 o valor de opacidade máxima (100%) ou de transparência mínima (0%). Isto permite mostrar todos os pontos calculados na mesma intensidade, se o valor de alfa for 1, ou de opacidade máxima. Nas figuras anteriores o valor de alfa era 0, ou de transparência máxima. Na Figura 12 o valor 0,5 foi estabelecido para o canal alfa, onde os pixels são 50% transparentes ou, o que possui o mesmo significado, 50% opacos.

FIGURA 12: VISUALIZAÇÃO COM LINHAS DA REGIÃO FILTRADA, ONDE O ALFA VALE 1 (100%), E DEMAIS LINHAS DA SUPERFÍCIE CALCULADA, ONDE O ALFA VALE 0,5 (50%)



A Figura 13 tem os mesmos valores para alfa utilizados na geração da Figura 12, mas no processo de *rendering* utilizou-se a representação com faces em uma vista ortogonal, a qual é uma vista de topo do plano *xy*. Esta vista ortogonal, entre outras, pode ser obtida pela subopção *XY* da opção chamada Plano, dentro do menu Projeção, existente no programa MFSV.

FIGURA 13: VISUALIZAÇÃO COM FACES DA REGIÃO FILTRADA, ONDE O ALFA VALE 1 (100%), E DEMAIS FACES DA SUPERFÍCIE CALCULADA, ONDE O ALFA VALE 0,5 (50%)

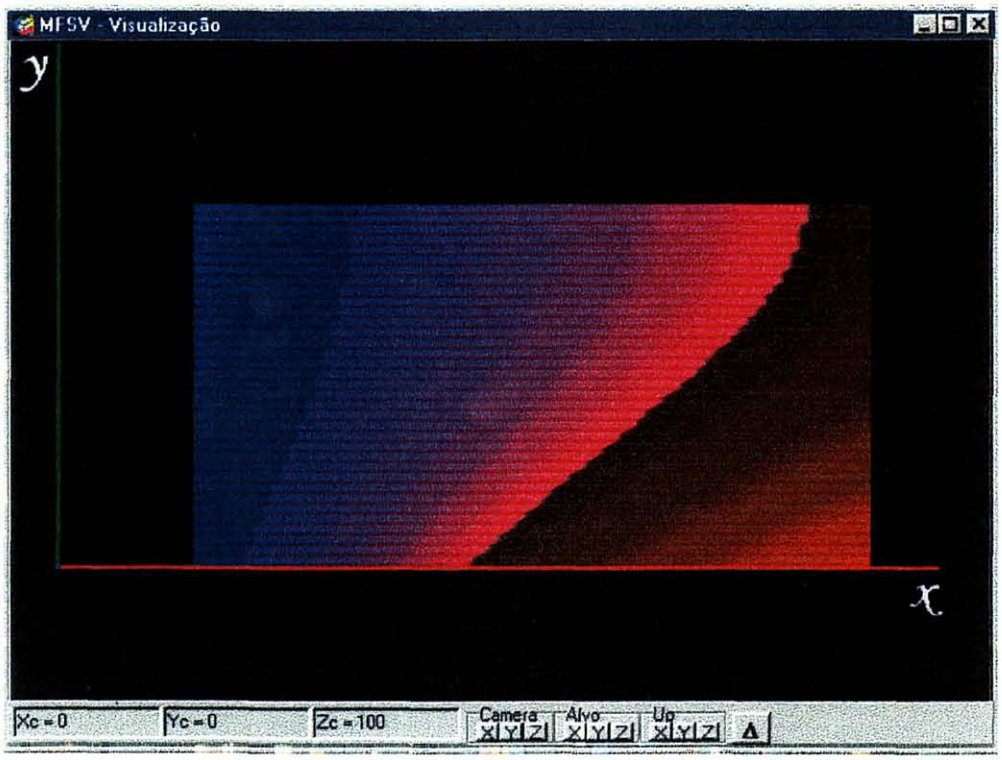
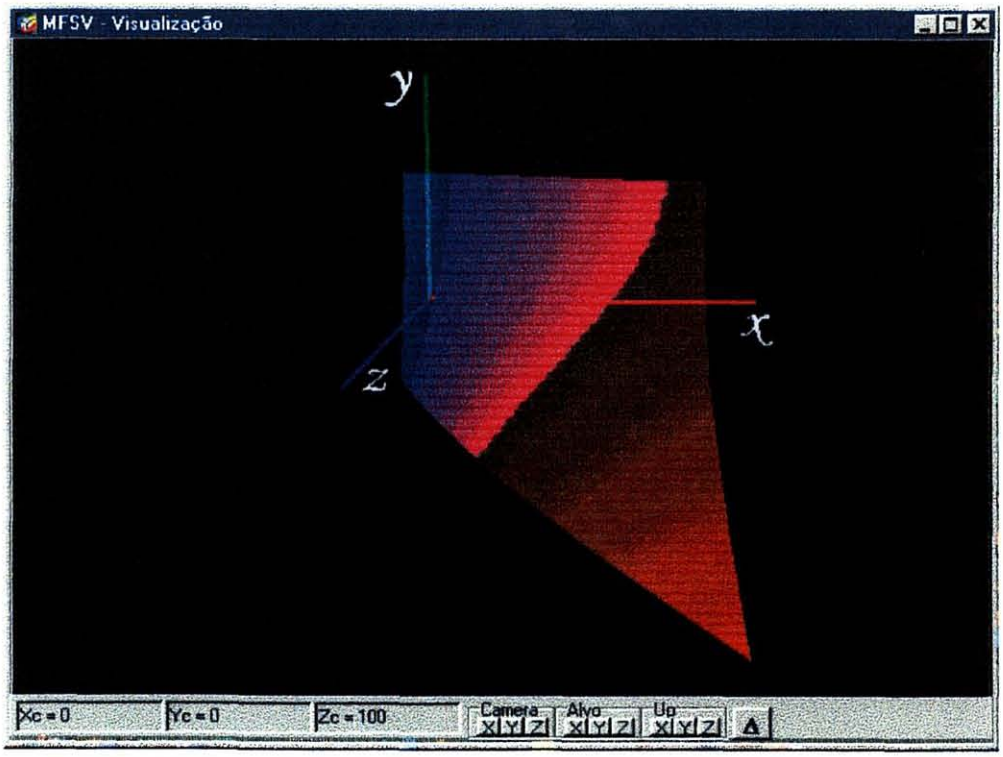


FIGURA 14: VISUALIZAÇÃO EM PERSPECTIVA



A Figura 14 apresenta uma vista em perspectiva das faces da região filtrada e demais faces da superfície com os mesmos valores para alfa utilizados para a geração da visualização mostrada na Figura 13. O programa MFSV possui a opção de vistas em perspectiva a partir do menu Projeção. Os ajustes necessários à obtenção da Figura 14 são os seguintes:

- Modelagem com faces
- Posição da Câmera: (Xc = 20, Yc = 20, Zc = 110)
- Posição do Alvo: (Xa = 10, Ya = -10, Za = 0)
- Plano de Profundidade em relação ao volume de visualização com valor 0

Para as Figuras 13 e 14 foi utilizada uma malha de 80 por 80 pontos. Esta malha maior permite uma melhor definição das bordas da região filtrada.

Todas as figuras apresentadas, neste capítulo, mostraram alguns dos recursos que o MFSV fornece. Os recursos foram utilizados para demonstrar o uso de filtros a partir de formulações matemáticas e como a região dos dados filtrada, de interesse do pesquisador, pode ser visualizada com o uso destes recursos.

6 ASPECTOS DE PROJETO E IMPLEMENTAÇÃO

A análise para o projeto e desenvolvimento do programa MFSV teve três abordagens importantes e vários aspectos relacionados a elas. As três abordagens foram:

- Suporte utilizado na computação gráfica
- Ambiente de Desenvolvimento Integrado (ADI) para a programação
- Projeto e implementação do programa visualizador MFSV

6.1 SUPORTE UTILIZADO NA COMPUTAÇÃO GRÁFICA

O suporte utilizado na computação gráfica do MFSV foi realizado pelo uso dos recursos da biblioteca gráfica OpenGL, como explicado no capítulo 3. Entretanto, o ambiente a ser escolhido para a realização do desenvolvimento do aplicativo MFSV deveria estar integrado a este suporte como ilustrado no diagrama da Figura 15, e este foi o ponto chave que determinou a escolha pelo *C++Builder*[60]. O estudo de uma ferramenta de programação visual utilizando o OpenGL é proposto por JACOBS[61] (1999).

FIGURA 15: DIAGRAMA DA INTEGRAÇÃO ENTRE O AMBIENTE DE DESENVOLVIMENTO E A BIBLIOTECA DE ROTINAS GRÁFICAS

[60] O compilador *C++ Builder* é um produto da divisão *Borland* da empresa *Inprise Inc.* Recentemente a *Borland* propôs o projeto *Kelix* para o *Linux*. Desta forma, os programas desenvolvidos em ferramentas como o *C++Builder* poderão migrar entre plataformas diferentes e, portanto, aproveitar os recursos de sistema que outros sistemas de computação dispõem.

[61] Outro produto que possui a mesma ADI e usa a mesma biblioteca visual de componentes que o *C++Builder* é o *Delphi*, também produto da *Borland*. A única diferença entre o *C++Builder* e o *Delphi* é a linguagem de programação utilizada: O *Delphi* utiliza a linguagem Pascal orientada a objetos. Portanto, o desenvolvimento em qualquer um dos dois ambientes é similar.

6.2 AMBIENTE DE DESENVOLVIMENTO INTEGRADO PARA A PROGRAMAÇÃO

O ambiente de programação deveria atender certas caraterísticas, que foram avaliadas como importantes segundo os atuais paradigmas da área de programação, e estas características incluem:

- Programação orientada a objetos e baseada em eventos (COHOON, 1999).
- Programação visual em ambientes de Desenvolvimento Rápido de Aplicações (conhecidos pela sigla *RAD*[62], como: o *Visual Basic*, o *Delphi*, entre outros).
- Implementação portável para diferentes plataformas.
- Fácil inclusão de componentes visuais desenvolvidos por terceiros.

Estas características foram encontradas no ADI *C++Builder*. Este ADI possui uma *interface* de uso simples e a facilidade de inclusão de novos componentes, neste ambiente, viabilizou a busca por um componente visual para o OpenGL. Embora o *C++Builder* já possua implementadas as funções para o uso do OpenGL, faltava a facilidade de criação e controle das janelas destinadas à visualização, uma vez que o OpenGL não tem funções específicas neste sentido. A criação de um componente visual, para este propósito, foi descartada pelo fato de ser fácil a aquisição de componentes prontos criados por terceiros e disponíveis para inclusão no ambiente do *C++Builder*. Foi encontrado um componente chamado *TOpenGL*[63], o qual, após instalado no ambiente da ferramenta RAD, possibilitou um desenvolvimento mais rápido dos recursos necessários ao programa visualizador MFSV.

---

[62] Nestes ambientes ficam mais evidentes, ainda, as virtudes da filosofia da programação visual *DOC-VIEW*, a qual sugere a separação clara entre a programação da interface para o usuário e a programação específica do propósito do aplicativo.

[63] O componente *TOpenGL*, da empresa inglesa *Hellix*, pode ser adquirido pela Internet a partir do endereço: http://www.hellix.com/

O Ambiente de Desenvolvimento Integrado (ADI) do *C++Builder* está constituído pelas seguintes principais áreas:

- *Barra de menus*: para a seleção de opções gerais do funcionamento do ADI.
- *Barra Ferramentas do Sistema*: possui ícones para as principais tarefas do ADI.
- *Paleta de componentes*: permite a escolha de componentes[64] para o aplicativo.
- *Object Inspector*: permite a atribuição de valores a propriedades de objetos ou a escolha de eventos a ser programados para estes componentes.
- *Formulário*: o ADI fornece de início um formulário, o qual é uma das janela do aplicativo durante sua execução. Outros formulários podem ser incluídos no aplicativo a ser desenvolvido.
- *Área de codificação*: permite a escrita de códigos em linguagem C ou C++.

6.3 PROJETO E IMPLEMENTAÇÃO DO PROGRAMA VISUALIZADOR MFSV

Um projeto de um sistema visualizador e conseqüente implementação do programa não é tarefa muito simples, exige o estudo de muitos aspectos sobre programação científica, como os propostos por ZACHARY (1996), e aspectos sobre a arquitetura e ambiente de visualizadores, como em KOCHHAR et. alli (1991).

6.3.1 Projeto do programa visualizador MFSV.

O projeto deve ser preciso quanto à funcionalidade do programa visualizador, quanto ao nível de interatividade que este programa deve ter com o usuário e quanto à viabilidade na condução do processo de implementação.

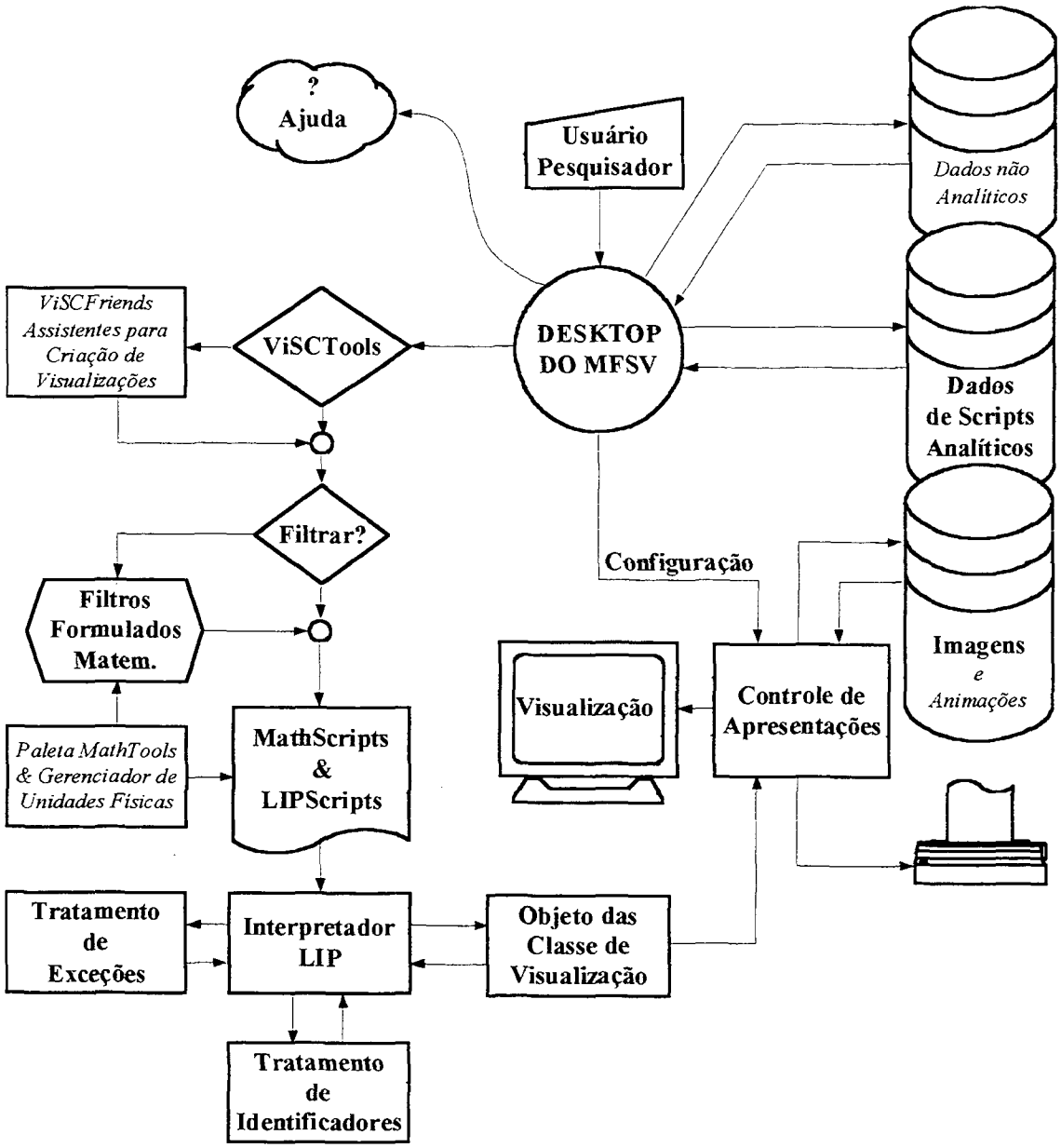
---

[64] Estes componentes, que são objetos, pertencem a um *framework* chamado *Visual Component Library (VCL)*, o qual é um conjunto de ferramentas prontas para uso na programação de aplicativos. Os componentes podem ser visuais ou não-visuais.

6.3.1.1 A funcionalidade do programa MFSV representada no projeto.

A funcionalidade do MFSV pode ser percebida pelas linhas de conexão apresentadas no diagrama da Figura 16, o qual mostra os diferentes módulos pretendidos para o programa ligados pelas referidas linhas de fluxo.

FIGURA 16: DIAGRAMA DO PROJETO CRIADO PARA O DESENVOLVIMENTO DO VISUALIZADOR MFSV



A funcionalidade prevista pelo projeto foi importante para os objetivos da implementação de caraterísticas específicas que este programa deve ter, principalmente as relacionadas aos filtros físico-matemáticos.

Os textos em itálico na Figura 16 são aspectos que não foram implementados no programa MFSV, até o momento. Deve-se notar que outras opções internas aos módulos, portanto, não visíveis na Figura 16, não chegaram a ser implementadas para o *desktop*[65] do MFSV. Porém, todas estas opções não implementadas não trazem prejuízo algum aos propósitos desta dissertação. Algumas destas opções são comentadas no capítulo 7 como trabalhos futuros.

O módulo do diagrama da Figura 16 indicado por Controle de Apresentações inclui as seguintes configurações relacionadas à visualização:

- Mover, rotacionar e escalonar objetos e eixos.
- Tipos de luzes e modificações das propriedades das fontes luminosas
- Tipos de *rendering*: *Wireframe*, *flat* ou *smooth*.
- Ajuste do canal alfa
- *Antialiasing*, *depth cuing* e *fog*
- Tipo da primitivas utilizada na representação de malhas: Ponto, linha ou face.
- Cores de desenho, de fundo e cores para representação de gradientes (*dégradé*).
- Controle de exibição do glifo que representa os eixos cartesianos no espaço 3D.
- Tipos de vista: Ortográfica plana ou isométrica e em perspectiva.
- Valores dos parâmetros do volume de visualização
- *Zoom*, *pan* e posição da câmera, alvo e vetor *up*[66].
- Resolução de saída para a janela de visualização.
- Controle de imagem e animações.

---

[65] O termo *desktop* é uma metáfora utilizada para interfaces gráficas que sugere a correspondência entre o gerenciamento de ícones, opções de menus e janelas na tela e o gerenciamento de objetos reais na mesa de trabalho de uma pessoa.

[66] O ajuste do vetor chamado *up* indica, tomando-se como a origem do sistema cartesiano as coordenadas da câmera, o giro que esta possui em relação aos eixos *x*, *y* e *z*. Normalmente, uma câmera possui as coordenadas (0 ,1 ,0) para o vetor *up*, ou também chamado de versor, as quais indicam uma câmera sem rotações em relação a horizontal e a vertical.

Umas das decisões de projeto tomadas é que o MFSV deveria sempre usar o caminho dos *scripts* para geração da visualização, mesmo para dados não analíticos. A razão para esta decisão foi o melhor controle dos dados por uma via única. Tal via sempre permitirá a inclusão de novas características aos dados e a inclusão de filtros formulados matematicamente. Obviamente, o programa deve ser muito interativo e muitas vezes o usuário não precisará escrever na área dos *scripts* e pode desconsiderar a sua presença. Entretanto, mesmo que o usuário precise alterar código, quase sempre estas modificações poderão ser efetuadas na linguagem matemática pelo uso dos *MathScripts*.

