RENATO CESAR POMPEU

SIMULAÇÃO NUMÉRICA E VISUALIZAÇÃO 3D INTERATIVA DE OBJETOS SOB FLUXOS IRROTACIONAIS EM TEMPO QUASE-REAL

CURITIBA

2010

RENATO CESAR POMPEU

SIMULAÇÃO NUMÉRICA E VISUALIZAÇÃO 3D INTERATIVA DE OBJETOS SOB FLUXOS IRROTACIONAIS EM TEMPO QUASE-REAL

Tese apresentada como requisito a obtenção do grau de Doutor em Ciências. Programa de Pós-Graduação em Métodos Numéricos em Engenharia – Mecânica Computacional, Universidade Federal do Paraná.

Orientador: Prof. Dr. Sergio Scheer

CURITIBA

2010

TERMO DE APROVAÇÃO

Renato Cesar Pompeu

"Simulação Numérica e Visualização 3D Interativa de Objetos sob Fluxos Irrotacionais em Tempo quase-real."

Tese aprovada como requisito parcial para obtenção do grau de Doutor no Programa de Pós-Graduação em Métodos Numéricos em Engenharia – Área de Concentração em Mecânica Computacional, Setores de Tecnologia e de Ciências Exatas da Universidade Federal do Paraná, pela seguinte banca examinadora:

Orientador:

Prof. Sergio Scheer, D. Sc.
Programa de Pós-Graduação em Métodos Numéricos em Engenharia - PPGMNE da UFPR

Prof. Tobias Bleninger, Dr.
Programa de Pós-Graduação em Engenharia de Recursos Hídricos e Ambiental – PPGERHA da UFPR

Prof. Maurício Felga Gobbi, Ph.D.
Programa de Pós-Graduação em Métodos Numéricos em Engenharia - PPGMNE da UFPR

Prof. Klaus de Geus, D.Sc.
Programa de Pós-Graduação em Métodos Numéricos em Engenharia PPGMNE da UFPR

Profª. Cinthia Obladen de Freitas, Dr.
Programa de Pós-Graduação em Informática Aplicada - PPGIA da PUC/PR

Curitiba, 22 de setembro de 2010.

Aos meus pais, Alberto e Meiry, professores,
pelo constante incentivo ao estudo e
verdadeira referência de vida.

AGRADECIMENTOS

A Deus, pela vida e saúde.

À minha esposa Hayl, meus filhos Camille e George, meus enteados Vânia e João Guilherme, pela paciência e apoio.

Ao meu orientador e mentor, Professor Sergio Scheer, por toda a dedicação, compreensão, orientação, paciência, confiança, apoio e amizade durante todo o período de mestrado e doutorado.

A todos os professores do programa, em especial ao Professor Roberto Dalledone Machado, por ter me acolhido nos primeiros dias, ainda no mestrado, e pelo incentivo constante.

À Maristela Bandil, pela amizade, competência e alto astral que reflete em todo CESEC.

Às amizades que fiz durante todo tempo que permaneci no PPGMNE.

Aos membros da banca de qualificação e defesa da Tese, Cinthia Freitas, Maurício Gobbi, Klaus de Geus e Tobias Bleninger, pelas sugestões decisivas e correções necessárias.

À UTFPR, pelo programa de liberação docente a pós-graduação.

RESUMO

De uma maneira geral, qualquer fluxo irrotacional e incompressível é governado pela equação de Laplace. Esta não possui resolução analítica para problemas reais de engenharia, os quais possuem domínios e condições de contorno complexas, exceto para poucos casos particulares. A Dinâmica dos Fluidos Computacional (DFC) é um método utilizado para resolver numericamente a equação de Laplace, satisfazendo condições iniciais e de contorno. Porém, ao se refinar ou estender um domínio calculado, a quantidade de dados numéricos resultantes aumentará proporcionalmente e a análise destes valores pode se tornar complexa e onerosa. Complementariamente, para a compreensão dos resultados, é importante uma representação visual.

A resolução numérica da equação de Laplace está descrita neste trabalho, com um algoritmo de solução inédito para as condições de contorno que atende qualquer forma geométrica em três dimensões. Desenvolveu-se um simulador que possibilita alterações geométricas de objetos 3D, calcula e visualiza interativamente velocidades, linhas de fluxo e força de sustentação para fluxos irrotacionais e incompressíveis em tempo quase-real. O sistema utiliza o método das diferenças finitas para a solução das equações. A interface gráfica foi desenvolvida utilizando, deste modo ineditamente para a DFC, a linguagem C++ e o VTK (Visualization Tool Kit). A quantidade, a origem das linhas de fluxo, a seleção do campo de velocidades, o cálculo da força de sustentação e a visualização estereoscópica são parâmetros que podem ser ajustados e selecionados para a visualização. O algoritmo passou por validações mostrando a capacidade de resolução em três dimensões. Assim, o simulador desenvolvido resolve, ao contrário dos softwares já existentes, o problema do cálculo e visualização interativa imediata ao se fazer modificações em objetos 3D. Este procedimento permitirá que se façam comparações entre formas geométricas imediatamente alteradas para que se possa escolher, entre elas, a que se adequar melhor às necessidades de um projeto.

Palavras chave: *Dinâmica dos Fluidos Computacional, Diferenças Finitas, VTK, 3D.*

ABSTRACT

In general, any irrotational and incompressible flow is governed by Laplace equation. This has no analytical solution to real problems of engineering, which have domains and complex boundary conditions, except for few special cases. The Computational Fluid Dynamics (CFD) is a method for numerically solving the Laplace equation satisfying the initial and boundary conditions. However, when we refine or extend a calculated field, the amount of numerical data resulting increase proportionally and the analysis of these values can become complex and costly. Complementarily, to the understanding of the results is important a visual representation.