A utilização do programa deve ser fácil e, baseado na observação do diagrama da Figura 16, consiste nos seguintes passos[67]:

- Escolher uma ferramenta para o fenômeno e/ou entidade.
- Escolher uma ferramenta de visualização e usar, ou não, os *ViSCFriends*, os quais assistem o usuário na criação de visualizações.
- Definir um ou mais filtros físico-matemáticos, utilizando ou não a paleta de recursos matemáticos chamada de *MathTools*.
- Escrever um *script* matemático que descreva o fenômeno.
- Escolher o comando que converte o *script* matemático para LIP.
- Alterar ou não o código existente na área de *LIPScripts*.
- Executar o código existente na área do *LIPScripts*.
- Realizar o controle das apresentações como explicado anteriormente.
- Salvar as informações obtidas nas visualizações.

A funcionalidade das características nomeadas como *ViSCFriends* e *MathTools* estão descritos no capítulo 7 como propostas de futuros trabalhos.

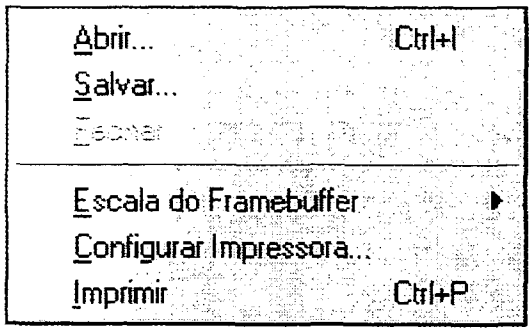
---

[67] É necessário observar que os dados podem vir de arquivos de dados previamente gerados e, portanto, muitas das opções aqui relacionadas podem não ser realizadas, aceitando-se os valores atuais estabelecidos no próprio arquivo. Na situação de arquivos de *scripts*, os quais são considerados dados analíticos, o usuário poderia ir direto à opção de conversão e execução do *script* e gerar a visualização ou usar o ambiente do MFSV somente para a apresentação de imagens e animações.

6.3.1.2 A interatividade do usuário com o programa MFSV.

O projeto do programa MFSV, no que se refere a interatividade com o usuário, previu a utilização de vários atalhos para a escolha de opções do ambiente como: Acionamento de teclas combinadas, uso dos botões do mouse sobre ícones de barras de ferramentas e regiões de janelas. Alguns exemplos desta característica interativa são: A solicitação de carga de um arquivo *script*, a qual pode ser realizada pelo acionamento do botão esquerdo do *mouse* sobre a barra de título das áreas de edição do próprio *script* e o acionamento do botão direito do *mouse* sobre a janela de visualização, o qual chama um menu estilo *pop-up*[68] para controle de gravação, abertura e impressão de imagens. Este menu está representado na Figura 17.

FIGURA 17: MENU POP-UP PARA CONTROLE DE GRAVAÇÃO, ABERTURA E IMPRESSÃO DE IMAGENS



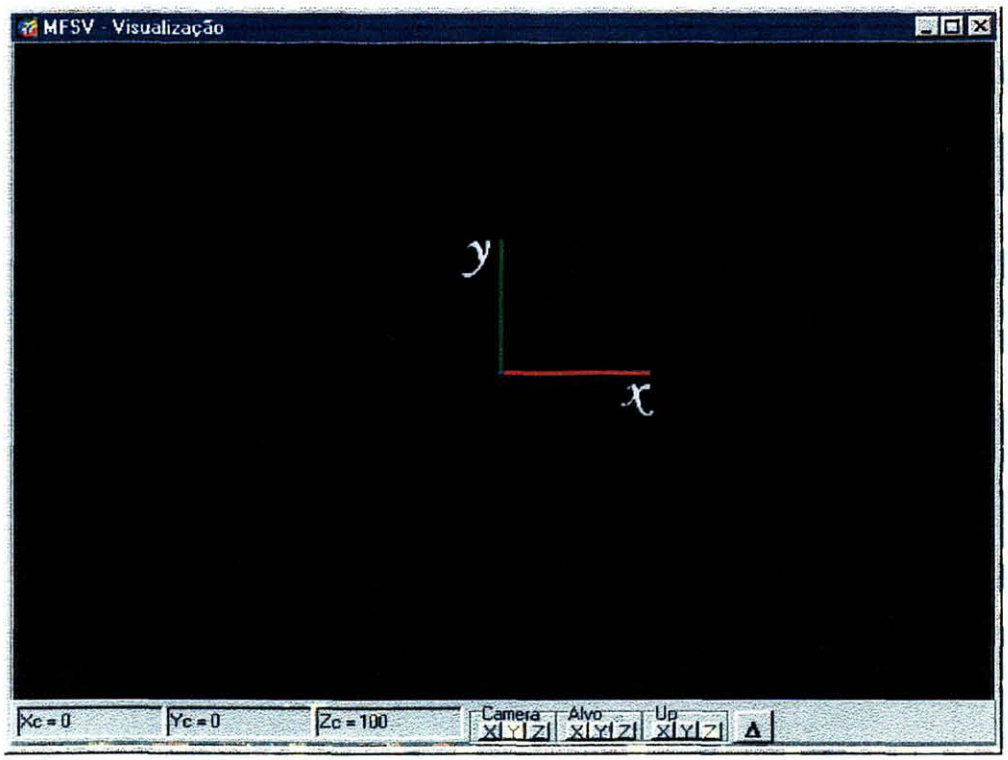
Um dos objetivos relacionados à janela de visualização, apresentada pela Figura 18, foi fornecer um grau satisfatório de interatividade com o usuário. Esta interatividade é notada pelas descrições das características presentes nesta janela.

Outras características da janela de visualização, além das opções relacionadas ao menu *pop up*, as quais foram resumidas no início deste item, serão apresentadas.

---

[68] Um menu *pop-up* é um menu que surge no ponto onde for acionado um botão do *mouse*.

FIGURA 18: JANELA DE VISUALIZAÇÃO DO PROGRAMA MFSV



A janela de visualização do MFSV foi dotada de um tipo peculiar de comportamento, o qual permite a ela ser, algumas vezes, uma janela filha da janela principal do programa, acompanhando a janela principal em seus movimentações e redimensionamentos, e, em outras vezes, ela se torna uma janela livre, desvinculada da principal. Este tipo de janela foi denominado, neste trabalho, como *janela adotada*.

Para tornar a janela de visualização uma janela livre, basta dar um duplo-clique no botão esquerdo do *mouse* sobre a sua barra de título. Outro duplo-clique na barra de título da janela de visualização, agora livre, maximiza a janela para ocupar toda a área da tela. Um clique sobre o botão de fechamento, o qual assemelha-se à letra "X", faz a janela retornar ao estado de janela filha. A janela de visualização estando na situação de filha é chamada, neste trabalho, de *Display Preview*. Estando tal janela como livre é chamada de *Display Final*.

Existente na base da janela de visualização, a chamada *barra de status* tem áreas de entrada de valores pelo teclado para coordenadas e pequenos botões acionáveis por cliques do *mouse*. Através das áreas rotuladas como *Xc*,*Yc* e *Zc*, as coordenadas da posição da câmera podem ser editadas pelo teclado, bastando dar um clique do botão esquerdo do mouse sobre as referidas áreas rotuladas. Estas áreas rotuladas podem ser alteradas para representarem coordenadas de outra característica da câmera. Por exemplo, pode-se clicar sobre um dos pequenos botões rotulados como *x*, *y* ou *z* existente abaixo da palavra *Alvo* e as áreas editáveis pelo teclado serão rotuladas, respectivamente, para *Xa*, *Ya* e *Za*, permitindo a alteração da posição do alvo para onde a câmera aponta. Pode-se, ainda, clicar sobre um dos botões abaixo da palavra *Up* e rotular as áreas editáveis como *Xup*, *Yup* e *Zup*, respectivamente.

Os pequenos botões, *x*, *y* e *z*, existentes abaixo das palavras *Câmera*, *Alvo* e *Up*, as quais representam propriedades de posição de câmera, têm outra característica que é dar um certo incremento ao valor de uma das coordenadas da propriedade de câmera que estiver com as coordenadas rotuladas nas áreas editáveis. Desta forma, clicando com o botão esquerdo do *mouse* sobre o botão *x*, por exemplo, incrementa-se o valor de *Xc*, ou *Xa*, ou *Xup*, dependendo de qual propriedade de câmera possui, no momento, os rótulos nas áreas editáveis. O valor do incremento é definido a partir do botão que possui o símbolo de delta, mas à direita na barra de *status*. O valor padrão para o incremento é 0,01 unidades para cada clique do *mouse*.

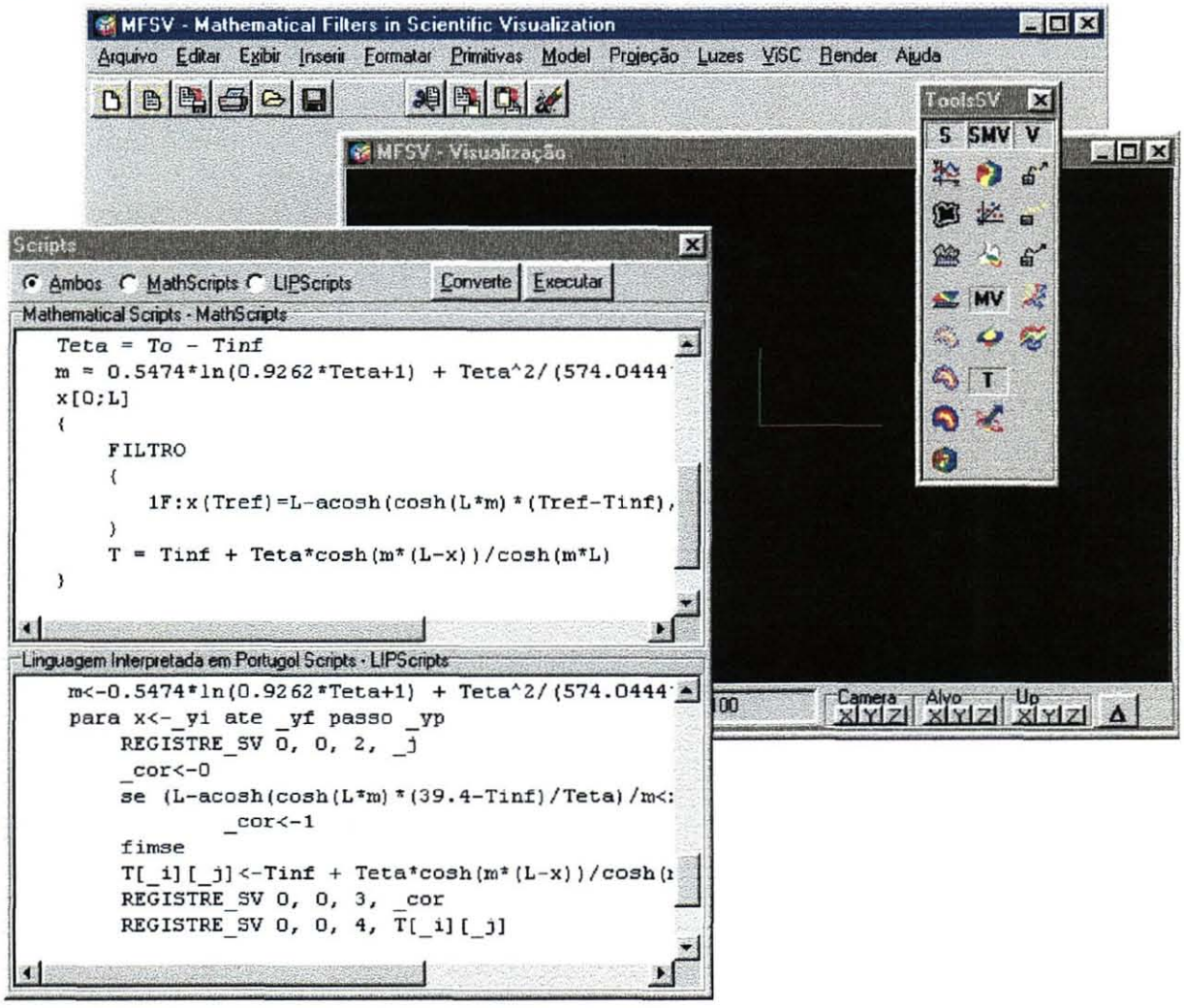
Quase todas as opções descritas podem ser escolhidas, também, a partir dos itens de menu do *desktop* do MFSV. Outras considerações poderiam ser feitas sobre a interação entre o usuário e o programa MFSV como: O fato de um simples clique do botão esquerdo do *mouse*, sobre uma área livre na janela principal do *desktop*, tornar visível a área de edição de *scripts*. Ou, a utilização interativa dos filtros físico-matemáticos para a fácil configuração pelo usuário-pesquisador do fenômeno e/ou entidade a ser visualizada. Entretanto, a atenção deve ser concentrada, doravante, nos aspectos de implementação que provará a viabilidade de todas as características previstas pelo projeto.

6.3.2 Implementação do programa visualizador MFSV

Os aspectos mais importantes de implementação do MFSV podem ser percebidos através das partes do projeto apresentado. Muitos outros aspectos poderiam ser abordados, porém somente os mais relevantes serão descritos neste trabalho.

A atual programação em ambientes gráficos facilita que o desenvolvimento dos aplicativos possa começar pelo preparo da interface visual de comunicação com o usuário. O MFSV tem um *desktop* como apresentado na Figura 19. A descrição das opções de menu, que foram implementadas, podem ser encontradas no Anexo 3.

FIGURA 19: *DESKTOP* DO MFSV



Nesta figura, pode-se ver em primeiro plano: A área de edição de *scripts* e a barra para escolha de ferramentas de visualização. A janela de visualização aparece como intermediária e, ao fundo, aparece a interface principal do programa MFSV.

A implementação dos módulos deste projeto seguiu uma convenção estabelecida, a qual define as expressões:

- *Unidade auxiliar*: constituída por dois arquivos, a saber: um arquivo de extensão .cpp, o qual possui métodos de classes, funções e declarações globais, e um arquivo de extensão .h, o qual possui identificadores interunidades e a descrição de classes e seus membros.
- *Unidade visual*: constituída pelos mesmos dois tipos de arquivos de uma unidade auxiliar, porém com um terceiro arquivo relacionando esta unidade a um formulário, o qual poderá ou não ser visível ao usuário durante a execução do programa.

Conforme tais definições as unidades auxiliares desenvolvidas para o MFSV, baseadas no projeto proposto, foram:

- *MFSV*: Chamada de projeto de compilação, esta unidade contém todas as referências às janelas e demais unidades do programa e faz a interface entre elas.
- *Interpretador*: Codificação responsável pela interpretação de códigos em linguagem LIP.
- *Arv_avl2*: Controla, em códigos escritos em LIP, a criação, armazenamento e apresentação de conteúdos de identificadores como variáveis simples e matrizes.
- *ClassesSV*: Possui todas as classes para objetos de visualização e suas respectivas rotinas transcritoras de textos em *MathScripts* para códigos em *LIPScripts*.
- *Exceções*: Utilizada pela unidade auxiliar `Interpretador`, esta unidade possui uma tabela com possíveis erros de execução de um programa escrito em LIP e estes são apresentados segundo um método da classe de `excecoes`.

As unidades visuais desenvolvidas para o MFSV são as seguintes:

- *MainMFSV*: Codificação da janela principal do *desktop* do programa, a qual, no código, foi chamada de `frmMFSV`[69].
- *Help*: Apresenta uma janela de ajuda, chamada de `frmSobre`, para o usuário.
- *Script*: Codificação de controle das áreas de edições de *MathScripts* e *LIPScripts*. Esta unidade visual está relacionada à janela chamada `frmScript`, a qual contém os *scripts* mencionados e dois botões: Um que executa o método de conversão, ou transcrição, do texto *MathScripts* para código na área *LIPScripts* e outro botão que executa o código em LIP.
- *ToolsSV*: Permite ao usuário a escolha de uma ferramenta de visualização a partir de uma janela chamada, no código, de `frmToolsSV`.
- *TelaSaidaVis*: Responsável pela apresentação das visualizações produzidas pelo programa. A janela associada a esta unidade visual é chamada de `frmVisualizacao`.
- *ResolucaoXY*: Esta unidade permite a definição da quantidade de pixels na horizontal e vertical da janela de visualização. A janela aliada a esta unidade é `frmResolucaoSaida`.
- *InLookAt*: Permite ao usuário a escolha dos parâmetros referentes às coordenadas no espaço para as posições da câmera, do alvo e do vetor *up* desta câmera. A janela associada a esta unidade é `frmInLookAt`.
- *BoxVisual*: Esta unidade solicita ao usuário os valores paramétricos limitadores do volume de visualização. A `frmBoxVisual` é a janela associada a esta unidade.
- *BoxPersp*: Solicita ao usuário os valores dos planos frontal e posterior do volume de visualização para as vistas em perspectiva cônica e solicita, também, o ângulo da lente para o ponto de observação. Tal ângulo é conhecido como *fovy*, na área da Computação Gráfica. O nome da janela associada a tal unidade é `frmBoxPersp`.

---

[69] Os nomes dados às janelas durante a codificação, recebeu o prefixo *frm* para enfatizar que se referem a formulários.

É importante observar que não faz parte dos objetivos deste trabalho apresentar a extensa codificação necessária à implementação do MFSV. Porém, alguns aspectos de implementação exigem a referência a estes códigos. Nestes casos serão citados os nomes dos elementos e unidades ou classes a que pertencem no código e, quando necessário, será indicada a observação de codificações presentes no Anexo 4 deste trabalho.

As partes mais complexas ou importantes na implementação do MFSV, segundo o projeto proposto no início deste capítulo, foram:

- O tratamento de conversão do texto na área *MathScript* para código em LIP, considerando a presença de filtros físico-matemáticos.
- O desenvolvimento da linguagem interpretada LIP.
- Estrutura de dados utilizada para controle dos identificadores utilizados em LIP.
- Aspectos das implementações gráficas usando OpenGL.
- Testar o visualizador criado para o estudo de caso usado neste trabalho e demonstrar a proposição enunciada no capítulo 1.

Além dos aspectos relacionados, a implementação deveria garantir uma suficiente interatividade do usuário com o *desktop*. Esta interatividade deveria objetivar o distanciamento do usuário em relação às programações de nível mais baixo, até mesmo de programações em linguagem LIP. A idéia principal foi manter, sempre que possível, o usuário escrevendo *scripts* matemáticos e de forma assistida pelo *desktop*.

6.3.2.1 Transcritor de *MathScripts* para *LIPScripts* e filtros físico-matemáticos.

O processo escolhido para a transcrição, ou conversão, de textos escritos em notação essencialmente matemática para código em linguagem LIP é um processo linear realizado por um método pertencente ao objeto da classe de visualização, o qual é criado

quando é escolhida a ferramenta de visualização. Cada classe de visualização tem um método transcritor específico segundo a ferramenta de visualização que representa.

As linhas lidas na área *MathScripts* são analisadas pelo método transcritor e geram uma ou mais linhas em LIP, as quais são colocadas na área *LIPScripts* segundo a lógica necessária para a visualização a ser obtida. Quando filtros formulados matematicamente, os quais são incluídos a partir dos *MathScripts*, são identificados, eles recebem um tratamento especial de transcrição, o qual subordina a(s) variável(is) fundamental(is) do problema objeto de estudo às restrições estabelecidas matematicamente no *MathScript*. A sintaxe matemática destes filtros e a forma como são transcritos dependem da formulação matemática destes filtros. A sintaxe matemática dos filtros aplicada à formulação matemática de funções e equações dependente de uma variável[70] é a seguinte:

```
FILTRO {
   <N°>F:<var(var_princ)> = <fórmula_filtrante> .: intervalos
   <N°>F:<var(var_princ)> = <fórmula_filtrante> .: intervalos
   ...
}
<var_fenômeno_a_ser_filtrado> = <equação_fenômeno_estudado>.
```

Onde <N°> é prefixo da letra "F", e esta composição cria um identificador para o filtro. A equação filtrante é regida por uma variável, indicada por <var>, a qual é dependente de outra variável, indicada por <var_princ>, que pertencente à fórmula filtrante, indicada como <fórmula_filtrante> na sintaxe do filtro. Esta variável, parâmetro, se refere aos possíveis valores definidos para os intervalos[71] matemáticos, os quais são contornos

[70] Até o momento, foi o único tipo de filtro criado para o MFSV com a finalidade de demonstração de tais recursos.

[71] Estes intervalos, matematicamente abertos ou fechados, estão após o símbolo de ".: ", o qual significa "donde" em liguagem matemática.

utilizados para a fórmula filtrante. Para o tipo de filtro apresentado a equação que sofrerá a interferência do filtro deverá vir imediatamente após a definição dos filtros. Na sintaxe geral, apresentada anteriormente, esta equação do fenômeno estudado, a ser visualizado, é associada à variável indicada por `<var_fenômeno_a_ser_filtrado>`.

O usuário escolhe uma ferramenta de visualização e, na área de edição *MathScripts*, são incluídas, automaticamente, linhas mínimas[72] necessárias para a criação da visualização. Obviamente, o usuário deverá completar os parâmetros destas linhas e as outras especificações necessárias, segundo o fenômeno escolhido. O usuário, na versão atual[73] do MFSV, é solicitado para a escrita da sintaxe dos filtros condicionantes. Entretanto, por se tratar de sintaxe similar às expressões matemáticas por ele empregadas, esta não traz dificuldades para o usuário.

As sintaxes de outras linhas de um texto em *MathScripts* são menos complexas e podem ser resumidas da seguinte forma:

- Atribuições utilizando o sinal de igual "=".
- Definição de faixas de variação que são transcritos como estruturas de repetição em LIP. Estas faixas são indicadas por uma variável escolhida pelo usuário, seguida de dois valores colocados entre parênteses "( )", em intervalos matematicamente abertos, e entre colchetes "[ ]", em intervalos matematicamente fechados. Os intervalos poderão, ainda, ser aberto à esquerda e fechado à direita "( ]" e vice-versa "[ )". As passagens matemáticas, subordinadas a este intervalo, devem vir entre chaves "{ }".

---

[72] No item 6.3.2.4 são apresentadas as linhas mínimas criadas para a ferramenta *Color Map*.

[73] Uma versão posterior poderá criar filtros a partir de quadros de diálogos que assistiriam o processo de criação e inclusão de filtros na área *MathScripts*.

- A escrita de equações matemáticas incluem todos os operadores aritméticos, incluindo o de potenciação, e muitas das funções científicas clássicas como: Seno, cosseno, logarítmicas entre outras. Apesar da escrita linear, como são inseridas na área de *MathScripts*, são muito familiares ao usuário pesquisador.
- Registro de dados calculados no objeto da classe visualizadora. Os dados calculados precisam ser registrados no objeto da classe visualizadora, porque a permanência de dados durante a execução de programas em LIP, oriundos de textos *MathScript*, é volátil, ou seja só existem enquanto o programa estiver em execução. O registro faz um "gancho" alocando os dados no objeto que tem as especificações da classe de visualização. Estes dados, posteriormente à execução do programa, serão utilizados para a manutenção da visualização em janela apropriada.

6.3.2.2 Desenvolvimento da linguagem LIP

O trabalho de desenvolvimento da Linguagem Interpretada em Portugol, ou LIP, foi baseado em um pequeno exemplo de interpretador recursivo descendente[74] de comandos e expressões chamado de *SMALLBASIC* (SCHILDT, 1988, p. 247). O exemplo era limitado, principalmente no aspecto relacionado ao tratamento de variáveis, porque permitia o uso de apenas 26 variáveis segundo as letras iniciais de nomes identificadores (de "*a*" a "*z*") e não possuía tratamento para a utilização de matrizes. Além, destes aspectos relacionados às estruturas de dados, o pequeno exemplo possui uma estrutura condicional, mas que não permite aninhamentos e não permite o bloco de comandos relacionados à cláusula chamada de `senão` (ou, também conhecida por, `else`). Outra deficiência era a ausência de funções científicas e operadores lógicos.

---

[74] Interpretador recursivo descendente ou *Descend Parsing*, como é referido muitas vezes na literatura.

Entretanto, este programa exemplo foi utilizado como base para o desenvolvimento de um interpretador mais completo, o qual foi chamado de LIP. Os comandos implementados para a linguagem LIP estão relacionados na Tabela 9.

TABELA 9: COMANDOS IMPLEMENTADOS NA LINGUAGEM LIP

COMANDO	DESCRIÇÃO
IMPRIMA	Imprime valores de expressões e mensagens na tela
LEIA	Recebe dados solicitados ao usuário
SE <expressão> ENTAO <bloco1 de comandos> [SENAO <bloco2 de comandos>] FIMSE	Realiza o condicionamento da execução de um bloco de comandos segundo a valor verdadeiro ou falso de uma expressão. Se o valor da expressão for verdadeira o bloco 1 será executado, caso contrário o bloco 2 (se existir a porção referente à cláusula senao). Os colchetes indicam o que é opcional no comando
VAPARA <N° de linha>	Realiza um salto incondicional para o início de uma linha numerada
PARA <v> <- <i>ATE<f> PASSO<p> <bloco de comandos> FIMPARA	Repete o bloco de comandos na faixa de variação que vai do valor indicado por <i> até o valor indicado por <f>. A razão de incremento ou decremento de um valor para outro, nesta contagem, é o valor dado para <p>. A variável <v> é que recebe valores sucessivos na variação.
SUBROTINA <N° de linha>	Executa uma subrotina por linha numerada e retorna a execução para o próximo comando após o comando SUBROTINA
RETORNE	Denota o ponto final, ou de retorno, de uma subrotina.
REGISTRE SV <lista de parâmetros>	Registra os dados na classe visualizadora segundo parâmetros
REAL <lista de variáveis>	Declara variáveis para uso no programa. Até o momento é o único tipo existente em LIP
FIM	Força a finalização da execução de um programa em LIP

Os operadores lógicos disponíveis na linguagem LIP estão na Tabela 10:

TABELA 10: OPERADORES LÓGICOS DA LINGUAGEM LIP

OPERADOR LÓGICO	DESCRIÇÃO
NÃO	Inverte o estado lógico de uma expressão, se verdadeiro passa para falso.
E	Exige que duas proposições associadas com E sejam verdadeiras, senão o resultado avaliado por este operador será falso.
OU	Exige que pelo menos uma das duas proposições associadas com OU seja verdadeira, senão o resultado avaliado por este operador será falso.

Os operadores relacionais são: <, <=, >, >=, = e <> que significa diferente.

Os operadores aritméticos são exatamente como os aplicados na escrita linear de expressões matemáticas. Porém, considerando o uso do sinal de circunflexo "^", presente entre dois operandos, para indicar a potência de um número elevado a outro, e o sinal de porcentagem "%", o qual representa a operação de resto entre dois números inteiros.

A atribuição de valores é realizada pelo operador de atribuição, cujo sinal é constituído por um sinal de menor seguido de um sinal de menos "<-". Este operador tem a mesma característica de atribuição múltipla como encontrada na linguagem C e apresentada pelos criadores desta linguagem (KERNIGHAN e RITCHIE, 1986, p. 31).

As funções científicas matemáticas disponíveis em LIP estão relacionadas na Tabela 11.

TABELA 11: FUNÇÕES CIENTÍFICAS MATEMÁTICAS DA LINGUAGEM LIP

FUNÇÃO	DESCRIÇÃO
`asenh() ou asinh()`	calcula o arco seno hiperbólico de um valor passado
`asen() ou asin()`	calcula o arco seno de um valor passado
`senh() ou sinh()`	calcula o seno hiperbólico de um ângulo passado
`sen() ou sin()`	calcula o seno de um ângulo passado
`acosh()`	calcula o arco cosseno hiperbólico de um valor passado
`acos()`	calcula o arco cosseno de um valor passado
`cosh()`	calcula o cosseno hiperbólico de um ângulo passado
`cos()`	calcula o cosseno de um ângulo passado
`atanh()`	calcula o arco tangente hiperbólico de um valor passado
`atan()`	calcula o arco tangente de um valor passado
`tanh()`	calcula a tangente hiperbólica de um ângulo passado
`tan()`	calcula a tangente de um ângulo passado
`sqrt()`	calcula a raiz quadrada de um valor passado
`abs()`	calcula o valor absoluto de um valor passado
`log()`	calcula o logaritmo na base 10 de um número passado
`ln()`	calcula o logaritmo na base e de um número passado
`exp()`	calcula a exponenciação do número *e (2.71828...)* elevado a um número passado

Um aspecto importante da unidade auxiliar chamada `Interpretador`, a qual possui uma classe, também, chamada `Interpretador`, cujos métodos que possui implementa todos os elementos descritos nas tabelas apresentadas e operadores citados. Tais níveis mostram a ordem de avaliação em expressões matemáticas. Estes níveis descrevem a ordem como as operações e funções ocorrem, a qual é a mesma realizada nos cálculos matemáticos tradicionais. Esta precedência pode ser alterada, em LIP, pelo uso de parênteses, também, como nos cálculos tradicionais. Os níveis de avaliação estão apresentados na Tabela 12.

TABELA 12: PRECEDÊNCIA DE CÁLCULO DOS MÉTODOS DA CLASSE `INTERPRETADOR`

MÉTODO DA CLASSE INTERPRETADOR	ORDEM DE AVALIAÇÃO	DESCRIÇÃO DA AVALIAÇÃO REALIZADA
`nivel1()`	11ª	processa uma atribuição
`nivel2()`	10ª	executa uma operação lógica E ou OU
`nivel3()`	9ª	processa uma operação relacional
`nivel4()`	8ª	opera adições e subtrações
`nivel5()`	7ª	opera multiplicações, divisões e restos
`nivel6()`	6ª	processa uma potência
`nivel7()`	5ª	operação lógica de negação NÃO
`nivel8()`	4ª	avalia um operador unário - ou +
`nivel9()`	3ª	avalia uma função matemática científica
`nivel10()`	2ª	calcula um índice para uma matriz
`nivel11()`	1ª	avalia uma subexpressão existente entre parênteses

A execução de um código em linguagem interpretada, ou compilada, começa pela análise léxica, a qual realiza, essencialmente, a extração das partes básicas de um texto-fonte chamadas de átomos, ou *tokens*, os quais são representações numéricas, de tamanho fixo para os seguintes elementos: Identificadores de variáveis, de constantes ou de nomes de subprogramas, palavras reservadas, números inteiros ou reais com ou sem sinal, cadeias de caracteres, sinais de pontuação e de operação, caracteres especiais, símbolos compostos por dois ou mais caracteres especiais, comentários, entre outros (JOSE NETO, 1987, p.118).

A interpretação, ou execução, de um programa em linguagem LIP é iniciada pela chamada do método `precompil()` pertencente à classe `Interpretador`. Este método extrai os átomos existentes no texto-fonte LIP e elimina delimitadores e comentários[75], reconhece as seqüências numéricas e identificadores. Entretanto, os identificadores necessitam de um tratamento especial e são, portanto, analisados em particular no próximo item deste capítulo.

O processo de transformação para átomos é executado uma vez e, se não existirem alterações no código fonte, o método `precompil()` não realizará estas tarefas novamente, passando à execução efetiva do programa constituído pelos átomos extraídos.

A verificação de erros no interpretador LIP é realizada pela unidade auxiliar `excecoes`[76]. Um erro pode ocorrer na escrita, ou sintaxe, de código em LIP ou na lógica, ou semântica, durante a execução de um programa. Quando um erro ocorre, um método da classe de `excecoes` é chamado e apresenta uma janela de mensagem ao usuário.

6.3.2.3 Estrutura de dados utilizada para controle dos identificadores utilizados em LIP

A necessidade de uma estrutura de dados que controlasse os identificadores surgiu em decorrência das deficiências apontadas para o *SMALLBASIC* citado no item anterior deste capítulo.

Um primeiro passo foi identificar os nomes de palavras reservadas em LIP e transformá-los em átomos, evitando a ambigüidade com nomes utilizados pelo usuário programador em LIP. Restando os identificadores criados pelo usuário, é necessário um tratamento que realize uma técnica de compressão e uma indexação para acesso rápido destes.

---

[75] Os comentários em LIP são chamados de cometários de linha e estas são iniciadas no código com o símbolo "#".

[76] O código fonte desta unidade encontra-se no Anexo 4.

JOSE NETO (1987, p. 118) afirma: "Identificadores também são cadeias de caracteres de comprimento variável, não podendo portanto ser utilizados diretamente como partes dos átomos pela dificuldade de manipulação que isto pode acarretar. Assim sendo, surge a necessidade de se efetuar uma compressão na sua representação, o que é feito em geral com o auxílio de uma *tabela de símbolos*[77]".

A tabela de símbolos proposta para o MFSV foi baseada em um tamanho fixo de identificador. O tamanho proposto foi de 4 caracteres, ou *bytes* na memória RAM. A idéia base foi considerar os 4 primeiros caracteres de um nome identificador, como um número inteiro na localização de memória em que se encontram. Este número é utilizado como um índice, ou chave de busca, para acessar a localização do identificador na estrutura de dados, a qual deve armazenar os valores alocados para estes identificadores. Uma estrutura de dados que permite acesso rápido é uma árvore binária de balanceamento dinâmico, como utilizada em VILLAS (1993, p. 110), chamada de *AVL*[78].

Houve a necessidade de estabelecer uma regra de boa formação para os nomes dos identificadores. Tal regra é formada pelas seguintes restrições:

- O primeiro caracter deve pertencer ao conjunto formado por:
  ('a'..'z', 'A'..'Z', '_'), que equivale a 53 possibilidades.
- O segundo, ou até o quarto, caracter deve pertencer ao conjunto:
  ('a'..'z', 'A'..'Z', '_', '0'..'9', ' '), que equivale a 63 possibilidades por caracter.

---

[77] Uma tabela de símbolos é uma tabela onde são armazenadas as cadeias de caracteres formadores dos nomes identificadores, tais como aparecem no texto-fonte. Em alguns casos os nomes são truncados para um tamanho fixo e em outros casos é utilizado um sistema de indexação para a localização destes identificadores na tabela. Tais índices podem ser utilizados como os átomos representantes deste identificadores.

[78] O termo *AVL* vem dos nomes dos dois matemáticos russos Adelson-Velskii e Landis em 1962, que propuseram este tipo de árvore pela primeira vez.

Considerando que um identificador escrito em letras minúsculas será considerado outro identificador, se pelo menos uma de suas letras for escrita em maiúscula. Isto caracteriza LIP, a exemplo da linguagem C, como sendo uma linguagem sensível ao caso. Deve-se observar, que os nomes identificadores que forem menores que quatro caracteres serão completados com caracter nulo, o qual não é o mesmo que '0' na tabela de caracteres dos computadores.

Para as implementações de identificadores que representam matrizes, e que usam pares de colchetes para encerrarem índices, a presença do primeiro colchete de abertura '[' determina o ponto onde finaliza o nome identificador. Se este colchete estiver além dos quatro primeiros caracteres, será considerado como estando após o quarto caracter. Se este colchete estiver aquém do quarto caráter, o nome do identificador será completado com nulos até a quarta posição. Desta forma o nome do identificador é tratado da mesma forma que identificadores não matriciais.

Por estas regras são obtidas as seguintes possibilidades, quanto ao número de identificadores (abreviado por *NI*), sendo ou não estes identificadores relacionados a matrizes:

$$NI = 53 + 53x63 + 53x63^2 + 53x63^3 = 13.466.240 \text{ identificadores.}$$

Em 1962 os russos ADELSON-VELSKII e LANDIS propuseram um teorema, o qual demonstra que a altura, h, máxima de uma árvore AVL, no pior caso, e segundo um número de nós M, será dada pela seguinte expressão[79]:

$$h < \log_{\phi}(M+2) \; - \; 2 \; + \; \log_{\phi}(\sqrt{5}), \qquad (27)$$

onde $$\phi = \left(1+\sqrt{5}\right)/2. \qquad (28)$$

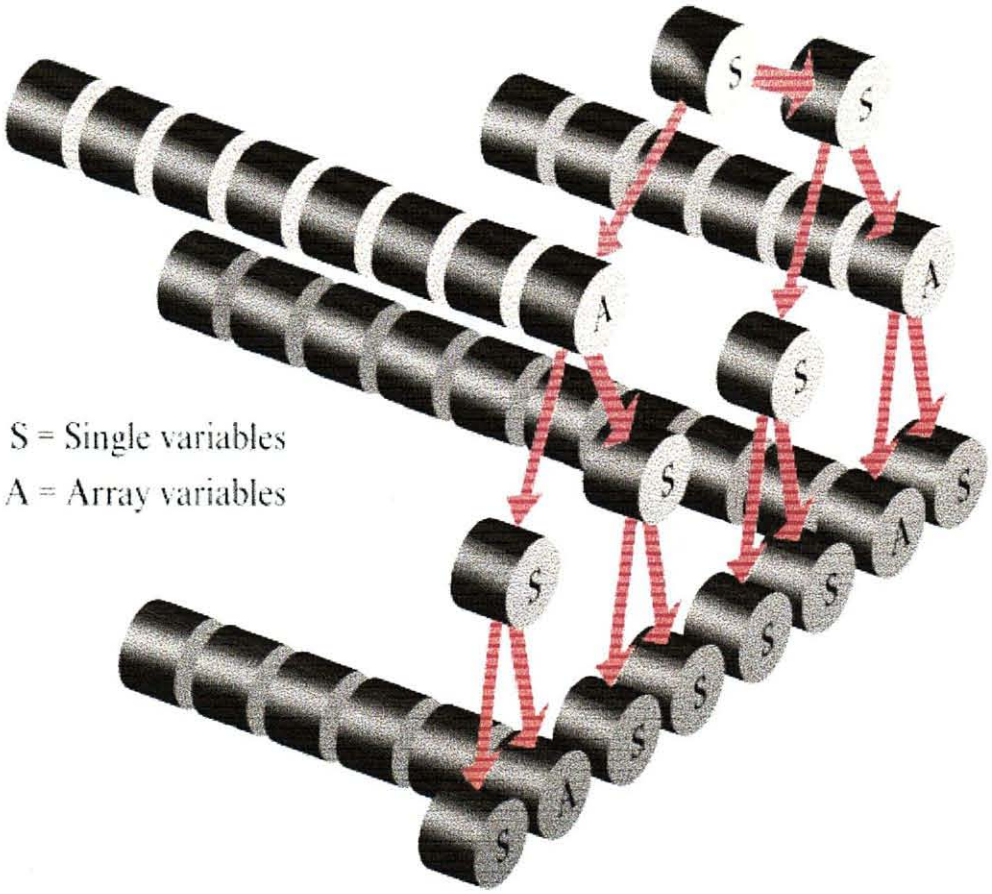
---

[79] Esta demonstração pode ser encontrada no endereço: http://www.eli.sdsu.edu/courses/fall95/cs660/notes/AVL/AVL.html da San Diego State University.

Assumindo que *M* seja igual a *NI* e considerando a equação determinada para o pior caso da árvore AVL, o qual determina o pior tempo de acesso ao nó mais distante da raiz. Nestas suposições o pior tempo de acesso ocorrerá para uma árvore com no máximo 33 nós de altura, e isto significa que com um máximo de 13 milhões de nomes identificadores criados serão necessárias apenas 33 comparações para achar o conteúdo de um identificador matricial ou não matricial. Percebe-se que a ocorrência de tal situação é praticamente impossível.

A Figura 20 mostra o esquema da árvore binária utilizada como gerenciadora dos nomes identificadores utilizados pela linguagem LIP. É importante observar que o tamanho da árvore não difere, se os nós pertencem ou não a identificadores de matrizes.

FIGURA 20: ÁRVORE GERENCIADORA DOS NOMES IDENTIFICADORES USADOS EM LIP



A Figura 20 mostra que as matrizes, em LIP, são todas tratadas como vetores, isto porque existe um cálculo de linearização de matrizes multidimensionais em um dos métodos da classe `Interpretador` antes do acesso aos controles da árvore. Desta forma, as matrizes escritas com mais de uma dimensão pelo programador LIP são linearizadas internamente antes de serem armazenadas na árvore.