The numerical solution of Laplace equation is described in this work with a novel algorithm for solving the boundary conditions that meet any geometric shape in three dimensions. It was developed a simulator that allows geometric alterations on 3D objects, calculates and displays interactively speeds, flow lines and lift to irrotacionais and incompressible flows in quasi-real time. The system uses the finite difference method for solving equations. The graphical interface was developed using, this way unprecedentedly for CFD, C++ and VTK (Visualization Tool Kit). The amount, source of flow lines, the selection of the velocity field, the calculation of lift and stereoscopic viewing are parameters that can be adjusted and selected for viewing. The algorithm has undergone validation showing the resolution capability in three dimensions. Thus, the developed simulator resolves, unlike the existing software, the problem of immediate interactive computation and visualization when making changes in 3D objects. This procedure will allow to make comparisons between geometric shapes changed immediately, to choose among them, the one that best fit the needs of a project

***Key words**: Computational Fluid Dynamics, Finite difference, VTK, 3D.*

Lista de Abreviaturas e Siglas

2D – *Duas dimensões*

3D – *Três dimensões*

CAD – *Computer-aided design* – Desenho auxiliado por computador

CFD – *Computational Fluid Dynamics* – Dinâmica dos fluidos computacional

Cg - *C for Graphics*

CPU – *Central Processor Unit* – Unidade central de processamento

CUDA – *Compute Unifiede Device Architecture* - Arquitetura de computação paralela

DFC – *Dinâmica dos Fluidos Computacional*

ELT – *Erro local de truncamento*

GPU – *Graphic Processor Unit* – Unidade de processamento gráfico

LIC - *Line Integral Convolution* – Convolução da integral de linha

MDF – *Método das Diferenças Finitas*

NASA – *National Aeronautics and Space Administration* – Administração Nacional do Espaço e da Aeronáutica

OLIC - *Oriented Line Integral Convolution* - Convolução da integral de linha orientada

PLIC - *Pseudo Line Integral Convolution* - Pseudo convolução da integral de linha

PVC – *Problema de Valor de Contorno*

PVI – *Problema de Valor Inicial*

RA – *Resultante Aerodinâmica*

UFLIC – *Unsteady Flow Line Integral Convolution* - Convolução da integral de linha de fluxos permanentes

VTK – *Visualization Tool Kit* - Pacote de ferramentas de visualização

Lista de Figuras

Figura 1.1 – Screenshots do simulador desenvolvido neste trabalho. O ponto em vermelho é a referência fixa escolhida como parâmetro sendo que a velocidade *Vx* varia devido as alterações na forma geométrica. 15
Figura 1.2 Modelo de visualização apresentado pelo software SolidWorks. Todo o cálculo é feito antes da visualização (fonte: Solid, 2010). 16
Figura 2.1 – Exemplos de Visualização Volumétrica: Iso-superfície, Rendering de Volume e Rede de Vetores. (fonte: Earnshaw e Wiseman, 1992). 21
Figura 2.2 – Comparação das técnicas de visualização aplicadas ao mesmo fluxo 2D: (a) plotagem de setas, (b) *streamlets*, (c) *Line Integral Convolution* (LIC), (d) baseado na topologia (fonte: Weiskopf e Erlebacher, 2004). 25
Figura 2.3 – Visualização 3D *Glyph-based* combinada com *streamlines* iluminadas (fonte: Weiskopf e Erlebacher, 2004) 26
Figura 2.4 – Combinação de streamlines, streamribbons, setas e codificação de cores para um fluxo 3D (fonte: Weiskopf e Erlebacher, 2004). 28
Figura 2.5 – 3D LIC com percepção de profundidade melhorada (fonte: Interrante e Grosch, 1998) 33
Figura 2.6 – Exemplos de visualização direta de fluxos (fonte: Schulz et al., 1999). 35
Figura 2.7 - Comparação de técnicas de FlowViz - FlowViz pelo uso de setas (esquerda) é comparada a FlowViz baseada na integração de objetos (meio) e FlowViz com espaço preenchido pelo uso da LIC (direita) (fonte: Hauser et al., 2002). 36
Figura 2.8 - As cadeias de setas indicam a magnitude e direção do vento ao longo da Austrália. As setas foram depositadas ao longo de stream lines, criadas pela otimização. O campo de dados vetoriais foi calculado utilizando um modelo numérico meteorológico. As altas velocidades são mostradas com setas mais largas e mais opacas (fonte: Turk e Banks, 1996) 36
Figura 2.9 – Exemplo de FlowViz usando stream lines iluminadas (fonte: Hauser et al.,2002) 37
Figura 2.10 – As streak balls mostram o fluxo em um conjunto de dados de uma aleta. As bolas vermelhas são computadas num grid curvilíneo esparso de nível 4, as amarelas em nível 3 e as verdes em nível 2. (fonte: Teitzel e Ertl, 1999) 37
Figura 2.11 – Um campo de fluxo passando através de um cilindro circular. (a) Função principal do fluxo (b) Superfícies principais do fluxo (c) superfícies principais utilizando o mapeamento de cores. (fonte: Cai e Heng, 1997) 38
Figura 2.12 – Princípio do *Spot Noise* : (a) mancha única, (b) textura de spot noise e (c) spot noise usado para visualizar um campo vetorial (fonte : Leeuw et al., 1996) 39
Figura 2.13 – Fluxo circular com (a) OLIC e (b) FROLIC (fonte: Wegenkittl e Gröller, 1997). 39
Figura 2.14 – Dois sistemas dinâmicos visualizados pelo uso do método das gotas de tinta virtuais. Um papel quadriculado é usado como fundo. (fonte: Löffelmann et al., 1997) 39
Figura 2.15 – Lado a lado uma comparação entre visualização experimental e spot noise (fonte: Leeuw et al., 1995). 40
Figura 2.16 – O processo da convolução LIC (fonte : Shen e Kao, 1997). 40
Figura 2.17 – Uma comparação entre três técnicas de LIC: UFLIC (esquerda), ELIC (centro), PLIC (direita) (fonte: Okada e Kao, 1997). 41
Figura 2.18 – Imagens volumétricas de um fluxo de dados 3D geradas usando LIC 3D. Fonte (Interrante e Grosch, 1997) 41