O maior custo computacional de tempo ocorre nas declarações de variáveis realizadas em LIP, porque é necessária a alocação de espaço para a inclusão do identificador e seu valor na estrutura da árvore. Outro custo ocorre na remoção de um identificador e seu valor de uma árvore, porque exige a árvore seja rebalanceada. Contudo, estas situações ocorrem somente no início e fim da execução do programa em LIP, porque os identificadores são globais e são todos criados no início do programa. Porém, se uma versão nova previr a inclusão de identificadores locais em subrotinas e estas sofrerem rechamadas em processos recursivos, este custo computacional será sensível e prejudicará os tempos de execução dos programas, merecendo um outro tratamento especial.

Um dos mais importantes aspectos de implementação que deve ser considerado é a necessidade de registrar os dados calculados pelos programas em LIP. A árvore, que mantém todos os dados calculados, é uma estrutura que só existe durante a execução do programa LIP e como os controles de visualização não devem ficar subordinados ao tempo de execução de programas LIP é necessário que o referido registro de dados seja feito para os atributos membros do objeto da classe de visualização.

6.3.2.4 Aspectos das implementações gráficas usando OpenGL.

Os principais aspectos relacionados à implementação utilizando o OpenGL estão presentes na classe da unidade visual `TelaSaidaVis`, a qual possui diversos controles de apresentação sobre a janela criada pelo uso do componente OpenGL, o qual é responsável

pelo suporte à computação gráfica necessária à visualização. A programação foi completada para os seguintes principais métodos da classe `frmVisualizacao`[80]:

- `OpenGL1GLInit()`: Método que habilita a utilização do canal alfa para o *framebuffer* e configura o modo de *rendering* para *Smooth*.
- `OpenGL1GLPaint()`: Método que será executado sempre que a janela de visualização precisar ser redesenhada. Neste método é controlada a exibição do glifo de coordenadas cartesianas tridimensionais, entre outras tarefas. Este método ainda é responsável pela chamada ao método `DisplayObjetos()`, o qual apresenta os objetos na janela de visualização.
- `OpenGL1Resize()`: Este método é disparado toda vez que o usuário alterar o tamanho da janela de visualização e está encarregado de definir a *Viewport* segundo as dimensões atuais do componente OpenGL. Esta *Viewport* é a janela de pixels, utilizada como sendo o plano de projeção dos objetos existentes no espaço tridimensional. A alteração para o modo das matrizes que controlam a projeção é o próximo passo. A matriz identidade é carregada na pilha para que as próximas ações, ou transformações lineares, não sejam prejudicadas pela presença de valores antigos de matrizes. Este método tem, ainda, como objetivo configurar o tipo de projeção utilizado, se em projeção ortogonal ou em perspectiva.
- `OpenGL1Click()`: Programado para provocar, intencionalmente, o redesenho da janela de visualização, este evento faz o redesenho para garantir a atualização da informação visual presente nesta janela. Este evento é disparado quando o usuário clica o botão do *mouse*.

---

[80] Os métodos aqui citados estão no Anexo 4 deste trabalho

- `DisplayObjetos()`: Este método não é um evento, mas é acionado pelo evento `OpenGL1GLPaint()`. Este método responsável pela criação e apresentação da geometria dos objetos a partir dos dados de um objeto de uma classe de visualização e utiliza os recursos da programação a partir da biblioteca gráfica do OpenGL. Podemos resumir as ações deste método em:
  - Verificação da existência de uma instância de uma classe de visualização[81].
  - Inicialização de algumas variáveis com base nos valores de atributos desta instância de objeto.
  - Criação de objetos para visualização.

É evidente que a programação de eventos, em um ambiente de programação visual, e a utilização do componente OpenGL simplificaram extremamente a realização dos aspectos relacionados à computação gráfica. Entretanto, o desenvolvimento deste software está apenas no início e muitas outras considerações poderiam ser implementadas, ainda que não tenham sido explicados todos os aspectos do programa MFSV.

6.3.2.5 Aplicando o MFSV no estudo de caso da área de Transferência de Calor

Existindo o fenômeno a ser explorado e suas descrições matemáticas formuladas, este fenômeno poderá ser visualizado. O processo de desenvolvimento da visualização começa pela escolha da ferramenta a ser utilizada no ambiente do programa.

A barra de ferramentas chamada *ToolsSV* reproduz todas as opções dos cinco primeiros itens pertencentes ao menu ViSC, a saber: *Scalar*, *Scalar & Multivariable*, *Multivariable*, *Tensor* e *Vector*.

---

[81] O MFSV trabalha com a ferramenta *Color Map* e não oferece, ainda, recursos para outras ferramentas de visualização. Entretanto, estas ferramentas poderão ser implementas com relativa facilidade para o *desktop* do programa.

A visualização a ser gerada, do fenômeno estudo de caso proposto para este trabalho, tem sua formulação matemática descrita no capítulo 5. Em tal capítulo, está, também, a ilustração dos resultados obtidos pela apresentação de várias figuras.

A seleção da ferramenta *Color Map* na barra *ToolsSV*, ou opção equivalente de item de menu, inicializará, automaticamente, as seguintes linhas obrigatórias, excetuando o texto de comentário que seguem o sinal "#", na área de edição chamada *MathScripts*:

```
[COLOR_MAP]                          # linha obrigatória, que especifica a ferramenta
UNIDADES []                          # implementação futura na MathTools
# inclua as linhas necessárias de definição de valores e constantes
REGISTRE_COLOR_MAP(;;;;;;;;)        # registro das variáveis principais
                                     # no objeto da classe de visualização
# inclua as linhas à formulação matemática do fenômeno e filtros
```

É necessário completar o *script* matemático, segundo o estudo de caso, incluindo as linhas que descrevem os filtros formulados matematicamente. Todas as linhas necessárias são:

```
#01: [COLOR_MAP]
#02: UNIDADES [m; oC; W]          # um exemplo de aspecto que esta linha poderá assumir
#03: L = 0.184                                  # informa que a aleta tem 0,184m
#04: Tinf = 29.5                    # informa que a temperatura ambiente é de 29,5°C
#05: REGISTRE_COLOR_MAP(T; P; 4.55; 27.6; 30; x; 0; L; 30)
#06: P[4.55; 27.6] {  # Esta linha poderia omitir os valores para a faixa já definidos na anterior
#07:    To = 72.01223159*ln(0.03815367*P+1)+Tinf    # equação (22) do cap. 4
#08:    Teta = To - Tinf                            # equação (24) do cap. 4
#09:    m = 0.5474*ln(0.9262*Teta+1) +              # equação (23) do cap. 4
                  Teta^2/(574.0444*exp(Teta^2/132.25)) + 3.5765
#10:    x[0;L] {      # poder-se-ia omitir, tb., os valores para esta faixa de variação de valores
#11:       FILTRO {                  # especificação do filtro segundo a equação (25) do cap. 4
             1F: x(Tref)=L-acosh(cosh(L*m)*(Tref-Tinf)/Teta)/m .:
                   x(45.0) > x > x(60.0)  # intervalos de temp. para Tref na eq. (25)
          }
#12:       T = Tinf + Teta*cosh(m*(L-x))/cosh(m*L)    # equação (21) do cap. 4
      }
   }
```

Um resumo lógico das linhas apresentadas na listagem do *MathScripts* pode ser o seguinte:

#01: Definição da ferramenta de visualização utilizada: *Color Map*.

#02: Poderá ser utilizada para determinar qual o sistema de unidades físicas deve ser utilizado para homogeneizar os cálculos, transformando automaticamente unidades em escalas compatíveis.

#03: Definição do comprimento, L, da aleta de aço em 0,184m.

#04: Definição da temperatura ambiente, *Tinf*, em 29,5°C.

#05: Registro das variáveis principais formadoras da malha do *Color Map*, no objeto da classe de visualização, informando que a Temperatura, T, é a variável principal à visualização, que a Potência, P, varia de 4,55W a 27,66W e deve ser calculada para 30 pontos, e a distância, x, ao longo da aleta, varia desde a sua base *0* até seu comprimento máximo L, e deve ser calculada para 30 pontos.

#06: Definição do tipo do intervalo fechado para a faixa de cálculos subordinados a Potência, P. Os valores indicados para a faixa são de uso opcional.

#07: Cálculo da Temperatura Inicial, *To*, para a Potência, P, em um ponto.

#08: Cálculo de uma variável física auxiliar chamada *Teta*.

#09: Cálculo do fator m.

#10: Definição do tipo de intervalo fechado para a faixa de cálculos subordinados a Distância, x, ao longo da aleta. Os valores da faixa são opcionais.

#11: Definição de um filtro formulado matematicamente pela equação (25) apresentada no capítulo 4. Os contornos para este filtro exigem que as Temperaturas calculadas fiquem no intervalo fechado de 45°C a 60°C.

#12: Formulação matemática do fenômeno a ser visualizado: As Distribuições de Temperaturas. A equação (21), do capítulo 4, fornece todos os valores possíveis.

Após criação do texto matemático que descreve a visualização, o usuário poderá realizar os ajustes necessários à visualização ou gerar a visualização antes. Se optar por gerar a visualização deverá clicar sobre o botão *Converter* existente na janela de *scripts*. Esta ação fará a criação do código fonte em LIP em função do texto na área *MathScripts*, o qual o usuário poderá ou não modificar. Se o usuário não souber como se realiza a programação de computadores, não poderá implementar acréscimos no código. Entretanto, a visualização obtida pelo código gerado automaticamente na conversão poderá ser suficiente aos seus propósitos. A execução do código em LIP é obtida pelo acionamento do botão chamado *Executar* presente na janela de *scripts*. Após acionado este botão, o programa em LIP é executado e a visualização é apresentada na janela de visualização. Esta visualização e os dados pertencentes a ela permanecem, mesmo após o encerramento da execução do programa em LIP. Neste momento, o usuário poderá fazer os ajustes, ou completar os ajustes necessários pelos controles de visualização disponíveis no *desktop* do MFSV.

O código fonte resultante, em LIP, está descrito no Anexo 4. Não será realizada uma análise detalhada de seus comandos, porque não faz parte dos objetivos deste trabalho e seu significado lógico é o mesmo descrito para o texto escrito na área de *MathScripts*. Entretanto, são necessárias algumas observações relacionadas a este código:

- A Temperatura entendida pelo usuário simplesmente como variável, em LIP é convertida para uma matriz bidimensional de 30 por 30 elementos, em virtude da segmentação da malha proposta pelo usuário.
- O registro das variáveis é realizado em pontos apropriados em LIP.
- São criadas variáveis internas auxiliares. Estas variáveis extras possuem o prefixo "_" e são importantes para o fornecimento dos dados ao objeto da classe de visualização.
- A transcrição do filtro proposto no *MathScripts* é encontrada, em LIP, como uma estrutura condicional SE, a qual subordina a variável cor como sendo a variável classificadora do valor da Temperatura. Tal classificação coloca o valor da Temperatura como pertencente ou não à área filtrada.

7 CONCLUSÕES E TRABALHOS FUTUROS

Apesar da área de Visualização Científica ser relativamente recente e receber a adesão de novos cientistas, muitos pesquisadores desconhecem o seu significado e permanecem analisando seus problemas com ferramentas clássicas de geração de gráficos. Cabe aos pesquisadores e desenvolvedores da área de ViSC aprimorar e propagar as capacidades dos programas de visualização procurando realizar o desenvolvimento ou aprimoramento de tais programas com a participação de profissionais de outras áreas. As sugestões e métodos descritos por eles são primordiais, afinal são os usuários das ferramentas e o motivo pelo qual elas são desenvolvidas.

Este trabalho analisou alguns programas de visualização existentes e concluiu a ausência, até onde se sabe, da característica usada como proposição fundamental dessa dissertação, a qual é enunciada como a presença de filtros formulados matematicamente para a seleção de sub-regiões de dados, analíticos ou não, destinados à visualização, sendo esta característica uma opção inerente e interativa ao ambiente de um programa visualizador, e sendo que estes filtros, também chamados de físico-matemáticos, foram definidos como formulações de funções, equações, matrizes, entre outras. A conclusão da análise motivou o desenvolvimento de um protótipo de software visualizador que atendesse à proposição apresentada. Este programa, que foi desenvolvido segundo os paradigmas atuais da programação e com a utilização da biblioteca de funções gráficas OpenGL, foi chamado de *Mathematical Filters in Scientific Visualization*, ou pela abreviação MFSV.

Para a completa demonstração da proposição, foi necessária a aplicação do MFSV a um estudo de caso. A área escolhida foi a da Transferência de Calor e o estudo de caso escolhido e a ela relacionado é a transferência de calor suposta unidimensional ao longo de

uma aleta em regime térmico permanente. Houve a necessidade da criação de um artefato científico para a realização de uma experiência que fornecesse os dados para a visualização. A realização desta experiência forneceu os dados que foram formulados matematicamente. Entretanto, era necessário um exemplo de filtro físico-matemático e este foi obtido pela dedução de uma equação relacionada ao fenômeno.

A utilização do programa MFSV no estudo de caso escolhido, gerando visualizações filtradas segundo à proposta desta dissertação, demonstrando assim a proposição enunciada e este foi o resultado principal e mérito deste trabalho.

O resultado principal deste trabalho provou ser possível o desenvolvimento de programas segundo a proposição enunciada. Porém, a contribuição que tais programas podem oferecer é a sua aplicação em diversas áreas do conhecimento. Isto mostra, conseqüentemente, a sua importância. São inúmeros e evidentes os exemplos de aplicação que exigem visualizações por seções filtradas de dados.

Por se tratar de um protótipo, o MFSV tem muitas características a serem completadas e expandidas, e estendendo-se estas sugestões como pesquisa e desenvolvimento para outros programas de visualização. Algumas características importantes incluem:

A criação de *ViSCFriends*, os quais seriam assistentes do usuário na criação da visualização de dados filtrados que através de quadros parametrizados gerariam parte das linhas de comando na área do *MathScript*. Estes assistentes aumentariam, ainda mais, a interatividade com o usuário dispensando-o da obrigação de escrever sintaxes preestabelecidas no texto do *MathScript*. Um dos *ViSCFriends* a ser desenvolvido seria o chamado *DynaViSC*, o qual permitiria ao usuário o acompanhamento dinâmico de suas ações de criação de primitivas e demais condições para a visualização, transformando os passos do usuário em um *script* secundário de ações, o qual poderia ser incorporado ao contexto do

MathScript através de opções no ambiente do programa. Assim procedendo, a visualização de um determinado fenômeno poderia ser construída dinamicamente, o que evitaria a escrita e parametrização excessiva de comandos via código, reforçando a idéia de menor codificação por parte do usuário.

A criação da *MathTools*, a qual é uma paleta de opções matemáticas desde as opções clássicas de uma calculadora científica até os complexos métodos numéricos e outras formulações matemáticas como arranjos matriciais, cálculo vetorial, entre outras. Entre as diversas características que a *MathTools* poderia ter, poder-se-ia incluir o uso de representações mais fiéis à simbologia da notação matemática normalmente utilizada pelos pesquisadores e a conversão automática entre unidades de grandezas físicas que seria empregada na homogeneização de valores em unidades diferentes presentes nos cálculos das linhas existentes na área de edição do *MathScript*. A paleta *MathTools* aumentaria a qualidade interativa entre o usuário e o programa visualizador.

Se o desenvolvimento das características *ViSCFriends* e *MathTools* fosse apenas um trabalho complementar, realmente não teriam muito valor científico. Entretanto, para a implementação destas características há a necessidade de estudos quanto a forma de interatividade com o usuário em *ViSCFriends* e a característica de conversão automática de valores com unidades físicas diferentes em *MathTools*, sendo que esta última característica exigirá convenções e critérios especiais.

Finalmente, poder-se-ia propor a utilização de diversas metodologias e técnicas existentes nos atuais visualizadores e em relação aos programas que permitissem a seleção de dados por filtros formulados matematicamente. Entre estas diversas metodologias e técnicas podem ser citadas o uso de elementos finitos, a visualização baseada em arquiteturas paralelas, a técnica exploratória em tempo-real (*realtime*), a visualização em sistemas de redes e outras características mais avançadas.

ANEXO 1 - ANÁLISE DAS BIBLIOTECAS GRÁFICAS DIRECT3D E OPENGL

i) DIRECT 3D

A biblioteca gráfica *Direct3D* é um componente que suporta *rendering* e animação no desenvolvimento de aplicativos para o sistema operacional Windows e está intrinsecamente relacionado ao uso do compilador *Visual C++*, todos produtos da *Microsoft*.

Os recursos que a biblioteca de rotinas gráficas do *Direct3D* possui estão reunidos nos seguintes tópicos:

- *Mapeamento de texturas*: tratando da aplicação de texturas à malhas, da transparência e da animação destas texturas.
- *Luzes e Sombras*: métodos de *rendering* e modelos de cor, tipos de fontes luminosas e sombras.
- *Molduras e animação*: molduras para o posicionamento de objetos e animação.
- *Morphing*: tratando da modificação da forma ou aparência de objetos para outras formas ou aparências.
- *Viewports*: para definição de pontos de visão e ângulos de visão, e o uso de múltiplos *viewports*.

O suporte do *Direct3D* para várias placas de vídeo é realizado pelas rotinas de uma Interface para Programas Aplicativos (*API*) chamada de *DirectDraw*.

Uma possível vantagem no uso do *Direct3D*, por fazer parte do pacote *DirectX* e em relação a outras bibliotecas, seria a inclusão de recursos para áudio nos aplicativos. Entretanto, o programa MFSV não precisa de tal recurso.

Apesar de todos os recursos apresentados pelo *Direct3D*, foram duas as razões para não utilizá-lo no desenvolvimento do MFSV:

- Pouca portabilidade, restringindo o seu uso ao sistema operacional Windows.
- O ambiente de programação visual aconselhado para seu uso era o *Visual C++*, e a decisão para implementação recaiu sobre o *C++Builder* produto da divisão *Borland da Inprise Inc*, como descrito no item 6.2 deste capítulo.

ii) OPENGL

a) Fundamentos da Biblioteca OpenGL

Devido à quantidade de situações em que o OpenGL pode ser empregado, os programas podem ser tornar complexos. Porém, a estrutura básica de um programa útil é simples e, essencialmente, está composta pelas seguintes ações:

- Inicialização da interface gráfica pelo recursos de gerenciamento de janelas do sistema hospedeiro
- Realização das funções OpenGL que estarão vinculadas aos eventos de controle do sistema operacional
- Atualização da janela gráfica e controle dos eventos pelo sistema operacional

Um exemplo de código em OpenGL que gera um quadrado no centro da janela de saída gráfica do sistema é apresentado na seguinte listagem de código fonte em linguagem C:

```
main()
{
  InitializaAWindowPlease();          // incialização da interface gráfica
  glClearColor(0.0,0.0,0.0,0.0);      // incialização da interface gráfica
  glClear(GL_COLOR_BUFFER_BIT);       // apaga a tela com a cor de fundo
  glColor3f(1.0,1.0,1.0);             // muda a cor corrente de desenho
  glOrtho(0.0,1.0,0.0,1.0,-1.0,1.0); // normaliza coordenadas da janela
  glBegin(GL_POLYGON);                // inicializa função gráfica
  glVertex3f(0.25, 0.25, 0.0);        // insere primeiro vértice
  glVertex3f(0.75, 0.25, 0.0);        // insere segundo vértice
  glVertex3f(0.75, 0.75, 0.0);        // insere terceiro vértice
  glVertex3f(0.25, 0.75, 0.0);        // insere último vértice
  glEnd();                            // finaliza função de desenho
  glFlush();                          // força a realização efetiva dos
                                      // gráficos até então produzidos
  UpdateTheWindowAndCheckForEvents(); // controla a janela e eventos
  return 0;
}
```

O OpenGL não dispõe de rotinas para inicialização, controle de janelas e eventos. As funções `InitializaAWindowPlease()` e `UpdateTheWindowAndCheckForEvents()`, devem ser recursos existentes no sistema operacional hospedeiro. A função `glBegin()` determina o tipo de primitiva gráfica a ser criada e o início da seqüência que formará tal primitiva. A função `glFlush()` garante que as primitivas gráficas geradas sejam apresentadas na tela, porque o sistema constrói as primitivas em um *buffer*[82] antes de apresentá-la na tela. As linhas de código existentes entre as funções `glBegin()` e `glEnd()` são as responsáveis pela criação do quadrado na área gráfica.

Obviamente, para que se possa empregar o OpenGL, é necessário entender um pouco sobre a sintaxe de suas funções. As funções, normalmente, têm as letras `gl` como prefixo, as quais caracterizam as funções como pertencentes à biblioteca OpenGL. Após este prefixo vem o nome identificador da função e, após tal nome, podem vir sufixos que indicam

---

[82] *Buffer* é uma área da memória *RAM* destinada ao armazenamento provisório ou transiente de dados.

a quantidade de parâmetros necessários para o uso da função e os tipos deles. O fato é que o OpenGL possui famílias de funções similares. Um exemplo disso, é a função `glVertex3f()` usada no exemplo do programa do quadrado. Esta função estabelece as coordenadas de um vértice no espaço. Todavia, podemos entender o fornecimento de seus parâmetros de várias formas, a saber: O sufixo `3f` indica que são necessários, à função, três parâmetros que são do tipo `float`[83]. Os três valores *(x, y, z)* representarão as coordenadas de um ponto no espaço tridimensional. Poder-se-ia utilizar outras funções da mesma família como: `glVertex3d()`, a qual exige que as três coordenadas passadas sejam do tipo `double`[84]. Poderiam ser usadas as funções para o fornecimento de um ponto no espaço bidimensional. Neste caso, um exemplo, é a função `glVertex2f()`, onde o número `2` indica que são necessários dois parâmetros *(x, y)* do tipo `float`. Os recursos da biblioteca OpenGL ficam disponíveis pela chamada através de linguagens de programação, como a linguagem C. O OpenGL fornece alguns tipos intrínsecos e estes estão relacionados na Tabela 13.

TABELA 13: TIPOS DO OPENGL E SUFIXOS DE TIPOS EMPREGADOS NAS FAMÍLIAS DE FUNÇÕES

SUFIXO	TAMANHO EM BITS DO DADO	TIPO EM OPENGL	CORRESPONDENTE EM LINGUAGEM C
b	8-bit integer	GLbyte	signed char
s	16-bit integer	GLshort	short
i	32-bit integer	GLint, GLsizei	int or long
f	32-bit floating-point	GLfloat, GLclampf	float
d	64-bit floating point	GLdouble, GLclampd	double
ub	8-bit unsigned integer	GLubyte, GLboolean	unsigned char
us	16-bit unsigned integer	Glushort	unsigned short
ui	32-bit unsigned integer	GLuint, Glenum, GLbitfield	unsigned int, unsigned long

[83] O tipo é uma palavra declarativa de linguagens de programação utilizado para a declaração de identificadores como, por exemplo: as variáveis. O tipo *float* é utilizado para declarar identificadores reais de precisão simples.

[84] O tipo *double* é, também, utilizado para declarar identificadores reais, mas com precisão chamada dupla.

É importante lembrar que o OpenGL permite, ainda através de suas funções, o fornecimento dos parâmetros às funções, de mesma família, na forma de um vetor[85]. O sufixo de tipo, neste caso, empregado pelo OpenGL é constituído por um dos sufixos apresentados na Tabela 13 seguido pela letra "`v`".

b) Gerenciamento de Estados e Objetos Geométricos

O OpenGL funciona como uma máquina de estados. Um estado é uma característica que depois de estabelecida, permanece vigente para as ações subseqüentes até a sua próxima alteração. Exemplos de variáveis de estado são abundantes em OpenGL, assim como as funções que as alteram. O estabelecimento da cor de traçado gráfico, por exemplo, é representado por uma função, `glColor*()`[86], que altera a respectiva variável de estado.

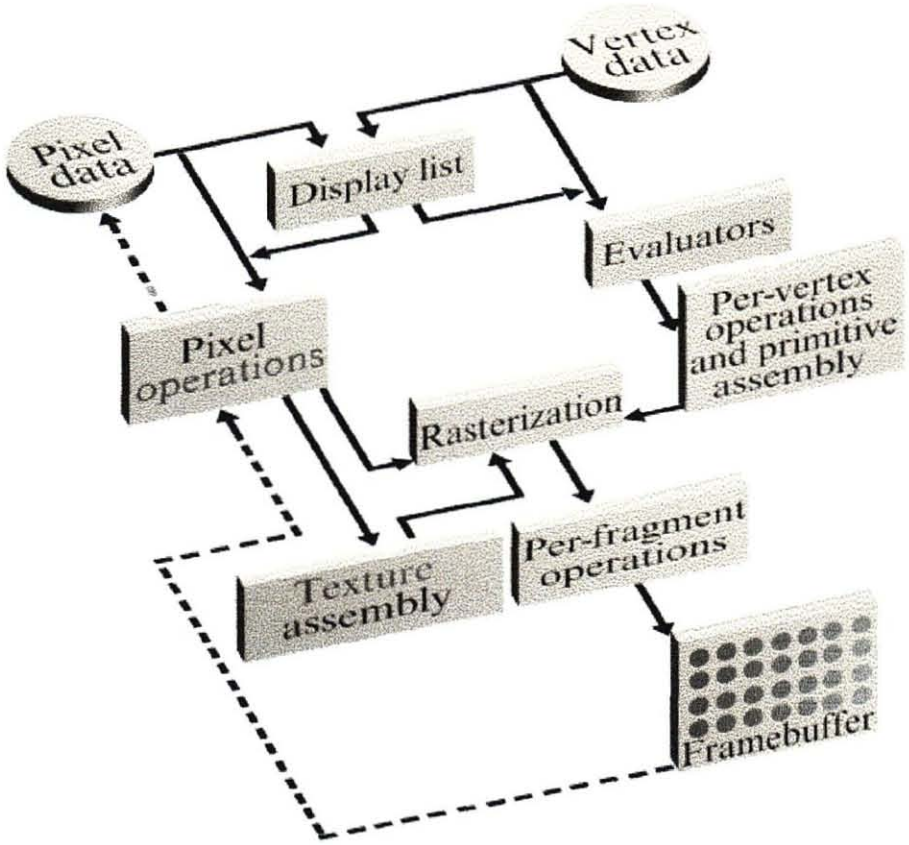
A maioria das implementações de OpenGL tem a mesma ordem de operações, uma série de estágios de processamento chamada *OpenGL Rendering Pipeline*. A Figura 21 mostra um esquema de como o tratamento de dados, a ordem das operações e a obtenção da saída gráfica ocorrem no OpenGL.

---

[85] Caso seja utilizada a linguagem C é necessário, portanto, passar o endereço na memória do parâmetro.

[86] O * indica, na notação do OpenGL, que existe uma família de funções associadas ao comando com diferentes sufixos.

FIGURA 21: ESTÁGIOS DE PROCESSAMENTO DO OPENGL: *OPENGL RENDERING PIPELINE*



FONTE: WOO ET. ALLI. (1999, P. 11)

O OpenGL trabalha, essencialmente, por dois caminhos:

- Processamento dos dados geométricos como vértices, linhas e polígonos.
- Processamento de *pixels* como operações de *pixels* e texturas

Em qualquer um dos dois caminhos podem ser utilizadas as *Display Lists*, as quais são registros, identificados por nomes, de todos os passos para construção dos dados geométricos e que podem ser utilizados em outro momento. A *display list*, gerada, poderá ser executada posteriormente de forma mais rápida que a reexecução das linhas de funções que a formaram originalmente. Esta característica propicia maior versatilidade às operações e transformações matriciais de geração e trabalho com primitivas geométricas. As *display lists* não são de caráter obrigatório e não são aplicáveis a todas as circunstâncias de geração de dados.

Os *Evaluators* trabalham na otimização da parametrização de primitivas geométricas curvilíneas e superfícies. Muitas vezes a descrição de uma destas primitivas pode ser realizada através de uma expressão polinomial para os pontos de controle da primitiva. Os *evaluators* provêm um método para a derivação de vértices usados para representar a superfície por meio de pontos de controle. Tal método é um mapeamento polinomial, o qual poderá produzir uma superfície normal, coordenadas de textura, cores, e valores espaciais de coordenadas dos pontos de controle.

As operações *Per-Vertex* transformam matricialmente os dados de vértices para as primitivas escolhidas na programação. São levados em conta a construção da primitiva e a possível escala de cores entre dois vértices estabelecidos.

O estágio das *Primitives Assembly* garante que porções não visíveis de objetos sejam descartadas ou realiza os cortes definidos por planos preestabelecidos.

As operações de *Pixel* efetuam todas as operações de tratamento dos valores de cores e posição dos bits nos *bitplanes*[87] do *framebuffer*.

A *Texture Assembly* permite, entre outras coisas, o mapeamento de texturas sobre objetos para dar maior realismo às cenas produzidas.

O processo de *Rasterization* é a conversão dos dados geométricos ou de *pixel* para construção de fragmentos. Cada fragmento retangular representa um *pixel* no *framebuffer*. Linhas e polígonos, espessura de linhas, tamanho de pontos, modelos de sombreamento, cálculo do *antialiasing* são elementos resolvidos neste estágio.

---

[87] Os *bitplanes* são áreas da memória RAM, de um computador, destinadas ao armazenamento de informações de *bits* dos *pixels*, os quais são os elementos gráficos básicos de formação das imagens em um monitor de vídeo.

As *Fragment Operations* ocorrem antes da informação ir para o *framebuffer*. Exemplos de *fragment operations* compreendem: a texturização , efeitos de *fog*, canal alfa, entre outros.

Entre as funções `glBegin()` e `glEnd()` são criados os objetos geométricos no OpenGL, os quais são constituídos por primitivas gráficas. O tipo da primitiva é informado como parâmetro da função `glBegin(param)` e os valores que pode assumir são:

- `GL_POINTS` - gerar vértices.
- `GL_LINES` - gerar linhas independentes.
- `GL_LINE_STRIP` - gerar linhas poligonais.
- `GL_LINE_LOOP` - gerar linhas poligonais que sejam polígonos.
- `GL_TRIANGLES` - gerar faces triangulares independentes.
- `GL_TRIANGLE_STRIP` - gerar faces triangulares adjacentes duas a duas.
- `GL_TRIANGLE_FUN` - gerar faces triangulares adjacente com um ponto comum.
- `GL_QUADS` - gerar superfícies quadrangulares independentes.
- `GL_QUAD_STRIP` - gerar superfícies quadrangulares adjacentes duas a duas.
- `GL_POLYGON` - gerar uma superfície com n vértices.

c) Posição do Observador e dos Objetos

Posicionar os objetos no espaço e orientar a posição de observação utilizando recursos de programação em OpenGL, significa utilizar o controle interno das operações matriciais de transformação do OpenGL. Translações, rotações, alterações de escala, transformação de perspectiva, além de outras, são realizadas simplesmente pela chamada à função apropriada.

O OpenGL provê, ainda, recursos para criação de planos de *clipping* e implementa todas as suas operações matriciais como estruturas de dados pilha. Outra possibilidade fornecida por esta biblioteca gráfica é a reversão de uma coordenada de tela para a obtenção das coordenadas mundo, um recurso particularmente útil à área da Visão Computacional, a qual está definida em GOMES (1994, p. 2) como: "A área que tem por finalidade obter, a partir de uma imagem (entrada), as informações geométricas, topológicas ou físicas sobre o cenário que deu origem a essa imagem".

O gráfico da Figura 22 apresenta o esquema dos estágios das transformações de vértices utilizado no OpenGL.

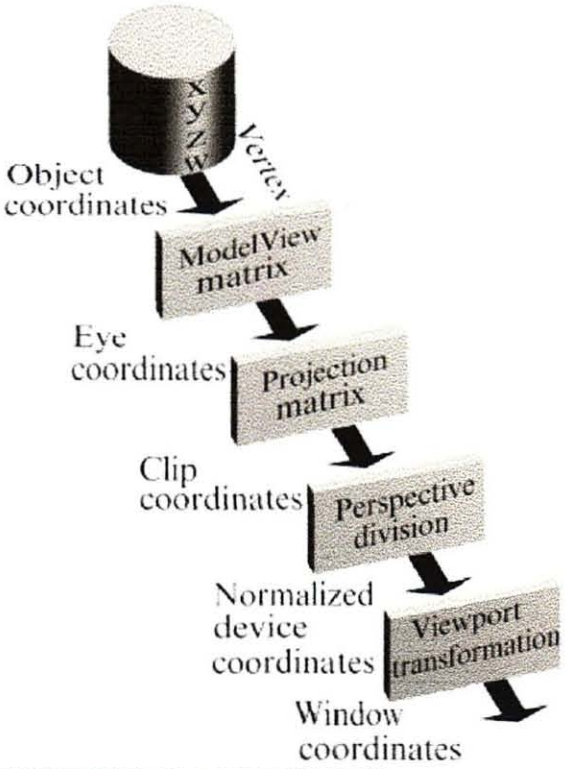
As matrizes de coordenadas são de dimensões 4x4 (aqui chamada de M) e as transformações decorrentes sobre um vértice (chamado v) têm a seguinte representação:

$$v' = Mv. \qquad (27)$$

O OpenGL unifica as operações de *Viewing*, a qual é a determinação do ponto de observação no mundo, e *Modeling*, a qual é a determinação da posição dos objetos no mundo, em uma mesma matriz de transformação chamada de *ModelView Matrix*. A razão para isto é que tanto faz deslocarmos todos os objetos em relação a uma origem ou deslocar o sistema de coordenadas em relação aos objetos. Logicamente, este tipo de abordagem traz algumas considerações que devem ser levadas em conta quando existir a necessidade de estabelecer a ordem que várias transformações matriciais, sucessivas, possam tomar. Uma vez que os produtos matriciais não são comutativos, uma rotação antes de uma translação ou uma translação antes de uma rotação, por exemplo, não produzem os mesmos resultados. Tudo irá depender de como são interpretadas as transformações em OpenGL, se sobre os objetos ou se sobre o sistema de coordenadas. A literatura a respeito do OpenGL, como em WOO (1997), sugere que se dê a preferência à interpretação de um sistema de coordenadas relativo e que,

portanto, as transformações matricias incidam sobre o sistema de coordenadas, em virtude da ordem natural de leitura dos produtos matriciais que tal interpretação promove.

FIGURA 22: ESTÁGIOS DAS TRANSFORMAÇÕES DE VÉRTICES DO OPENGL



FONTE: WOO ET. ALLI. (1999, P. 98)

As transformações de projeção, *Projection Matrix* na Figura 22, são as equivalentes à escolha de uma lente e ajustes de enquadramento em uma máquina fotográfica. São determinados os planos de corte frontal e posterior, além dos planos de corte à esquerda e à direita, e os planos superior e inferior formando-se, desta forma, o chamado Volume de Visualização. O volume de visualização determinará, portanto, o que deve ou não aparecer na cena escolhida e de que forma tais objetos aparecerão, se em perspectiva ou em vista ortográfica. Evidentemente, em Computação Gráfica, pode-se obter vistas de modelos a partir de vistas ortográficas, porém também por vistas a partir de transformações de perspectiva. Portanto, o volume de visualização, se do tipo para perspectiva ou se do tipo para vista ortográfica, conterá os objetos, ou parte deles, que aparecerão na visualização. Um volume de visualização tem a forma de um tronco de pirâmide, quando associado a uma perspectiva e a forma de um prisma de seção reta, quando associado a uma vista ortográfica.

A próxima fase é a das transformações de divisão de perspectiva, *Perspective Division* na Figura 22, onde as coordenadas são normalizadas segundo o volume de visualização escolhido e levadas à área de saída gráfica definida, chamada de *Viewport*.

Finalmente, as transformações de saída gráfica, na Figura 22 chamada de *Viewport Transformation*, onde os valores de coordenadas mundo são calculados para valores de coordenadas tela para o dispositivo de saída gráfica do sistema de computação. Nesta fase ocorrem, também, algumas operações de *pixels* antes da apresentação do *framebuffer*.

As transformações matriciais são controlada por estruturas de dados chamadas pilhas e, portanto, o controle da entrada e saída das matrizes nas pilhas do OpenGL determinará o sucesso para as transformações desejadas. O OpenGL fornece recursos para salvar o contexto das pilhas e sua recuperação posterior.

d) Uso de Cores

Não é necessário o detalhamento extenso e completo sobre a teoria física da percepção das cores para seu emprego neste trabalho. A descrição rápida dos principais aspectos das funções de OpenGL relacionados ao emprego de cores será suficiente. Outro aspecto a considerar é que o *hardware* de saída gráfica podem variar na quantidade de cores que pode representar. Para todos os efeitos, consideraremos um sistema padrão com 24 bits, também chamado *True Color*.

O OpenGL fornece dois modos de escolha de cores, a saber: Modo RGBA (*Red, Green, Blue, Alpha*) e o Modo *Color-Index*. Para cada um dos modos existe um conjunto particular de funções definidas em OpenGL.

Notoriamente, a família de funções mais importante para escolha de cores no modo

RGBA do OpenGL é a família `glColor*()`. Dependendo do sufixo escolhido para esta função, ter-se-á uma especificação particular para os parâmetros. Se, por exemplo, for utilizado o sufixo `4ub` os valores para cada canal RGBA deverão estar compreendidos na faixa de números inteiros de 0 a 255. Caso o sufixo fosse `4f` os valores para tais canais deveriam estar na faixa de números reais de 0 a 1, a qual é chamada Faixa Normalizada de Cores.

O OpenGL fornece a família de funções `glIndex*()` para o modo indexado após a criação de uma tabela de cores, também conhecida como *Color Map*, ou *Lookup Table*. Neste caso, é necessário um único parâmetro numérico passado à função da família, o qual é entendido como um índice para a tabela de cores escolhidas.

Adotou-se, neste trabalho, a utilização do modo RGBA com o *rendering* gerado pelo método chamado *Smooth*.

Um último aspecto relacionado ao emprego de cores está baseado na característica do OpenGL ser uma máquina de estados e ter a entrada em separado para cada um dos vértices que compõem a primitiva. Portanto, pode-se estabelecer cores diferentes entre as chamadas à função que define as coordenadas de cada vértice. Se existir uma aresta entre dois vértices, sendo cada vértice criado segundo uma especificação de cor diferente, esta aresta possuirá um gradiente de cores, as quais são o resultado de uma interpolação linear em relação às especificações de cor dos dois vértices. Existindo faces cujos vértices apresentem diferentes cores, se dará o mesmo efeito em relação aos pontos das superfícies destas faces. No MFSV tal recurso foi utilizado para interpolar linearmente os valores de cores entre dois pontos consecutivos da malha.

e) Canal Alfa

Durante o desenvolvimento do MFSV foi decidida a utilização da propriedade de níveis de transparência fornecida pelo canal alfa para dar uma escala de importância, maior ou menor, às regiões filtradas. Uma região filtrada possui um formato equacionado de recorte de uma superfície ou de um volume. Esta área selecionada poderá ser a única apresentada na visualização, ou ser a mais destacada em relação ao restante da superfície ou do volume, bastando, para tanto, aumentar ou diminuir o valor estabelecido para o canal alfa das cores desta região filtrada.

O OpenGL pode fornecer a especificação do valor do canal alfa pelas famílias de funções que definem as cores. Para tanto, basta anexar ao sufixo empregado na função o número 4, o qual informa que deverá existir o parâmetro referente ao canal alfa. São necessárias outras configurações para um bom uso da propriedade do canal alfa, mas neste trabalho o detalhamento delas não será tratado. Algumas destas configurações encontram-se comentadas no código fonte do MFSV.

f) Uso de *Bitmaps*

Existem inúmeros recursos relacionados ao uso de *bitmaps* com o OpenGL, mas para as finalidades deste trabalho a atenção especial foi a utilização dos *bitmaps* com o único propósito de salvar em disco e carregar do disco arquivos de imagem das visualizações geradas. Aliás as figuras presentes no capítulo 5 foram geradas e salvas a partir do MFSV.