Figura 2.19 – Visualização de LIC volumétrica: O campo de fluxo é explorado usando um plano de recorte, que é mudado de posição interativamente (fonte: Rezk-Salama et al., 1999). 41
Figura 2.20 – Construção de Hierarquias do campo vetorial (fonte: Hecker et al., 1999)..... 42
Figura 2.21 – Um campo vetorial 3D visualizado pelo método do agrupamento (fonte: Garcke et al., 2000)..... 42
Figura 2.22 - Fluxo passando um cilindro estreito com vórtices aproximados por elipses e streamlines mostrados em uma fatia (fonte: Sadarjoen e Post, 1999)..... 43
Figura 2.23 – Visualização de uma estrutura de fluxo global de um quadrado. O filtro de distância par é usado com uma distância de 0.0001*H – onde H é a altura do conjunto de dados (fonte: de Leww e van Liere, 1999)..... 43
Figura 2.24 - Regiões com elevada densidade helicidal normalizada e stream tubes através dessas regiões (fonte: Van Walsum et al., 1996). 44
Figura 2.25 – Reconstrução de uma superfície esqueletal (fonte: Reinders et al., 2000). 44
Figura 2.26 – Simulação de um modelo de turbina Kaplan. Este estudo de caso descreve a utilização da visualização do resultado de simulação CFD para o projeto de turbinas (fonte: (Roth e Peikert, 1996). 45
Figura 2.27 - Contornos de pressão estática em um capacete de bicicleta (fonte: Fluent, 2010). 48
Figura 2.28 - Previsões de fluxo de ar de refrigeração para otimizar a transferência de calor em torno de motor elétrico (fonte : CFX, 2010) 49
Figura 2.29 – Ambiente de visualização CFDStudio (fonte: ANSWER,2010). 50
Figura 2.30 Flowmaster 1D e Co-simulação 3D (fonte: Flowmaster, 2010) 51
Figura 2.31 Fluxo através de edifícios do complexo Juberiah Beach – Dubai (fonte: Cham, 2010)..... 52
Figura 2.32 Simulação da dissipação da poluição atmosférica urbana usando o Fluidyn PANAIR (fonte Fluidyn, 2010)..... 53
Figura 2.33 Quadro de filme MPEG da malha adaptativa de um fluxo de vento turbulento. As duas seções são coloridas de acordo com a norma do vetor de velocidades. Tempo de processamento: 7 horas (fonte: Gerris, 2010). 55
Figura 3.1 – Classificação possível da Mecânica dos Fluidos (fonte: Fox e McDonald, 2001) 58
Figura 3.2 - Representação dos 26 casos de fronteiras de um ponto dentro de uma malha computacional, usados para o cálculo do potencial. 62
Figura 4.1 – Cubo com (a) 6 linhas de corrente e (b) 30 linhas de corrente usado como referência na comparação de velocidade de processamento entre a compilação Borland C++, Visual C++ e Visual C#. 68
Figura 4.2 – Comparação entre Borland C++, Visual C++ e Visual C#. Nos três casos a performance, medida em FPS, diminui com o aumento das linhas de corrente visualizadas. . 68
Figura 4.3 – Interface do Cmake. 70
Figura 4.4 - Componente *VTKRenderWindow* instalado no Borland C++ Builder 6. 71
Figura 4.5 - Acesso às classes do VTK pelo Borland Builder C++. 72
Figura 4.6 – Menu Principal..... 72
Figura 4.7 – Fluxograma do simulador desenvolvido..... 73
Figura 4.8 - Exemplos de formas geométricas: semi-esfera, esfera+cubo e perfil de asa, visualizados utilizando o programa desenvolvido neste trabalho. 74
Figura 4.9 - Visualização das linhas de corrente em tempo quase-real, usando o simulador desenvolvido neste trabalho. 75

Figura 4.10 - Exemplos de visualização de linhas de corrente iniciadas a partir de um plano de origem, com representação por streamribbons, e a partir de uma reta de origem, com representação por streamlines. 77
Figura 4.11 - Visualização do gradiente de velocidades (a), de isolinhas (b) e ambos (c), feita a partir de planos de corte no sentido dos eixos xy. 78
Figura 4.12 - Visualização estérea de linhas de corrente a partir do modo *cyan-magenta*. 79
Figura 4.13 Menu *opções* para seleção interativa 80
Figura 4.14 Ferramentas para alterar as dimensões, em tempo quase real, dos objetos. 81
Figura 4.15 – Janela de saída de dados. 82
Figura 4.16 - Opções de visualização de linhas de fluxo (a) 8 streamlines iniciadas por uma linha de origem e (b) 144 streamlines iniciadas por um plano de origem. 82
Figura 4.17 - Simulador desenvolvido neste trabalho para visualização 3D de escoamentos tridimensionais sobre objetos. 83
Figura 5.1 – Grid representativo para a resolução da equação de Laplace. A face amarela possui um potencial P e a face azul um potencial nulo. 87
Figura 5.2 – Erro local relativo em função do número de iterações. 88
Figura 5.3 – Erro relativo após a inserção e retirada de figuras geométricas do grid. 89
Figura 5.4 – Verificação do erro relativo. 89
Figura 5.5 – 576 streamlines através de um cubo 90
Figura 5.6 - 576 streamlines através de uma esfera 90
Figura 5.7 – Componentes das velocidades à frente (a) e atrás (b) de uma esfera. 91
Figura 5.8 - Componentes das velocidades acima (a) e abaixo (b) de uma esfera. 92
Figura 5.9 – Força de sustentação em uma esfera. 93
Figura 5.10 – Força de sustentação em um perfil de asa com fundo plano. (a) Força obtida igual a 24,96 N e (b) Força obtida igual a 29,93 N. 94