ANEXO 2 - VALORES DAS MEDIÇÕES EFETUADAS NA EXPERIÊNCIA DE TRANSFERÊNCIA DE CALOR EM ALETAS E A COMPARAÇÃO ENTRE AS LÂMPADAS UTILIZADAS NO ARTEFATO E AS RESISTÊNCIAS ELÉTRICAS EQUIVALENTES.

Legenda: U = Tensão em Volts

A = Corrente Elétrica em Amperes

R = Resistência Elétrica Equivalente em Ohms

L = Lâmpada com Potência Nominal em Watts

p*l* = circuito paralelo das lâmpadas e *l* posição da lâmpada no circuito

s = valor efetivo de saída no sistema aquecedor.

Cada item corresponde a um ajuste na tensão da fonte e quantidade e potência das lâmpadas utilizadas.

a) Fonte: $U_e = 30$ V

$U_p = 24{,}4$ V

$A_{p1} = 0{,}350$ A $\Leftrightarrow$ $R_{p1} = 69{,}70\ \Omega$ $\Leftrightarrow$ $L_{p1} = 100$ W

$A_{p2} = 0$ A $\Leftrightarrow$ $R_{p2} = \infty\ \Omega$ $\Leftrightarrow$ $L_{p2} = 0$ W

$A_{p3} = 0$ A $\Leftrightarrow$ $R_{p3} = \infty\ \Omega$ $\Leftrightarrow$ $L_{p3} = 0$ W

$A_p = 0{,}350$ A $\Leftrightarrow$ $R_p = 69{,}70\ \Omega$ $\Leftrightarrow$ $L_p = 100$ W

U_s = 4,7 V $\Leftrightarrow$ A_s = 0,350 A $\Leftrightarrow$ P_s = 1,64 W

b) Fonte: $U_e = 60$ V

$U_p = 47{,}8$ V

$A_{p1} = 0{,}500$ A $\Leftrightarrow$ $R_{p1} = 95{,}60\,\Omega$ $\Leftrightarrow$ $L_{p1} = 100$ W

$A_{p2} = 0$ A $\Leftrightarrow$ $R_{p2} = \infty\,\Omega$ $\Leftrightarrow$ $L_{p2} = 0$ W

$A_{p3} = 0$ A $\Leftrightarrow$ $R_{p3} = \infty\,\Omega$ $\Leftrightarrow$ $L_{p3} = 0$ W

$A_p = 0{,}500$ A $\Leftrightarrow$ $R_p = 95{,}60\,\Omega$ $\Leftrightarrow$ $L_p = 100$ W

U_s = 6,7 V $\Leftrightarrow$ A_s = 0,500 A $\Leftrightarrow$ P_s = 3,33 W

c) Fonte: $U_e = 30$ V

$U_p = 19{,}2$ V

$A_{p1} = 0{,}308$ A $\Leftrightarrow$ $R_{p1} = 62{,}40\,\Omega$ $\Leftrightarrow$ $L_{p1} = 100$ W

$A_{p2} = 0{,}307$ A $\Leftrightarrow$ $R_{p2} = 62{,}40\,\Omega$ $\Leftrightarrow$ $L_{p2} = 100$ W

$A_{p3} = 0$ A $\Leftrightarrow$ $R_{p3} = \infty\,\Omega$ $\Leftrightarrow$ $L_{p3} = 0$ W

$A_p = 0{,}615$ A $\Leftrightarrow$ $R_p = 31{,}20\,\Omega$ $\Leftrightarrow$ $L_p = 200$ W

U_s = 8,3 V $\Leftrightarrow$ A_s = 0,615 A $\Leftrightarrow$ P_s = 5,10 W

d) Fonte: $U_e = 30$ V

$U_p = 16{,}1$ V

$A_{p1} = 0{,}277$ A $\Leftrightarrow$ $R_{p1} = 58{,}20\,\Omega$ $\Leftrightarrow$ $L_{p1} = 100$ W

$A_{p2} = 0{,}277$ A $\Leftrightarrow$ $R_{p2} = 58{,}20\,\Omega$ $\Leftrightarrow$ $L_{p2} = 100$ W

$A_{p3} = 0{,}276$ A $\Leftrightarrow$ $R_{p3} = 58{,}20\,\Omega$ $\Leftrightarrow$ $L_{p3} = 100$ W

$A_p = 0{,}830$ A $\Leftrightarrow$ $R_p = 19{,}40\,\Omega$ $\Leftrightarrow$ $L_p = 300$ W

U_s = 11,3 V $\Leftrightarrow$ A_s = 0,830 A $\Leftrightarrow$ P_s = 9,46 W

e) Fonte: $U_e = 60$ V

$U_p = 39{,}1$ V

$A_{p1} = 0{,}438$ A $\Leftrightarrow$ $R_{p1} = 89{,}40\,\Omega$ $\Leftrightarrow$ $L_{p1} = 100$ W

$A_{p2} = 0{,}437$ A $\Leftrightarrow$ $R_{p2} = 89{,}40\,\Omega$ $\Leftrightarrow$ $L_{p2} = 100$ W

$A_{p3} = 0$ A $\Leftrightarrow$ $R_{p3} = \infty\,\Omega$ $\Leftrightarrow$ $L_{p3} = 0$ W

$A_p = 0{,}875$ A $\Leftrightarrow$ $R_p = 44{,}70\,\Omega$ $\Leftrightarrow$ $L_p = 200$ W

U_s = 11,8 V $\Leftrightarrow$ A_s = 0,875 A $\Leftrightarrow$ P_s = 10,31 W

f) Fonte: $U_e = 30$ V

$U_p = 13{,}5$ V

$A_{p1} = 0{,}511$ A $\Leftrightarrow$ $R_{p1} = 26.48\,\Omega$ $\Leftrightarrow$ $L_{p1} = 200$ W

$A_{p2} = 0{,}256$ A $\Leftrightarrow$ $R_{p2} = 52{,}81\,\Omega$ $\Leftrightarrow$ $L_{p2} = 100$ W

$A_{p3} = 0{,}153$ A $\Leftrightarrow$ $R_{p3} = 88{,}30\,\Omega$ $\Leftrightarrow$ $L_{p3} = 60$ W

$A_p = 0{,}920$ A $\Leftrightarrow$ $R_p = 14{,}70\,\Omega$ $\Leftrightarrow$ $L_p = 360$ W

U_s = 12,2 V $\Leftrightarrow$ A_s = 0,920 A $\Leftrightarrow$ P_s = 11,03 W

g) Fonte: $U_e = 30$ V

$U_p = 13{,}5$ V

$A_{p1} = 0{,}485$ A $\Leftrightarrow$ $R_{p1} = 27{,}80\,\Omega$ $\Leftrightarrow$ $L_{p1} = 200$ W

$A_{p2} = 0{,}485$ A $\Leftrightarrow$ $R_{p2} = 27{,}80\,\Omega$ $\Leftrightarrow$ $L_{p2} = 200$ W

$A_{p3} = 0$ A $\Leftrightarrow$ $R_{p3} = \infty\,\Omega$ $\Leftrightarrow$ $L_{p3} = 0$ W

$A_p = 0{,}970$ A $\Leftrightarrow$ $R_p = 13{,}90\,\Omega$ $\Leftrightarrow$ $L_p = 400$ W

U_s = 13,4 V $\Leftrightarrow$ A_s = 0,970 A $\Leftrightarrow$ P_s = 13,30 W

h) Fonte: $U_e = 30$ V

$U_p = 10{,}3$ V

$A_{p1} = 0{,}442$ A $\Leftrightarrow$ $R_{p1} = 23{,}25\ \Omega$ $\Leftrightarrow$ $L_{p1} = 200$ W

$A_{p2} = 0{,}442$ A $\Leftrightarrow$ $R_{p2} = 23{,}25\ \Omega$ $\Leftrightarrow$ $L_{p2} = 200$ W

$A_{p3} = 0{,}221$ A $\Leftrightarrow$ $R_{p3} = 46{,}50\ \Omega$ $\Leftrightarrow$ $L_{p3} = 100$ W

$A_p = 1{,}105$ A $\Leftrightarrow$ $R_p = 9{,}30\ \Omega$ $\Leftrightarrow$ $L_p = 500$ W

$U_s = 14{,}9$ V $\Leftrightarrow$ $A_s = 1{,}105$ A $\Leftrightarrow$ $P_s = 16{,}45$ W

i) Fonte: $U_e = 60$ V

$U_p = 33{,}6$ V

$A_{p1} = 0{,}392$ A $\Leftrightarrow$ $R_{p1} = 85{,}80\ \Omega$ $\Leftrightarrow$ $L_{p1} = 200$ W

$A_{p2} = 0{,}392$ A $\Leftrightarrow$ $R_{p2} = 85{,}80\ \Omega$ $\Leftrightarrow$ $L_{p2} = 200$ W

$A_{p3} = 0{,}391$ A $\Leftrightarrow$ $R_{p3} = 85{,}80\ \Omega$ $\Leftrightarrow$ $L_{p3} = 100$ W

$A_p = 1{,}175$ A $\Leftrightarrow$ $R_p = 28{,}60\ \Omega$ $\Leftrightarrow$ $L_p = 500$ W

$U_s = 15{,}9$ V $\Leftrightarrow$ $A_s = 1{,}175$ A $\Leftrightarrow$ $P_s = 18{,}73$ W

j) Fonte: $U_e = 30$ V

$U_p = 8{,}6$ V

$A_{p1} = 0{,}410$ A $\Leftrightarrow$ $R_{p1} = 21{,}00\ \Omega$ $\Leftrightarrow$ $L_{p1} = 200$ W

$A_{p2} = 0{,}410$ A $\Leftrightarrow$ $R_{p2} = 21{,}00\ \Omega$ $\Leftrightarrow$ $L_{p2} = 200$ W

$A_{p3} = 0{,}410$ A $\Leftrightarrow$ $R_{p3} = 21{,}00\ \Omega$ $\Leftrightarrow$ $L_{p3} = 200$ W

$A_p = 1{,}230$ A $\Leftrightarrow$ $R_p = 7{,}00\ \Omega$ $\Leftrightarrow$ $L_p = 600$ W

$U_s = 16{,}7$ V $\Leftrightarrow$ $A_s = 1{,}230$ A $\Leftrightarrow$ $P_s = 20{,}66$ W

k) Fonte: $U_e = 60$ V

$U_p = 28{,}0$ V

$A_{p1} = 0{,}714$ A $\Leftrightarrow$ $R_{p1} = 39{,}27\,\Omega$ $\Leftrightarrow$ $L_{p1} = 200$ W

$A_{p2} = 0{,}357$ A $\Leftrightarrow$ $R_{p2} = 78{,}32\,\Omega$ $\Leftrightarrow$ $L_{p2} = 100$ W

$A_{p3} = 0{,}214$ A $\Leftrightarrow$ $R_{p3} = 130{,}95\Omega$ $\Leftrightarrow$ $L_{p3} = 60$ W

$A_p = 1{,}285$ A $\Leftrightarrow$ $R_p = 21{,}80\,\Omega$ $\Leftrightarrow$ $L_p = 360$ W

U_s = 17,3 V $\Leftrightarrow$ A_s = 1,285 A $\Leftrightarrow$ P_s = 22,17 W

l) Fonte: $U_e = 60$ V

$U_p = 27{,}9$ V

$A_{p1} = 0{,}715$ A $\Leftrightarrow$ $R_{p1} = 39{,}00\,\Omega$ $\Leftrightarrow$ $L_{p1} = 200$ W

$A_{p2} = 0{,}715$ A $\Leftrightarrow$ $R_{p2} = 39{,}00\,\Omega$ $\Leftrightarrow$ $L_{p2} = 200$ W

$A_{p3} = 0$ A $\Leftrightarrow$ $R_{p3} = \infty\,\Omega$ $\Leftrightarrow$ $L_{p3} = 0$ W

$A_p = 1{,}430$ A $\Leftrightarrow$ $R_p = 19{,}50\,\Omega$ $\Leftrightarrow$ $L_p = 400$ W

U_s = 19,3 V $\Leftrightarrow$ A_s = 1,430 A $\Leftrightarrow$ P_s = 27,60 W

m) Fonte: $U_e = 60$ V

$U_p = 22{,}4$ V

$A_{p1} = 0{,}622$ A $\Leftrightarrow$ $R_{p1} = 36{,}00\,\Omega$ $\Leftrightarrow$ $L_{p1} = 200$ W

$A_{p2} = 0{,}622$ A $\Leftrightarrow$ $R_{p2} = 36{,}00\,\Omega$ $\Leftrightarrow$ $L_{p2} = 200$ W

$A_{p3} = 0{,}311$ A $\Leftrightarrow$ $R_{p3} = 72{,}00\,\Omega$ $\Leftrightarrow$ $L_{p3} = 100$ W

$A_p = 1{,}555$ A $\Leftrightarrow$ $R_p = 21{,}80\,\Omega$ $\Leftrightarrow$ $L_p = 500$ W

U_s = 21,2 V $\Leftrightarrow$ A_s = 1,555 A $\Leftrightarrow$ P_s = 33,29 W

n) Fonte: $U_e = 60$ V

$U_p = 19{,}1$ V

$A_{p1} = 0{,}584$ A $\Leftrightarrow$ $R_{p1} = 32{,}70\ \Omega$ $\Leftrightarrow$ $L_{p1} = 200$ W

$A_{p2} = 0{,}583$ A $\Leftrightarrow$ $R_{p2} = 32{,}70\ \Omega$ $\Leftrightarrow$ $L_{p2} = 200$ W

$A_{p3} = 0{,}583$ A $\Leftrightarrow$ $R_{p3} = 32{,}70\ \Omega$ $\Leftrightarrow$ $L_{p3} = 200$ W

$A_p = 1{,}750$ A $\Leftrightarrow$ $R_p = 10{,}90\ \Omega$ $\Leftrightarrow$ $L_p = 600$ W

U_s = 23,8 V $\Leftrightarrow$ A_s = 1,750 A $\Leftrightarrow$ P_s = 41,96 W

A seguir é apresentado um resumo de todas as Tensões e Potências empregadas, e Temperaturas em graus centígrados medidas na base da aleta de aço, junto à superfície do aquecedor, onde a temperatura é a inicial e mais alta.

U_s (V)	P_s (W)	T_o (°C)
4,7 V	1,64 W	33,83 °C
6,7 V	3,33 W	38,03 °C
8,3 V	5,10 W	42,18 °C
11,3 V	9,46 W	51,49 °C
11,8 V	10,31 W	53,17 °C
12,2 V	11,03 W	54,57 °C
13,4 V	13,30 W	58,80 °C
14,9 V	16,45 W	64,28 °C
15,9 V	18,73 W	68,01 °C
16,7 V	20,66 W	71,02 °C
17,3 V	22,17 W	74,01 °C
19,3 V	27,60 W	80,90 °C
21,2 V	33,29 W	88,11 °C
23,8 V	41,96 W	97,86 °C

ANEXO 3 - OPÇÕES CRIADAS SEGUNDO OS MENUS DO DESKTOP DO PROGRAMA VISUALIZADOR MFSV.

Por se tratar de um protótipo de programa visualizador muitas das opções propostas nos menus e barras de ferramentas não foram implementadas. As opções codificadas e existentes nos menus são as seguintes:

- Menu **Arquivo.**
  - *Novo* : Cria novos scripts vazios.
  - *Abrir*: Abre *scripts* existentes a partir de arquivos em disco que possuem a extensão *.svs* (*Scientific Visualization Script*).
  - *Salvar Script*: Salva com o nome atual os *scripts* atuais.
  - *Salvar Script Como*: Permite a mudança do nome dos scripts no momento de salvá-los.
  - *Fechar Script*: Fecha *scripts* atuais, mas solicita antes a confirmação do usuário
  - *Abrir Imagem*: Carrega uma imagem *Bitmap* para a área de visualização.
  - *Configurar impressão*: Possibilita a configuração da impressora.
  - *Imprimir:* Imprime o conteúdo da janela de visualização. Esta impressão obedece as configurações realizadas pelo usuário.
  - *Sair*: Encerra a seção de trabalho com o *software* e sai do programa.

- Menu **Exibir.**
  - *Scripts*: Exibe ou oculta a área de edição de *scripts*
  - *Barra de Ferramentas | ToolsSV*: Exibe ou oculta a barra de ferramentas de visualização *chamada de ToolSV.*
  - *Eixos - X - Y - Z*: Opções que exibem ou ocultam a apresentação de um eixo do glifo que representa a origem (0,0,0) do sistema cartesiano tridimensional.

- Menu **Formatar.**
  - *Dégradé | Faixa Púrpura*: Estabelece que a faixa de cores utilizadas para a representação de gradientes é a faixa que está entre o azul e vermelho passando por todas nuanças entre elas.
  - *Primitivas | Pontos, Linhas, Faces*: Estabelece se a representação de vértices de uma superfície ou de um objeto possuirá ou não arestas, e se possuirá ou não faces em *rendering*

- Menu **Model**.
  - *Escalonar*: Permite alterar a proporção de medida de um eixo em relação aos outros. Se um fenômeno visualizado possuir em um ou dois eixos medidas muito inferiores ou superiores aos demais, podemos alterar a proporção para melhor ajustar a área de visualização.

- Menu **Projeção.**
  - *Ortogonal:* Fixa que o sistema de apresentação da visualização seja em projeção ortogonal e não em perspectiva. Todas as opções do menu Projeção que estejam entre as opções Ortogonal e Perspectiva são exclusivas da opção Ortogonal, além das opções *Zoom* e *Pan* deste menu. A opção Lente e Plano de Visualização é exclusiva à opção Perspectiva. A opção Posição de Visualização e Alvo é comum à Ortogonal e à Perspectiva.
  - *Box do Volume de Visualização*: Estabelece os valores de *clipping* esquerdo, direito, superior, inferior, frontal e posterior do volume de visualização. Desta forma, somente as partes de objetos ou objetos internos a este volume serão apresentados na visualização.
  - *Plano | XY, XZ, YZ, XY-1, XZ-1, YZ-1*: Permite a escolha da vista ortogonal a partir dos planos notáveis do sistema cartesiano tridimensional. Os planos indicados com o índice -1, representam a vista simétrica em relação ao eixo ortogonal ao plano formado pelos eixos indicados pelas letras usadas na opção.
  - *Isométrica | SW->NE, SE->NW, NE->SW, NW->SE*: Estabelece uma visualização em vista isométrica com a coordenada *y* (*UP*) sempre positiva, segundo um vetor orientação que possua 45 graus em projeção no plano *XY* em relação aos eixos *x* e *z*.
  - *Perspectiva*: Esta opção passa a projeção para a forma de perspectiva cônica e ativa a opção Lente e Plano de Visualização, a qual estabelece o volume de visualização para a perspectiva cônica a partir dos seguintes valores: Ângulo da lente (*fovy*) entre 0 e 180 graus e que informa qual é a abertura do campo de visualização. Quanto maior for este valor, mais panorâmica será a visualização, porém mais distorcida também ficará. O valor *default* é de 45graus, o qual está próximo do valor encontrado para a visão humana. Os dois próximos valores que podem ser estabelecidos são o plano de projeção e do plano posterior ou de profundidade do volume de visualização. Estes valores determinam o que será ou não visível na visualização. O valor estabelecido como padrão para o plano de projeção é 100 e para o plano posterior é -100 em relação ao eixo ortogonal aos referidos planos.
  - *Posição de Visualização e Alvo*: Esta opção determina o ponto onde está situado o observador ou câmera em relação aos objetos no espaço tridimensional. Existem três ternos ordenados que podem ser definidos: Posição da Câmera, Posição do Alvo e o Vetor Giro da Câmera. A Posição da Câmera determina onde ela se encontra no espaço tridimensional em relação aos objetos. Os valores padrões de (*x*, *y*, *z*) são 0, 0 e 100, respectivamente. Desta forma a câmera está 100 unidades sobre o eixo *z*, o qual está orientado inicialmente como para fora da tela. A Posição do Alvo determina para onde a câmera esta apontada. Os valores (*x*, *y*, *z*) padrões são 0, 0 e 0 respectivamente. Desta forma a câmera está apontada, inicialmente, para a origem do sistema. O último parâmetro é a orientação do Vetor de Giro da Câmera, o qual indica qual o ângulo de rotação da câmera em relação ao plano que é ortogonal ao vetor que passa pelos pontos de posição da câmera e posição do alvo. Os valores iniciais são 0, 1 e 0, respectivamente a *x*, *y* e *z*. Desta forma, a câmera está com o chamado "*up*" voltado para cima e temos uma imagem sem inclinações laterais. Todas estas opções também estão disponíveis em uma barra de *status* na base da janela de visualização para dar maior interatividade com o usuário. Além, delas existe um ícone com um desenho da letra grega delta na extremidade inferior direita da janela, o qual permite estabelecer um valor numérico que é o passo para a razão de incremento no avanço dos valores de coordenadas, quando utilizados os botões da barra de *status* mencionada.
  - *Zoom*: Esta opção determina a proximidade ou afastamento da câmera em relação aos objetos no espaço tridimensional. O valor passado a este parâmetro é um fator de porcentagem normalizada em relação a distância atual entre a câmera e os objetos.
  - *Pan*: Esta opção determina os deslocamentos laterais da câmera em relação aos objetos no espaço. Da mesma forma, esta opção usa uma relação de porcentagem normalizada da posição atual do vetor de observação dos objetos.

- Menu **ViSC**.
  - *Scalar | Color Map (2D pixels)*: Esta opção determina que a escolha do usuário foi para o emprego da ferramenta e técnica de visualização conhecida como *Color Map*. O MFSV tem implementada, até o momento, somente esta opção que inicializa a área de edição de *script* matemático com algumas exigências mínimas de sintaxe. Após tal inicialização pode-se estabelecer os demais requisitos da visualização segundo o(s) fenômeno(s) escolhido(s).

- Menu **Render.**
  - *Smooth (Gouraud)*: Estabelece o tipo *Smooth* para ser o tipo atual de *rendering* e, portanto, permitirão maior realismo aos objetos após o processo de *rendering*
  - *Alpha | Alpha Mínimo*: Pode-se informar o valor de intensidade do canal alfa a partir de um valor normalizado de 0 a 1. Utiliza-se o canal alfa para mascarar as regiões filtradas pelo programa MFSV no estudo de caso escolhido, para esta dissertação.
  - *Display Preview*: Esta opção mantém a janela de visualização, como uma pequena janela presa no canto inferior direito do Desktop do MFSV.
  - *Display Final*: Esta opção liberta a janela de visualização, tornando-a uma janela independente da janela principal do aplicativo. Desta forma pode-se maximizá-la ou dar algum tamanho específico para ela. Como esta janela não tem um comportamento exatamente como janela totalmente filha e nem como janela totalmente independente, foi chamada de janela adotada. Enquanto a janela está presa à janela principal, comporta-se como uma janela filha e após sua liberação, comporta-se como uma janela independente. No primeiro caso é designada como *Display Preview* e no segundo caso como *Display Final*.
  - *Resolução:* Estão disponíveis alguns tamanhos em pixels para a janela de visualização, quando a esta estiver livre, a saber: 320x200, 256x243, 512x486, 640x480, 720x486, 800x600 e 1024x768. Estes tamanhos foram escolhidos, porque estão entre os mais utilizados em aplicações *Multimídia* e saídas de imagem para TV. Além destes tamanhos preestabelecidos podem ser configurados outros pelo usuário, através da opção Outra.

- Menu **Ajuda.**
  - *Sobre*: Chama uma janela com informações sobre o software e informações de ajuda ao usuário.

ANEXO 4 - CÓDIGOS-FONTE

Os fontes apresentados aqui têm o caráter apenas ilustrativo, não sendo, muitas vezes, comentadas as suas linhas de código.

a) Arquivo fonte em linguagem LIP referente ao código usado à visualização do fenômeno escolhido como estudo de caso, e que foi convertido a partir do texto em *MathScript*.

```
real T[30][30]
real P, x, L, Tinf, To, Teta, m
real _xi, _xf, _xn, _xp, _yi, _yf, _yn, _yp, _i, _j, _cor
L    <- 0.184
Tinf <- 29.5
_xi  <- 4.55
_xf  <- 27.6
_xn  <- 30-1
_xp  <- (_xf-_xi)/_xn
_yi  <- 0
_yf  <- L
_yn  <- 30-1
_yp  <- (_yf-_yi)/_yn
_cor <- 1
_xf  <- _xf + _xp/4
_yf  <- _yf + _yp/4
REGISTRE_SV 0, 0, 0, 0  #Registro da instância e tipo de ferramenta de visualização
_i   <- -1
para P <- _xi ate _xf passo _xp
    _i   <- _i+1
    _j   <- 0
    REGISTRE_SV 0, 0, 1, _i        # registro da variável contadora dos pontos em x
    To   <- 72.01223159*ln(0.03815367*P+1)+Tinf
    Teta <- To - Tinf
    M    <- 0.5474*ln(0.9262*Teta+1) + Teta^2/(574.0444*exp(Teta^2/132.25))
            + 3.5765
    para x <- _yi ate _yf passo _yp
        REGISTRE_SV 0, 0, 2, _j    # registro da variável contadora dos pontos em y
        _cor <- 0
        se (L-acosh(cosh(L*m)*(45.0-Tinf)/Teta)/m>x  e
            x>L-acosh(cosh(L*m)*(60.0-Tinf)/Teta)/m) entao
                _cor <- 1
        fimse
        T[_i][_j] <- Tinf + Teta*cosh(m*(L-x))/cosh(m*L)
        REGISTRE_SV 0, 0, 3, _cor       # registro que filtra os valores de T[x][y]
        REGISTRE_SV 0, 0, 4, T[_i][_j]            # registro da variável principal
        _j   <- _j+1
    fimpara
fimpara
fim
```

b) Arquivo fonte cabeçalho (.h) da unidade que descreve as classes de visualização:

```
//---------------------------------------------------------------------------
#ifndef ClassesSVH
#define ClassesSVH


//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
#include "OpenGL.h"


//---------------------------------------------------------------------------


struct ptoCM
{
    char      status_cor; // referente a variável _cor do pseudocódigo
    float     pz;         // coordenada z calculada da visualização
    GLubyte   pcor[4];    // cores RGBA utilizadas para cada ponto
};


class ColorMapClass
{
  private:
    float  maior_z, menor_z, xi, xf, xp, xn, yi, yf, yp, yn;
    int    i, j;                 // informam qual o ponto no momento
    ptoCM  *ponto;             // ponteiro a ser alocado dinamicamente
  public:         // atributos públicos
    GLenum prim;               // tipo de primitiva utilizada no Color Map
                               // GL_LINE_LOOP ou GL_TRIANGLES
    bool   borda;              // aresta comum entre faces visível ou não

                  // métodos públicos
    bool        transcritor(TMemo *SMCod, TMemo *SPCod);//Scr.Matem=>Pseudo
    void        reg_faixa_cores(void);
    void        reg_cores(void);
    void        reg_var_xi(float _xi) {xi = _xi;};      // INLINE
    void        reg_var_xf(float _xf) {xf = _xf;};      // INLINE
    void        reg_var_xp(float _xp) {xp = _xp;};      // INLINE
    void        reg_var_xn(float _xn) {xn = _xn;};      // INLINE
    void        reg_var_yi(float _yi) {yi = _yi;};      // INLINE
    void        reg_var_yf(float _yf) {yf = _yf;};      // INLINE
    void        reg_var_yp(float _yp) {yp = _yp;};      // INLINE
    void        reg_var_yn(float _yn) {yn = _yn;};      // INLINE
    void        reg_ind_i(int _i)     {i  =  _i;};      // INLINE
    void        reg_ind_j(int _j)     {j  =  _j;};      // INLINE
    void        reg_status_cor(char _status_cor)        // INLINE
                    { ponto[i*(int)(xn+1) + j].status_cor = _status_cor; };
    void        reg_ponto(float zcoord);
    bool        aloca_pontos(void);
    void        desaloca_pontos(void);
    int         StrTok(TMemo *str, int indice, AnsiString strbusca);
    AnsiString  DelBrancos(AnsiString str);
    AnsiString  DelAllBrancos(AnsiString str);
    AnsiString  SubstVirgPto(AnsiString str);
    void        tabelar_fil(TMemo *SMCod);
```

```
                // métodos que retornam valores
    float       ret_var_xi(void) { return xi; }          // INLINE
    float       ret_var_xf(void) { return xf; }          // INLINE
    float       ret_var_xp(void) { return xp; }          // INLINE
    float       ret_var_xn(void) { return xn; }          // INLINE
    float       ret_var_yi(void) { return yi; }          // INLINE
    float       ret_var_yf(void) { return yf; }          // INLINE
    float       ret_var_yp(void) { return yp; }          // INLINE
    float       ret_var_yn(void) { return yn; }          // INLINE
    ptoCM       ret_ptoCM(int i, int j)
                    { return ponto[i*(int)(xn+1) + j]; } // INLINE
};
//---------------------------------------------------------------------------
extern ColorMapClass *CM; // instância de uma ferramenta Color Map na RAM
extern bool SV_CM;
//---------------------------------------------------------------------------
#endif
```

c) Arquivo C++ (.cpp) da unidade que trata as exceções em programas LIP:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "Excecoes.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)

double math_erro;
bool   DEPURACAO = false;

char *tabela_excecoes[] =
{
   "<OK>: Sem Erros!",
   "<SINTAXE>: Erro Geral de Sintaxe!",
   "<SINTAXE>: Erro de Parenteses, Chaves ou Colchetes não Balanceados!",
   "<SEMÂNTICO>: Erro por Expressão ou Código Inexistentes ou Inválidos!",
   "<ALOCAÇÃO>: Erro na Alocação de Memória!",
   "<IDENTIFICADOR>: Erro no Nome Identificador de Variável, etc.!",
   "<CRIAÇÃO DE IDENTIFICADOR>: Erro na Tentativa de Criação de Variável!",
   "<DIMENSÃO>: Erro no Tamanho de Variável na Memória. Tamanho Fora de Faixa!",
   "<ATRIBUIÇÃO>: Erro na Tentativa de Avaliação de uma Variável ou Atribuição de
                  Valor a uma Variável!",
   "<LABEL INEXISTENTE>: Erro na Chamada de Linha não Rotulada!",
   "<LABEL REPETIDO>: Erro em Código com Mais de Uma Ocorrência do Mesmo Rótulo!",
   "<OVERFLOW DE PILHA>: Erro pela Ultrapassagem do Limite Superior de Pilha!",
   "<UNDERFLOW DE PILHA>: Erro pela Ultrapassagem do Limite Inferior de Pilha!",
   "<EXTENSÃO INVÁLIDA>: Erro no Nome de Arquivo. Extensão Inválida!",
   "<CRIAÇÃO DE ARQUIVO>: Erro na Tentativa de Criar um Arquivo!",
   "<ABERTURA DE ARQUIVO>: Erro na Tentativa de Abrir um Arquivo!",
   "<CABEÇALHO EM SCRIPT MATEMÁTICO>: Erro no Cabeçalho do Script!",
   "<PARAMETRIZAÇÃO INCOMPLETA>: Erro na Quantidade de Arqgumentos!",
   "<REGISTRO DE VISUALIZAÇÃO>: Erro pela Falta de Registro de Dados!"
};

// mostra uma mensagem de erro conforme um codigo passado
void mens_erro(int erro, char *prompt)
{
   if(erro)
      Application->MessageBox(tabela_excecoes[erro], prompt, MB_OK);
   else
      Application->MessageBox(tabela_excecoes[erro], "Sem Erros", MB_DEFBUTTON1);
}
```

d) Arquivo cabeçalho (.h) das classes da unidade que trata dos identificadores declarados pelo usuário em código escrito em LIP.

```
#ifndef arv_avl2H
#define arv_avl2H
/*
  idxvar e calculado com base no nome (string) da variável.
  Sao considerados somente os 4 (quatro) primeiros caracteres.
  Assim: 1o. caractere na faixa de A..Z, a..z e o caractere de '_'
                        (53 possiveis simbolos)
         2o., 3o. e 4o. caracteres na faixa de A..Z, a..z,'_' e 0..9
                        (63 possíveis caracteres por posicao)
         Temos um total de mais de 53*63*63*63 = 13.252.491 indices
         de identificadores, sendo que uma matriz ocupa um único índice
         sao mais de 13.252.491, porque podemos ter variáveis usando
         somente um caractere, ou somente dois, ou somente três, ou todos
         os caracteres da string
  exemplos:
  ABE1 = 1o. = 65, 2o. = 66, 3o. = 69 e 4o. = 49
*/

#define  STRINGT    0x00 // bin = 0000 0000       não implementada
#define  DOUBLE       0x01 // bin = 0000 0001
#define  MXSTRING   0x80 // bin = 1000 0000       não implementada
#define  MXDOUBLE     0x81 // bin = 1000 0001

// definicao de tipo
typedef unsigned char UCHAR;
typedef unsigned int  ULINT;

class uvars // CLASSE QUE CONTROLA UM NÓ NA ÁRVORE
{
  private: // atributos privados
      ULINT idxvar;   //indice da variavel e chave de busca da arvore
      void  *p_dado;  //valor da variavel
      ULINT *imatr;   //tabela de fatores. Conversao de índices de matrizes
      ULINT dimensao; //quantas dimensoes tem a matriz da esquerdap/direita
      ULINT tam;      // tamanho em bytes do elemento na memoria
      UCHAR tipo_var; //xyyyyyy0 = string onde x: x=0 singular ou 1=matriz
                      // os bits y nao importam
                      //xyyyyyy1=numerica onde y=(0 ou 1) e x idem anterior
                      // os bits y permitem subtipos (int,real simples,etc)
                      //x0000001=double em linguagem C
                      // exemplos: 00000001 = numero double simples
                      //           10000001 = matriz de doubles
                      //           00000011 = (poderia ser) float simples
                      //           10000011 = (poderia ser) matriz de floats
  public: // serviços públicos
      int   config_var(void *, UCHAR, ULINT, UCHAR *);
      void  *retorna_var(void);
      ULINT retorna_idx(void);
      UCHAR retorna_tipo(void)      { return tipo_var; };
      ULINT retorna_tam(void)       { return tam;      };
      ULINT retorna_dim(void)       { return dimensao; };
      ULINT retorna_imatr(ULINT i) { return *(imatr+i); };
      void  atribui_var(char *, ULINT);
     // void  mens_erro(int);
      bool  copie(uvars *, uvars *, ULINT, ULINT *);
      bool  salvar_var(uvars **, uvars *, ULINT ndi=0, ULINT *di=NULL);
      void  desaloque_ptr_dimensoes(void);
      // funções auxiliares de uso da classe arv
      void  remove_no(uvars *);
      int   compara(uvars *, uvars *);
      // bool  alguma_funcao_trav(arv_t); // ativar quando necessária

};
```

```
#define arv_t uvars *

class arv // CLASSE QUE CONTROLA A ÁRVORE UTILIZADA PARA MAPEAR IDENTIFICADORES
{
  // atributos privados
      arv           *arv_l, *arv_r;
      short         arv_b;
      arv_t         arv_p;
      //  void  *tag; // campo de livre uso (usado para o ptr de um campo p_dado em
arv_t)

  // servicos privados
      bool  arv_ins_balanceia(arv **, arv_t, bool *);
      int   arv_del_balanceia(arv **, arv_t, bool *, bool *);
      void  arv_del_balanceia_aux(arv **, bool *, arv **, bool *);
      void  arv_bal_L(arv **, bool *);
      void  arv_bal_R(arv **, bool *);
      // ponteiros de servicos privados e auxiliares

  public: // servicos publicos
      arv(void);  // construtor da classe árvore
      ~arv(void); // destrutor  da classe árvore

      arv   *retorna_l(arv *);
      arv   *retorna_r(arv *);
      arv_t  retorna_p(arv *);

      void   arv_init(arv **);
      arv_t  arv_busca(arv **, arv_t);
      bool   arv_inclui(arv **, arv_t);
      int    arv_elimina(arv **, arv_t);
      void   arv_desaloca_arvore(arv **);
      //bool arv_trav(arv **);  // ativar quando necessário
};
#endif
```

e) Arquivo cabeçalho (.h) da classe de formulário da unidade que controla as apresentações na janela de visualização

```
//---------------------------------------------------------------------------
#ifndef TelaSaidaVisH
#define TelaSaidaVisH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Menus.hpp>
#include "mwajpeg.hpp"
#include <ExtCtrls.hpp>
#include "OpenGL.h"
#include <Dialogs.hpp>
#include <Buttons.hpp>
#include <ComCtrls.hpp>
#include <ToolWin.hpp>
//---------------------------------------------------------------------------
class TfrmVisualizacao : public TForm
{
__published:        // IDE-managed Components
    TPopupMenu *mnuPopImagem;
    TMenuItem  *mnuPopAbrirImagem;
    TMenuItem  *mnuPopSalvarImagem;
    TMenuItem  *N1;
```

```
TMenuItem  *mnuPopImprimirImagem;
TMenuItem  *N2;
TMenuItem  *mnuPopSairdaVisualizacao;
TMenuItem  *mnuPopFecharImagem;
TMenuItem  *mnuPopPrinterSetup;
TMenuItem  *mnuEscaladoFramebuffer;
TMenuItem  *mnuPopProporcional;
TMenuItem  *mnuPopDesformatada;
TMenuItem  *mnuPopPreencherPagina;
TMenuItem  *mnuPopRenderizar;
TImage     *PopImagem;
TImage     *icone;
TOpenDialog *PopAbrirDialogo;
TSaveDialog *PopSalvaDialogo;
TPrintDialog *PopPrint;
TPrinterSetupDialog *PopPrinterSetup;
TToolBar *BarradeTitulo;
TStaticText *Titulo;
TSplitter *Splitter1;
TSplitter *Splitter2;
TSplitter *Splitter3;
TTimer *Timer1;

TOpenGL *OpenGL1;              // OBJETO QUE REPRESENTA A JANELA DE VISUALIZAÇÃO

TPanel *Panel1;
TEdit *edtX;
TEdit *edtY;
TEdit *edtZ;
TGroupBox *GroupBox1;
TBitBtn *edtXc;
TBitBtn *edtYc;
TBitBtn *edtZc;
TGroupBox *GroupBox2;
TBitBtn *edtXa;
TBitBtn *edtYa;
TBitBtn *edtZa;
TGroupBox *GroupBox3;
TBitBtn *edtXup;
TBitBtn *edtYup;
TBitBtn *edtZup;
TBitBtn *edtDelta;
void __fastcall mnuPopSalvarImagemClick(TObject *Sender);
void __fastcall mnuPopAbrirImagemClick(TObject *Sender);
void __fastcall mnuPopFecharImagemClick(TObject *Sender);
void __fastcall mnuPopSairdaVisualizacaoClick(TObject *Sender);
void __fastcall mnuPopImprimirImagemClick(TObject *Sender);
void __fastcall mnuPopPrinterSetupClick(TObject *Sender);
void __fastcall mnuPopProporcionalClick(TObject *Sender);
void __fastcall mnuPopDesformatadaClick(TObject *Sender);
void __fastcall mnuPopPreencherPaginaClick(TObject *Sender);
void __fastcall FormActivate(TObject *Sender);
void __fastcall FormResize(TObject *Sender);
void __fastcall FormClick(TObject *Sender);
void __fastcall FormPaint(TObject *Sender);
void __fastcall Timer1Timer(TObject *Sender);
void __fastcall FormClose(TObject *Sender, TCloseAction &Action);
void __fastcall TituloDblClick(TObject *Sender);
void __fastcall edtXEnter(TObject *Sender);
void __fastcall edtXcKeyUp(TObject *Sender, WORD &Key, TShiftState Shift);
void __fastcall edtYcKeyUp(TObject *Sender, WORD &Key, TShiftState Shift);
void __fastcall edtZcKeyUp(TObject *Sender, WORD &Key, TShiftState Shift);
void __fastcall edtXaKeyUp(TObject *Sender, WORD &Key, TShiftState Shift);
void __fastcall edtYaKeyUp(TObject *Sender, WORD &Key, TShiftState Shift);
void __fastcall edtZaKeyUp(TObject *Sender, WORD &Key, TShiftState Shift);
void __fastcall edtXClick(TObject *Sender);
void __fastcall edtYClick(TObject *Sender);
```

```
    void __fastcall edtZClick(TObject *Sender);
    void __fastcall edtZupKeyUp(TObject *Sender, WORD &Key, TShiftState Shift);
    void __fastcall edtXupKeyUp(TObject *Sender, WORD &Key, TShiftState Shift);
    void __fastcall edtYupKeyUp(TObject *Sender, WORD &Key, TShiftState Shift);
    void __fastcall edtXcClick(TObject *Sender);
    void __fastcall edtXaClick(TObject *Sender);
    void __fastcall edtXupClick(TObject *Sender);
    void __fastcall edtDeltaClick(TObject *Sender);

    // eventos do componente OpenGL
    void __fastcall OpenGL1GLInit(TObject *Sender);  // MÉTODO APRESENTADO A SEGUIR
    void __fastcall OpenGL1GLPaint(TObject *Sender); // MÉTODO APRESENTADO A SEGUIR
    void __fastcall OpenGL1Resize(TObject *Sender);  // MÉTODO APRESENTADO A SEGUIR
    void __fastcall OpenGL1Click(TObject *Sender);   // MÉTODO APRESENTADO A SEGUIR

private:            // User declarations
public:                                 // User declarations
    __fastcall TfrmVisualizacao(TComponent* Owner);
    void __fastcall ChamaPopAbrirImagem(TObject *Sender);
    void __fastcall ChamaPopImprimirImagem(TObject *Sender);
    void __fastcall ChamaPopPrinterSetup(TObject *Sender);
    // serviços relacionados ao componente OpenGL OpenGL1
    void __fastcall DisplayObjetos(void);           // MÉTODO APRESENTADO A SEGUIR
};
//---------------------------------------------------------------------------
extern PACKAGE TfrmVisualizacao *frmVisualizacao;
extern GLdouble xc,yc,zc, xa,ya,za, xup,yup,zup;
extern GLfloat  xleft,xright, ybottom,ytop, znear,zfar;
extern GLfloat  pfovy, pznear,pzfar;
extern GLfloat  x_scale, y_scale, z_scale;
//---------------------------------------------------------------------------
#endif
```

f) Trecho do arquivo C++ (.cpp) que apresenta o método `DisplayObjetos()`, o qual é membro da classe de formulário da unidade que controla as apresentações na janela de visualização. Este método foi criado segundo um algoritmo para tecer uma malha de pontos em um *grid* estabelecido para o *Color Map*.