SUMÁRIO

1 INTRODUÇÃO 13
1.1 Definição do Problema 15
1.2 Motivação 16
1.3 Objetivos 17
1.4 Características 17
1.5 Organização da Tese 18
2 MÉTODOS DE VISUALIZAÇÃO 19
2.1 Visualização de Fluxos 21
2.2 Softwares CDF Comerciais Existentes 45
2.2.1 Breve comparação entre os programas existentes 55
3 MODELO CFD NUMÉRICO PARA A EQUAÇÃO DE LAPLACE 3D 57
3.1 Dinâmica dos Fluidos Computacional (CFD) 57
3.2 Condições de Contorno 61
3.3 Velocidades 63
3.4 Implementação Computacional 64
4 MODELO DE VISUALIZAÇÃO DA EQUAÇÃO DE LAPLACE 3D 66
4.1 VTK e Borland C++ Builder 6 69
4.2 Desenvolvimento e Implementação 71
4.2.1 Figuras geométricas 73
4.2.2 Linhas de Corrente 74
4.2.3 Planos de Corte 78
4.2.4 Visualização Estereoscópica 79
4.2.5 Opções de Interatividade 79
5 VALIDAÇÃO E ESTUDOS DE CASO 85
5.1 Força de sustentação (Lift) 92
6 CONCLUSÃO 95
REFERÊNCIAS 98
ANEXOS 118
C++ 118
C++ Builder 6 118
Microsoft Visual C++ 119
Microsoft Visual C# 119
Visualization Toolkit – VTK 119
APÊNDICE 1 Listagem de programas de computador 121

1 INTRODUÇÃO

Os fluidos e suas propriedades vêm sendo pesquisados desde a antiguidade. Estes estudos eram eminentemente experimentais, pois não havia nenhuma teoria física que explicasse o movimento e suas propriedades. Em muitas indústrias vários produtos estão diretamente relacionados com a Dinâmica dos Fluidos e muitos recursos são investidos para o desenvolvimento de novas tecnologias, computacional e experimental (Krüger et al., 2005). Os resultados experimentais, como os obtidos em túneis de vento, são muito dispendiosos e de difícil reprodução, por exemplo, um escoamento em torno de um submarino, as dimensões tornariam o experimento oneroso. Por esta razão, a modelagem matemática e a simulação numérica vêm ganhando muito espaço. Este é o método chamado Dinâmica dos Fluidos Computacional ou CFD (Computational Fluid Dynamics) em inglês. Basicamente o usuário da CFD está interessado em obter as distribuições de velocidades, pressões, densidades tensões, temperaturas etc., na região do escoamento.

Com simulações em computador pode-se acelerar o desenvolvimento de um projeto, por meio da redução do tempo de testes em túneis de vento, por exemplo. Como todos os parâmetros simulados são "virtuais", a princípio basta alterá-los e executar novamente o programa de simulação (Fortuna, 2000). Para visualização de campos vetoriais tridimensionais, uma variedade de técnicas foi desenvolvida no passado. Estas técnicas incluem métodos baseados em geometria, como linhas de corrente e partículas animadas, que lembram o que os cientistas estão habituados a ver nos seus experimentos, assim como técnicas mais avançadas que utilizam hardware gráfico para gerar texturas de fluxo realistas (Li, 2007).

O desenvolvimento de novos produtos comerciais leva as indústrias a terem acesso a resultados que, anteriormente, só eram possíveis em laboratórios ou grandes centros acadêmicos. Os novos projetos, ou atualizações de antigos, desenvolvidos com os recursos da CFD são mais econômicos, pois reduzem o tempo de trabalho, e trazem estudos sobre escoamentos, fluxos, pressões e velocidades.

Gordon E. Moore, em 1965, profetizou que o número de transistores dos chips teria um aumento de 100%, pelo mesmo custo, a cada período de 18 meses. Este padrão mantém-se até hoje e permite que o processamento evolua da mesma forma trazendo, em conseqüência, maior capacidade de resolução de equações complexas. Hoje existem placas de vídeo com 480 núcleos, disponíveis para computadores comuns (desktops ou notebooks), com

processamento paralelo e soluções computacionais para workstations aonde o desempenho chega a ser 250 vezes superior a um simples PC (nVidia, 2010).

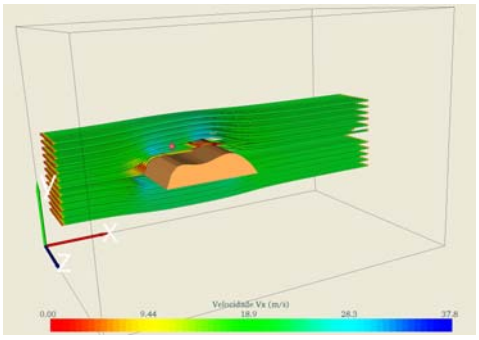
Desta forma, aproveitando este crescimento tecnológico, este trabalho relata o desenvolvimento de um simulador para a visualização de fluxos irrotacionais sobre objetos 3D, utilizando um algoritmo desenvolvido pelo autor, aplicado de forma inédita, com o uso de pacotes gratuitos de visualização, como o VTK (VTK, 2010) e com uma interface amigável e executável em plataformas comerciais, como o MS Windows (Windows, 2010). Tal aplicativo permite a visualização de fluxos tridimensionais sobre objetos, interativamente, no qual o cálculo e a visualização são executados em tempo quase-real[1]. A figura 1.1 mostra *screenshots* feitos imediatamente após pequenas mudanças em uma forma geométrica. Acompanhando uma janela de resultados e selecionando um ponto como referência, o usuário pode chegar a valores desejados de velocidades fazendo as modificações que achar conveniente no projeto, ou na forma geométrica apresentada. No exemplo mostrado na figura 1.1 foram usadas 144 streamlines (linhas de corrente), dispostas de maneira a não prejudicar a visualização da referência desejada. Neste exemplo a componente da velocidade usada para comparação foi referente ao eixo X (*Vx)* e os valores obtidos variaram com a mudança do perfil. O usuário poderá ter como objetivo o aumento, a redução ou mesmo a preservação da velocidade ou a obtenção de um valor mínimo ou máximo de força de sustentação devido ao fluxo aplicado.

Embora os métodos computacionais evoluam constantemente, Fortuna (2000) cita que: "É um erro pensar que a CFD irá substituir as técnicas experimentais e as análises teóricas. Ela permite racionalizar o planejamento dos experimentos, por meio da redução do número de horas necessárias em túneis de vento, por exemplo."