```
void __fastcall TfrmVisualizacao::DisplayObjetos(void)
{
  if(CM) // existe uma instância de Color Map
  {      // cria a malha do Color Map
         // CM->ret_ptoCM(x,y); // acesso a um ponto do Color Map

     int   i, j, xn, yn;
     float x, y, dx, dy;

     // captura dados do objeto da classe visualizadora
     if(Aval->mnuMalhaPontos->Checked) CM->prim = GL_POINTS;
     if(Aval->mnuMalhaLinhas->Checked) CM->prim = GL_LINE_LOOP;
     if(Aval->mnuMalhaFaces->Checked)  CM->prim = GL_TRIANGLES;
     CM->borda = Aval->mnuBordaCompartilhadaAparente->Checked;
     x  = CM->ret_var_xi();      // marca valor inicial para o eixo dos x
     xn = (int)CM->ret_var_xn(); // marca quantidade de pontos em x
     dx = CM->ret_var_xp();      // taxa de variação em x
     yn = (int)CM->ret_var_yn(); // marca quantidade de pontos em y
     dy = CM->ret_var_yp();      // taxa de variação em y
```

```
    for(i=0; i<xn; i++) // cria a superfície a ser visualizada - todos os pontos
    {
       y  = CM->ret_var_yi();   // marca valor inicial para o eixo dos y
       for(j=0; j<yn; j++)
       {
           // faz a face inferior esquerda (GL_TRIANGLES) ou faz quadrilátero
           // ciclo de construção sentido antihorário
           glBegin(CM->prim);
               // primeiro ponto da face inferior em um plano xy
               // ponto inferior esquerdo em um plano xy
               glColor4ubv(CM->ret_ptoCM(i, j+1).pcor);                  // COR
               glVertex3f(x, y+dy, CM->ret_ptoCM(i, j+1).pz);            // PONTO
               // segundo ponto da face inferior em um plano xy
               // ponto superior esquerdo em um plano xy
               glColor4ubv(CM->ret_ptoCM(i, j).pcor);                    // COR
               glVertex3f(x, y, CM->ret_ptoCM(i, j).pz);                 // PONTO
               if(CM->prim == GL_TRIANGLES)
                   if(CM->borda) glEdgeFlag(GL_TRUE);
                   else          glEdgeFlag(GL_FALSE);
               if(CM->prim==GL_LINE_LOOP) // quadrilátero, mais um vértice
               {   // terceiro ponto da face de um quadrilátero
                   // ponto superior direito em um plano xy
                   glColor4ubv(CM->ret_ptoCM(i+1, j).pcor);              // COR
                   glVertex3f(x+dx, y, CM->ret_ptoCM(i+1, j).pz);        // PONTO
               }
               // terceiro ponto se face triangular inferior em um plano xy ou
                  // quarto ponto se face de um quadrilátero
               // ponto inferior direito em um plano xy
               glColor4ubv(CM->ret_ptoCM(i+1, j+1).pcor);                // COR
               glVertex3f(x+dx, y+dy, CM->ret_ptoCM(i+1, j+1).pz);       // PONTO
               if(CM->prim == GL_TRIANGLES) glEdgeFlag(GL_TRUE);
           glEnd();
           // faz outro triangulo (se GL_TRIANGLES) para completar um quadril.
           if(CM->prim!=GL_LINE_LOOP)
           {
               // ciclo de construção sentido antihorário
               glBegin(CM->prim);
                   // primeiro ponto da face superior em um plano xy
                   glColor4ubv(CM->ret_ptoCM(i+1, j).pcor);              // COR
                   glVertex3f(x+dx, y, CM->ret_ptoCM(i+1, j).pz);        // PONTO
                   // segundo ponto da face superior em um plano xy
                   glColor4ubv(CM->ret_ptoCM(i+1, j+1).pcor);            // COR
                   glVertex3f(x+dx, y+dy, CM->ret_ptoCM(i+1, j+1).pz);   // PONTO
                   if(CM->prim == GL_TRIANGLES)
                       if(CM->borda) glEdgeFlag(GL_TRUE);
                       else          glEdgeFlag(GL_FALSE);
                   // terceiro ponto da face superior em um plano xy
                   glColor4ubv(CM->ret_ptoCM(i, j).pcor);                // COR
                   glVertex3f(x, y, CM->ret_ptoCM(i, j).pz);             // PONTO
                   if(CM->prim == GL_TRIANGLES) glEdgeFlag(GL_TRUE);
               glEnd();
           }
           y += dy;
       }
       x += dx;
    }
  }
}
```

g) Trecho do arquivo C++ (.cpp) que apresenta o método `OpenGL1GLInit()`, o qual é membro da classe de formulário da unidade que controla as apresentações na janela de visualização.

```
void __fastcall TfrmVisualizacao::OpenGL1GLInit(TObject *Sender)
{
   glEnable(GL_BLEND);
   glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
   glShadeModel(GL_SMOOTH);    // estilo de renderização inicial
}
//---------------------------------------------------------------------------
```

h) Trecho do arquivo C++ (.cpp) que apresenta o método `OpenGL1GLPaint()`, o qual é membro da classe de formulário da unidade que controla as apresentações na janela de visualização.

```
void __fastcall TfrmVisualizacao::OpenGL1GLPaint(TObject *Sender)
{
   GLfloat dist_xyz=30.0; //sqrt((xc-xa)*(xc-xa)+(yc-ya)*(yc-ya)+(zc-za)*(zc-za));
   glClear(GL_COLOR_BUFFER_BIT);
   glLoadIdentity();
   gluLookAt(xc,yc,zc, xa,ya,za, xup,yup,zup);

   // plota se necessário os eixos x, y, z
   if(Aval->mnuXONOFF->Checked)
   {
     glBegin(GL_LINES);
        glColor3f(1.0, 0.0, 0.0); // X  - RED
        glVertex3f(0.0, 0.0, 0.0);
        glVertex3f(dist_xyz, 0.0, 0.0);
     glEnd();
   }
   if(Aval->mnuYONOFF->Checked)
   {
     glBegin(GL_LINES);
        glColor3f(0.0, 1.0, 0.0); // Y  - GREEN
        glVertex3f(0.0, 0.0, 0.0);
        glVertex3f(0.0, dist_xyz, 0.0);
     glEnd();
   }
   if(Aval->mnuZONOFF->Checked)
   {
     glBegin(GL_LINES);
       glColor3f(0.0, 0.0, 1.0);  // Z  - BLUE
       glVertex3f(0.0, 0.0, 0.0);
       glVertex3f(0.0, 0.0, dist_xyz);
     glEnd();
   }
   glScalef(x_scale, y_scale, z_scale);
   DisplayObjetos();
   glFlush();
}
//---------------------------------------------------------------------------
```

i) Trecho do arquivo C++ (.cpp) que apresenta o método `OpenGL1Resize()`, o qual é membro da classe de formulário da unidade que controla as apresentações na janela de visualização.

```
void __fastcall TfrmVisualizacao::OpenGL1Resize(TObject *Sender)
{
   glViewport(0, 0, OpenGL1->Width, OpenGL1->Height);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   if(Aval->mnuOrtogonal->Checked)            // vista ortogonal
   {
       GLfloat aspecto = 1.0; //(GLfloat)OpenGL1->Width/(GLfloat) OpenGL1->Height;
       if (aspecto <= 1.0)
       {
          glOrtho(xleft, xright, ybottom, ytop/aspecto, znear, zfar);
       }
       else
       {
          glOrtho(xleft, xright*aspecto, ybottom, ytop, znear, zfar);
       }
   }
   else                                       // perspectiva
   {
       GLfloat aspecto = (GLfloat)OpenGL1->Width/(GLfloat) OpenGL1->Height;
       gluPerspective(pfovy, aspecto, pznear, pzfar);
   }
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}
```

j) Trecho do arquivo C++ (.cpp) que apresenta o método `OpenGL1Click()`, o qual é membro da classe de formulário da unidade que controla as apresentações na janela de visualização.

```
//---------------------------------------------------------------------------
void __fastcall TfrmVisualizacao::OpenGL1Click(TObject *Sender)
{
   frmVisualizacao->Hide();
   frmVisualizacao->Show();
}
```

REFERÊNCIAS

1 BEJAN, Adrian. **Transferência de calor**. São Paulo : Editora Edgar Blücher Ltda, 1996.

2 BESANT, C.B. **CAD/CAM** : projeto e fabricação com o auxílio do computador. Rio de Janeiro : Editora Campus Ltda, 1985.

3 BRODLIE, K. W.; CARPENTER, L. A.; EARNSHAW, R.A. et alli. (Eds.). **Scientific visualization** : Techniques and Applications. Berlin : Springer-Verlag, 1992.

4 COHOON, James P.; DAVIDSON, Jack W. **C++ Program Design : An Introduction to Programming and Object-Oriented Design.** 2nd. ed. Virginia : McGraw Hill, 1999

5 COLAÇO, Marcelo J.; ORLANDE, Helcio R. B. Transient heat transfer of a diesel engine piston. **1° Encontro de Usuários do NACAD-COPPE/UFRJ**, Rio de Janeiro, 1996.

6 _____; COTTA, Renato M. Software para estudo de difusão de calor em geometrias irregulares usando diferenças finitas. **Anais do II Congresso Regional de Estudantes de Engenharia Mecânica**, UFRJ, Rio de Janeiro, p. 317-322, 1995.

7 CONNER, D.B.; SNIBBE, S.S.; HERNDON,D.C. et al. Three-Dimensional Widgets, **Proceeding of the 1992 Workshop on Interative 3D Graphics**, pp. 183-188, 1992

8 EARNSHAW, Rae.A.; WATSON, David. **Animation and scientific visualization** : tools & applications. London : Academic Press Limited, 1993.

9 FONSECA, L.T.; REIS, L.P.; MARTHA,L.F. Uma arquitetura para construção de ferramentas de manipulação para visualização interativa de dados volumétricos, X SIMPÓSIO BRASILEIRO DE COMPUTAÇÃO GRÁFICA E PROCESSAMENTO DE IMAGENS - SIBGRAPI'97, Campos do Jordão, 1997.

10 FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. **Lógica de programação** : a construção de algoritmos e estruturas de dados. 2 ed. Curitiba : Makron Books do Brasil Editora Ltda, 2000.

11 FRANK, K.; LANG, U. Data-dependent surface simplification. **Eurographics Workshop on Visualization in Scientific Computing**, Blaubeuren, 1998.

12 FREITAS, Carla Maria Dal Sasso; Methodology for Selecting Visual Representations in Scientific and Simulation Applications. **SIBGRAPI'93 VI Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens**, p. 89-97, Recife. Editores Luiz Henrique de Figueiredo e Jonas de Miranda Gomes, 1993.

13 FREITAS, Carla Maria Dal Sasso; GEUS, Klaus; MARTINEZ, Maria Laura et alli. **Tutorial visualização científica.** Curitiba : SIBGRAPI'94. VII Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens, 1994.

14 GOMES, Jonas; VELHO, Luiz. **Computação Gráfica** : imagem. Rio de Janeiro : Instituto de Matemática Pura e Aplicada - IMPA, 1994.

15 HETZEL, Willian. **Guia completo ao teste de software**. Rio de Janeiro : Editora Campus Ltda, 1987.

16 HILL, Francis S., Jr.; HILL Francis J. **Computer graphics using OpenGL**. 2nd. ed. New Jersey : Prentice-Hall Inc., 2000.

17 INCROPERA, Frank P.; DeWITT, David P. **Fundamentos de transferência de calor**. 3. ed. Rio de Janeiro, RJ : LTC Livros Técnicos e Científicos Editora S.A., 1990.

18 JACOBS, Jon Q. **Delphi developer's guide to OpenGL**. Texas : Wordware Publishing Inc., 1999.

19 JOSÉ, João, Neto. **Introdução à compilação**. Rio de Janeiro : Livros Técnicos e Científicos Editora S.A., 1987.

20 JUCÁ, P.C.S.; PRATA, A.T. Transferência de Calor em Superfícies Aletadas: Influência do Perfil de Temperatura na Base da Aleta, ANAIS DO 1° SIMMEC, pp. 193-200, Belo Horizonte, 1991.

21 KEMPF, Renate; FRAZIER, Chris. **Opengl reference manual** : the official reference document for OpenGL, Version 1.1, 2nd. ed. Massachusetts : Addison Wesley Longman, Inc., 1997.

22 KERNIGHAN, Brian W.; RITCHIE, Dennis M. C a linguagem de programação. Rio de Janeiro : Editora Campus Ltda, 1986

23 KOCHHAR, Sandeep; FRIEDELL, Mark; SISTARE, Steve; JUDA, Janusz; LAPOLLA, Mark; MARKS, Joe; MCMURRY, Peter; KOSAK, Corey; SHIEBER, Stuart. A design environment for scientific and program visualization. **COMPUGRAPHICS'91**, p. 321-332, 1991.

24 KOTAKE, Susumu; HIJAKATA, Kunio. **Numerical simulations of heat transfer and fluid flow on a personal computer**. Amsterdam : Elsevier Science Publishers B.V., 1993.

25 LEVOY, Marc. Spreadsheets for images. **Proceedings of SIGGRAPH 94, Computer Graphics Proceedings, Annual Conference Series**. p. 139-146, Orlando. ACM Press. Edited by Andrew Glassner,1994.

26 MALISKA, C. R.; REIS, M. V. F.; MALISKA JUNIOR, C. R.; DIHLMANN, A. Heat Transfer 1.0 – An Educational Software for Heat Conduction Teaching, ASME PROCEEDINGS OF THE 32ND – NATIONAL HEAT TRANSFER CONFERENCE, vol. 6, pp. 53-59, Baltimore, 1997.

27 _____. Classroom Experiences with Heat Transfer 1.0 – A Software for Heat Conduction Teaching, INTERNATIONAL MECHANICAL ENGINEERING CONGRESS & EXPOSITION, IMECE 98, APPROVED FOR PRESENTATION, November 15-20, 1998, Anaheim, USA.

28 _____; CABRAL, R.B. Heat Transfer 1.0 – A Software for Heat Conduction Teaching-Computational Structure and Class-Room Experiences, PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON ENGINEERING EDUCATION – ICEE 98, Rio de Janeiro, 1998.

29 MALISKA JUNIOR,C.R.; DIHLMANN. A. ISO3D - Visualizador Tridimensional Para Campos Escalares e Vetoriais, ANAIS DO XII CILAMCE - CONGRESSO IBERO LATINO AMERICANO SOBRE MÉTODOS COMPUTACIONAIS PARA A ENGENHARIA, pp. 354 - 363, Porto Alegre, 1992.

30 McCORMICK, B. H.; DeFANTI, T. A.; BROWN, M.D. Visualization scientific computing. **Computer Graphics**, v. 21, n.6, p. 1-5, 1987.

31 MORAN, Patrick J.; HENZE, Chris. Large field visualization with demand-driven calculation. **IEEE Visualization '99**, p. 27-34, San Francisco. IEEE. Edited by David Ebert and Markus Gross and Bernd Hamann, 1999.

32 NISHIMURA, Tatsuo; FUJIWARE, Masaki; MIYASHITA, Hisashi. Visualization of temperature fields of transient natural convection with maximum density effect in a water-filled enclosure by chiral nematic liquid crystals. **Journal of Chemical Engineering**, v. 23, n. 2, p. 241-244, Toyama, 1990.

33 REISDORPH, Kent. **Teach yourself Borland C++Builder 3 in 21 days**. Indianápolis : Sams Publishing, 1998.

34 _____. **Borland C++Builder 4 unleashed** : the comprehensive solutions. Indianápolis, IN : Sams Publishing, 1999.

35 ROSENBLUM, Lawrence J.; EARNSHAW, Rae A.; ENCARNAÇÃO, José L. et alli. **Scientific visualization** : advances and challenges. London : Academic Press Limited, 1994.

36 RUMPF, Martin; SCHMIDT, Alfred; SIEBERT, Kunibert G. Functions defining arbitrary meshes : A flexible interface between numerical data and visualization. **Computer Graphics Forum**, v. 15, n. 2, p. 129-142, 1996.

37 SCHILDT, Herbert. **C: power user's guide**. Berkeley : McGraw-Hill, 1988.

38 TRUJILLO, Stan. **Tudo o que você precisa saber para programar em Direct 3D**. Rio de Janeiro : Editora Ciência Moderna, 1997.

39 VILLAS, Marcos V.; FERREIRA, Andréa G. M.; LEROY, Patrick G. et alli. **Estruturas de Dados** : conceitos e técnicas de implementação. Rio de Janeiro : Editora Campus Ltda, 1993.

40 WAGNER, T.; BERGERON, R. D. A model and a system for data-parallel visualization. **Proceeding of IEEE Visualization 95**, Atlanta, p. 224-231, 1995.

41 WIRTH, Niklaus. **Algorithms & data structures**. Prentice-Hall, 1986.

42 WOO, Mason; NEIDER, Jackie; DAVIS, Tom; SHREINER, Dave. **OpenGL programming guide** : the offcial guide to learning OpenGL, version 1.2, 3th. ed. Massachusetts : Addison Weslay Longman, Inc., 1999.

43 ZACHARY, Joseph. **Introduction to scientific programming** : computational problem solving using maple and C. Berlin : Springer-Verlag, 1996.