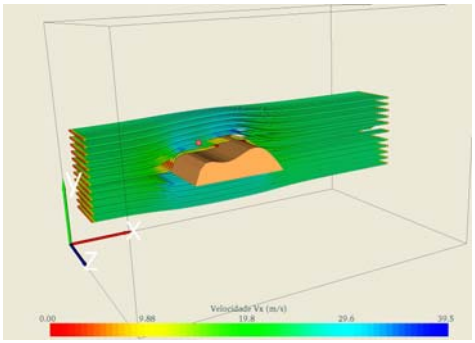
Ainda, dentro do mesmo pensamento, Hamming (1973), citado por Fortuna (2000) diz: "O objetivo da análise numérica é dar insight aos seus usuários, e não números". Deve-se, portanto, pensar na CFD como uma ferramenta que nos auxilia a compreender a natureza dos fenômenos envolvendo o movimento de fluidos. O simulador, descrito neste trabalho auxiliará o desenvolvimento de um produto industrial ou comercial, ou parte dele, onde o estudo sobre o escoamento de fluidos seja importante, como num capacete para um usuário de motocicleta ou carro de corrida. As formas geométricas iniciais ou definitivas poderão ser analisadas e modificadas em tempo quase-real até se atingir o resultado desejado. Tal

[1] O método mais antigo de contar o tempo apoiava-se em alguns fenômenos prontamente reconhecíveis. Um exemplo é a contagem dos dias em auroras, que encontramos em Homero (Whitrow, 1993). Neste trabalho considera-se tempo quase-real o atraso entre o fornecimento de dados e a resposta alcançada, devido ao processamento computacional.

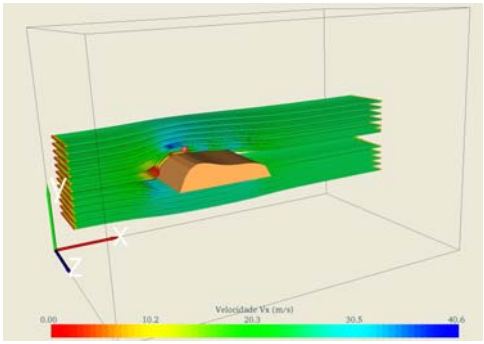
aplicativo, além de seu uso comercial, poderá ser usado didaticamente como ferramenta de visualização e cálculo em disciplinas nas quais a aplicação da CFD seja objeto de estudo.



(a)



(b)



(c)

Figura 1.1 – Screenshots do simulador desenvolvido neste trabalho. O ponto em vermelho é a referência fixa escolhida como parâmetro sendo que a velocidade *Vx* varia devido as alterações na forma geométrica.

1.1 Definição do Problema

O uso de ferramentas computacionais como elemento inovador e dinamizador para aumentar a competitividade das empresas e principalmente por parte das pequenas e médias empresas requer uma visão de mercado diferenciada, considerando que a utilização de tal

ferramenta é investimento e não custo. O emprego de ferramentas computacionais tal como a de CFD além de melhorar a competitividade, oferece tecnologia para realização de experimentos visando explorar os fenômenos que, às vezes, são impossíveis de serem estudados em laboratórios.

Existem no mercado vários pacotes CFD comerciais e acadêmicos e, normalmente, são extensões de aplicativos CAD, como o Flow Simulation da SolidWorks (figura 1.2). Alguns mais conhecidos são: Fluent (2010), CFX (2010), ANSWER (2010), Flowmaster (2010), Phoenics (Cham, 2010) e Fluidyn (2010). O pacote CFD da Ansys, por exemplo, custa cerca de R$ 325.000,00 no Brasil (fornecido por www.ESSS.com.br). Alguns desenvolvimentos CDF geraram aplicativos gratuitos, mas são executáveis apenas em plataformas Linux e não possuem uma visualização dos resultados, apenas a resolução matemática, como o FOAM (2010) e o Gerris (2010). Estes novamente recorrem a outros pacotes de visualização, fornecidos pela Ansys, por exemplo, para expor seus resultados. Além disso, apesar da alta qualidade de imagens geradas e da grande capacidade computacional, nenhum desses aplicativos, gratuitos ou não, possui a possibilidade de mudança da geometria de objetos, aliado à visualização tridimensional interativa de linhas de fluxos, em tempo real ou quase-real. Para obter-se um filme é necessário unir as imagens estáticas quadro a quadro após dezenas de horas de processamento. Esta é uma característica de todos os aplicativos encontrados no mercado. Com um processamento e visualização em tempo quase-real será possível gerar filmes ao mesmo tempo em que um produto está sendo desenvolvido. Isto permitirá uma avaliação contínua e o armazenamento de informações para análise futura.

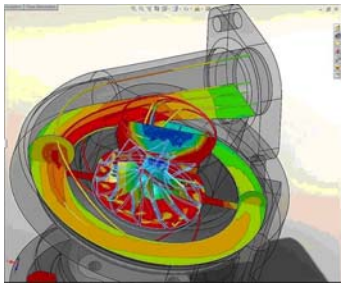


Figura 1.2 Modelo de visualização apresentado pelo software SolidWorks. Todo o cálculo é feito antes da visualização (fonte: Solid, 2010).

1.2 Motivação

Aproveitando a lacuna existente nos aplicativos computacionais comerciais, a evolução da capacidade computacional e utilizando os conhecimentos adquiridos durante o

processo acadêmico, este trabalho apresenta a construção de um sistema de visualização 3D de fluxos irrotacionais, através da solução numérica da equação de Laplace, utilizando campos vetoriais de fluxo constante com grades uniformes, simulando um túnel de vento virtual (Bryson, 1992), mas com a possibilidade de alteração geométrica de objetos, a visualização de gradientes de velocidades e o cálculo da força de sustentação em tempo quase-real.

1.3 Objetivos

Este trabalho tem como objetivos:

Levantar o estado da arte sobre as formas de visualização científica de escoamentos;

Fazer um pequeno estudo e comparação entre modelos comerciais CFD, disponíveis no mercado, que permitem a visualização do escoamento através de objetos;

Construir um modelo numérico para a equação de Laplace tridimensional que contempla a solução do escoamento sobre diversos objetos, permitindo a alteração geométrica e resolução em tempo quase-real;

Desenvolver um modelo de visualização interativa integrado com a solução numérica;

Avaliar e validar a ferramenta desenvolvida;

Concluir com sugestões para aperfeiçoamentos e melhorias futuras.

1.4 Características

Como características, o simulador demonstrado neste trabalho inicialmente trabalha com malhas estáticas, ou seja, o objeto alterado dentro da malha é interpretado como novo a cada instante e o objetivo é a comparação entre as alterações para a seleção da forma geométrica mais adequada, a critério do projetista. Porém, ao se alterar as formas geométricas, o método utilizado mostra uma evolução pseudo-dinâmica, que é visualizada pelas linhas de fluxo sendo modificadas em tempo quase-real, mostradas em uma janela de visualização.

Com este simulador o usuário terá acesso às simulações computacionais CFD, na forma da solução da equação de Laplace para a visualização do campo de velocidades e linhas de fluxo, acrescido de uma visualização interativa simultânea. Ele poderá ser aplicado no desenvolvimento de produtos onde a ação de fluxos tem significado importante para o projeto final. O sistema utiliza o método das diferenças finitas para a solução das equações. Para a

visualização das linhas de fluxo utiliza-se a integração de Runge-Kutta. A interface gráfica foi desenvolvida utilizando-se a linguagem C++ (Deitel e Deitel, 2001) (Alves, 2002) associada ao software gratuito VTK (Schroeder et al., 2004), que é um sistema de software aberto com um conjunto de bibliotecas para processamento de imagem e visualização 3D. Um algoritmo de visualização estereoscópica foi utilizado, assim como planos de corte nos três eixos com visualização do gradiente de velocidades e/ou isolinhas. A possibilidade de manipulação dos objetos e o processamento numérico em tempo quase-real para a visualização de linhas de fluxo, assim como um algoritmo de cálculo são descritos neste trabalho.

O capítulo 4 traz um pequeno fluxograma e a apresentação do simulador desenvolvido.

1.5 Organização da Tese

Este trabalho é constituído por seis capítulos, cujo comentário é brevemente discutido na sequência:

- O Capítulo 1 apresenta uma breve descrição do problema, a motivação pela escolha do tema e os objetivos principais deste trabalho;
- No Capítulo 2 é apresentado um pequeno estudo sobre o estado da arte da visualização tridimensional, onde se identificam os métodos de visualização de fluxos, baseados em suas características, além de uma breve pesquisa sobre ferramentas comerciais existentes que disponibilizam, de alguma forma, o cálculo e a visualização da CFD;
- No Capítulo 3 fala-se sobre a Dinâmica dos Fluidos Computacional, com um breve histórico, sua origem e suas aplicações e as resoluções e implementações computacionais do modelo numérico tridimensional usado neste trabalho.
- O Capítulo 4 traz o modelo de visualização dos resultados computacionais obtidos, traduzidos pela construção do simulador onde houve a utilização da linguagem de programação C++ aliada a pacotes gratuitos de visualização. As opções de interação e visualização oferecidas também são descritas;
- No Capítulo 5 mostra-se a validação do algoritmo de cálculo desenvolvido, assim como alguns estudos de caso;
- O Capítulo 6 encerra este trabalho com as conclusões.

2 MÉTODOS DE VISUALIZAÇÃO

Neste capítulo faz-se um breve levantamento do estado da arte sobre as formas de visualização de fluxos. Procura-se classificar os tipos de visualização em função dos dados obtidos e as formas de apresentação.

O conceito de visualização possui interpretações de diversos autores. Este trabalho cita algumas:

a) Para Paiva et al. (Paiva et al., 1999), visualização é um termo relacionado aos métodos que permitem a extração de informações relevantes a partir de complexos conjuntos de dados, processo geralmente feito através da utilização de técnicas de computação gráfica e processamento de imagens. Denomina-se visualização científica quando estes conjuntos de dados representam fenômenos complexos e o objetivo é a extração de informações científicas relevantes.

b) Iescheck (Iescheck, A.L., 2006) entende visualização como a criação de imagens gráficas que mostram dados para as interpretações visuais, buscando explorar a habilidade inata do ser humano de impor ordem, reconhecer padrões e construir um modelo mental. Assim, a visualização é um processo cognitivo relacionado à extração de informações a partir de estímulos visuais. Pode-se interpretar a visualização como um método computacional que incorpora coleta, organização, modelagem e representação de dados e que, desta forma, combina a capacidade visual de análise com os recursos computacionais disponíveis.

c) Robertson (Robertson, 1991) cita que "Fundamental para a interpretação, a análise e a compreensão adequada dos dados, a visualização está diretamente relacionada com a forma de representação adotada e esta, por sua vez, depende do tipo de informação desejada e das diferentes características a serem abordadas, inclusive características intrínsecas do dado. Uma representação adequada é a chave para apreciação crítica e compreensiva do dado, beneficiando os processos de análise, processamento e tomada de decisão."

d) De acordo com Earnshaw e Wiseman (Earnshaw e Wiseman, 1992), a visualização científica, também conhecida por análise visual de dados, é um método computacional que transforma o simbólico em geométrico, para possibilitar a observação dos processamentos e simulações realizados a partir de dados. Este método engloba o entendimento e a síntese da imagem, e atua como uma ferramenta tanto para interpretação de dados de imagem inseridos em um computador quanto para geração de imagens a partir de conjuntos de dados. A visualização científica está relacionada com a exploração de dados e informações na busca

pela compreensão intuitiva do dado, objetivo fundamental das pesquisas científicas, e pode ser considerada como um processo gráfico análogo à análise numérica.

O avanço tecnológico notado ao longo das últimas décadas tornou possível a realização de simulações e processamentos com um grau de complexidade muito maior do que anteriormente. Como resultado de tais procedimentos tem-se uma grande quantidade de dados e informações numéricos, na sua maioria, cujo entendimento e manipulação eficientes são prejudicados pela falta de uma representação adequada. Houve então, de acordo com Bartroli, citado por Iescheck (Iescheck, A. L., 2006), a necessidade de explorar novos algoritmos e técnicas, uma vez que as técnicas existentes não eram suficientes para proporcionar a visualização dos resultados complexos. A visualização científica tornou-se a união de processamentos complexos e computação gráfica, com o intuito de auxiliar os pesquisadores para melhor entender estruturas complexas.

Os sistemas de visualização científica são combinações de sistemas de hardware, software e procedimentos e envolvem conceitos de computação gráfica, processamento de imagem, sistemas, ciências cognitivas e outros. Através da visualização científica é possível ver e compreender, sobre uma tela bidimensional, dados multidimensionais e conjuntos de dados, o que de outra forma não seria possível. A diferença entre apresentação gráfica e visualização científica é, basicamente, que com a primeira busca-se a comunicação dos resultados das análises e, com a segunda, busca-se o entendimento dos dados e das informações utilizados nas análises.

Earnshaw e Wiseman (Earnshaw e Wiseman, 1992) citam que: “Visualização científica é um amálgama de ferramentas e técnicas que busca promover novas dimensões de discernimento para soluções de problemas utilizando a tecnologia atual.”

Uma área dentro da visualização científica que tem apresentado um rápido crescimento é a visualização de volumes (figura 2.1). A visualização de volumes é usada para criar imagens a partir de um conjunto de dados vetoriais ou campos escalares definidos em grades multidimensionais, ou seja, é o processo de projetar um conjunto de dados multidimensionais, geralmente tridimensionais, sobre uma imagem plana bidimensional, para obter o conhecimento da estrutura interna do dado. Um número associado a cada ponto de um volume é denominado de valor escalar nesse ponto. Campo escalar é o conjunto de todos os valores escalares do volume. A vantagem da visualização tridimensional está na forma como se vê a informação. Na visualização volumétrica simula-se a realidade espacial, para permitir ao usuário alcançar um reconhecimento e entendimento mais rápido, sem que haja a necessidade de construir um modelo mental prévio para fazer análises.

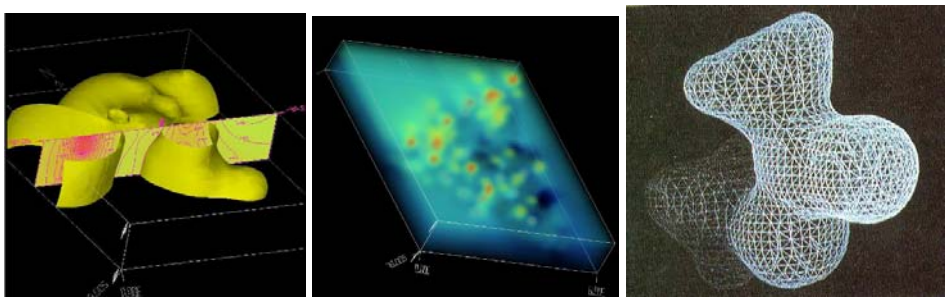


Figura 2.1 – Exemplos de Visualização Volumétrica: Iso-superfície, Rendering de Volume e Rede de Vetores. (fonte: Earnshaw e Wiseman, 1992).

A Visualização, como um campo separado de investigação e desenvolvimento em ciência da computação, resolve precisamente a ponte entre os dados e o usuário. As soluções de visualização ajudam os usuários a explorar, analisar e apresentar seus dados.

Na visualização de fluxos ou, abreviando em inglês, *FlowViz* (*Flow Visualization*) - uma das sub-áreas tradicionais de visualização - uma rica variedade de campos de aplicação é dada, incluindo indústria automotiva, aerodinâmica, design de turbocompressores, meteorologia e simulação do clima e aplicações médicas, com características bastante diferentes relacionadas com os objetivos e os dados do usuário. Consequentemente, o espectro de soluções da FlowViz é muito rico, abrangendo múltiplas dimensões dos aspectos técnicos, como por exemplo, soluções 2D vs 3D, técnicas para dados estacionários ou dependentes do tempo (Hauser et al., 2002). Várias opções de subdividir esse vasto campo da literatura são possíveis. Hesselink, Post e van Wijk (Hesselink et al., 1994), por exemplo, abordaram o difícil problema de saber como categorizar soluções sobre FlowViz em sua visão geral nas suas pesquisas em 1994.

2.1 Visualização de Fluxos

De acordo com as diferentes necessidades dos usuários, existem diferentes abordagens para a visualização de fluxos. Uma é para facilitar a, tão direta quanto possível, translação entre os dados do fluxo em métodos de visualização, tais como o desenho de setas. Esses tipos de soluções de FlowViz permitem a investigação imediata do fluxo de dados, sem muito esforço associado.

Para uma melhor comunicação do comportamento em longo prazo, induzidas por fluxos dinâmicos, as abordagens baseadas em integração, primeiramente integram o fluxo de dados e usam a integral resultante como base para a visualização.

Outro passo na complexidade é desempenhar uma computação ainda maior e derivar informações topológicas ou aspectos do fluxo antes de fazer o mapeamento da visualização. Com este tipo de soluções de FlowViz, uma quantidade significativa de computação é gasta durante a visualização, para ajudar o usuário com a interpretação do fluxo de dados. Isto é especialmente útil (ou mesmo necessário) em casos em que são dadas grandes séries de dados, por exemplo, muitos dos passos de tempo num fluxo transiente em 3D (Hauser et al., 2002).

A visualização de sistemas complexos, tais como os biológicos, físicos e mecânicos, por meio de dados instrumentais e simulados, torna-se hoje indispensável para cientistas e engenheiros computacionais, pois de outra maneira seria impossível avançar na pesquisa sobre fenômenos que tais sistemas representam (Castro V. et al., 2000).

A visualização de fluxos é um tema importante na visualização científica e tem sido objeto de investigação ativa por muitos anos. Normalmente, os dados provêm de simulações numéricas, tais como os da dinâmica dos fluidos computacional, e precisam ser analisados por meio de visualização para proporcionar a compreensão do escoamento. Com o rápido aumento do poder computacional para simulações, a procura de métodos mais avançados visualização tem crescido (Weiskopf e Erlebacher, 2004).

Existe um grande número de técnicas de visualização de campos vetoriais, que podem ser distinguidas de acordo com as suas propriedades que respeitam a uma série de categorias. Na classificação seguinte são consideradas muitas questões importantes, mais do que uma taxonomia completa. Essas questões devem ser levadas em conta na escolha de uma forma de visualização (Weiskopf e Erlebacher, 2004).

Em um esquema de classificação, as técnicas são distinguidas pela relação entre um campo vetorial e sua representação visual associada. A visualização direta baseada em um ponto (point-based) leva em consideração o campo vetorial em um ponto e possibilita a representação visual em sua vizinhança. O campo vetorial é diretamente mapeado para primitivas gráficas, no sentido de que não há realização de processamento de dados intermediários. Outra classe é baseada em curvas características obtidas pelo rastreamento de partículas. Essa terceira classe pré-processa exaustivamente os dados para identificar características importantes, que depois servirão de base para a visualização.

Outro tipo de propriedade é a densidade de representação: o domínio pode ser escasso ou densamente coberto por objetos visualizados. Densidade é particularmente útil para a detecção de subclasses de partículas. Relacionada com a densidade está a distinção entre métodos local e global. Uma técnica global mostra essencialmente o fluxo total, enquanto

importantes características do fluxo poderiam ser eventualmente evidenciadas por uma técnica local.

A escolha de um método de visualização também é influenciada pela estrutura dos dados. A dimensão do campo em que o campo vetorial é definido desempenha um papel importante. Por exemplo, estratégias que funcionam bem em 2D poderiam ser menos úteis em 3D por causa das questões de percepção: o reconhecimento da orientação e posição espacial das primitivas gráficas são mais difíceis e primitivas importantes poderão ser obstruídas por outras. A dimensão também afeta o desempenho; uma técnica 3D tem que processar substancialmente mais dados. Se a visualização é limitada às fatias, ou cortes, ou hipersuperfícies em geral de um fluxo 3D, a projeção de vetores tangencialmente às hipersuperfícies tem de ser considerada. Além disso, uma distinção tem que ser feita entre os dados tempo-dependente e tempo-independente. Um fluxo permanente geralmente é muito menos exigente e as *streamlines* (linhas de fluxo), *streaklines* (linhas de emissão), e *pathlines* (linhas de trajetória) são idênticas. Por último, o tipo de malha (*grid)* tem de ser levado em consideração. Os dados podem ser fornecidos, por exemplo, em grids uniformes, retilineares, curvilíneos, ou não estruturados. Os tipos de grid afetam principalmente os algoritmos de visualização no que diz respeito ao armazenamento de dados, mecanismos de acesso e formas de interpolação. Idealmente, a representação final da visualização não dependerá da forma do grid.

Note-se que não existe uma técnica "ideal" para todas as questões da visualização. Por isso, muitas vezes é útil combinar diferentes abordagens para uma efetiva visualização geral.

De acordo com Weiskopf e Erlebacher (Weiskopf e Erlebacher, 2004), a classificação da visualização de fluxos é apresentada resumidamente e descrita a seguir:

Visualização direta baseada no ponto (Point-Based)

- Glifos ou setas;
- Isolinhas

Representações esparsas para técnicas de rastreamento de partículas

- Pathlines;
- Streamlines;
- Streaklines;
- Time lines;
- Streamlets;
- Streamribbons;
- Streamtubes;
- Dash tubes;
- Streampolygons;

- Streamballs;
- Streamsurfaces;
- Stream arrows;
- Time surfaces.

Representações densas para técnicas de rastreamento de partículas

- Spot Noise;
- LIC;
- OLIC;
- FROLIC;
- PLIC;
- UFLIC;
- DLIC;
- LEA;
- IBFV
- Splatting;
- Line bundles.

Visualização direta baseada no ponto (Point-Based). A tradicional técnica de plotagem de setas é um exemplo conhecido para a visualização direta do fluxo baseado em glifos. As pequenas setas são desenhadas em pontos discretos do grid, mostrando a direção do fluxo e servindo como valores locais para o campo de velocidades (figura 2.2 (a)). O fluxo é visualizado por segmentos de reta cujos comprimentos representam a magnitude da velocidade. Os desenhos de setas podem ser aplicados diretamente aos campos vetoriais dependentes do tempo, habilitando as setas a se adaptarem ao campo de velocidades para o tempo atual. Para representações 3D, as seguintes questões têm de ser consideradas: a posição e orientação de uma seta são mais difíceis de ser entendida, devido à projeção sobre o plano 2D e uma seta poderia ocultar outras que estejam em segundo plano. O problema da confusão com a quantidade de setas pode ser dirigido destacando setas com orientações em uma variedade especificada pelo usuário, ou selecionando a origem das setas. A iluminação e sombras podem servir para melhorar a percepção espacial. Por exemplo, sombreamento pode ser aplicado a visualizações em 2D fatiadas de um fluxo 3D (Klassen e Harrington, 1991).

Glifos mais complexos podem ser utilizados para fornecer informações adicionais sobre o fluxo em um determinado ponto do fluxo (de Leew e van Wijk, 1993). Dados típicos codificados em glifos compreendem velocidade, aceleração, curvatura, rotação local, cisalhamento e convergência (Wittenbrink et al., 1996).

Outra estratégia é mapear as propriedades do fluxo em um único valor e aplicar técnicas conhecidas para a visualização de dados escalares. Tipicamente, a magnitude da velocidade ou de um dos componentes da velocidade é utilizada. Para a visualização de um fluxo 2D, um

mapeamento a cores ou a isolinhas (linhas de contorno) muitas vezes é aplicado. As técnicas de visualização de volume têm de ser empregadas no caso dos dados em 3D (Clyne e Dennis, 1999) (Glau, 1999) (Ono et al., 2001).

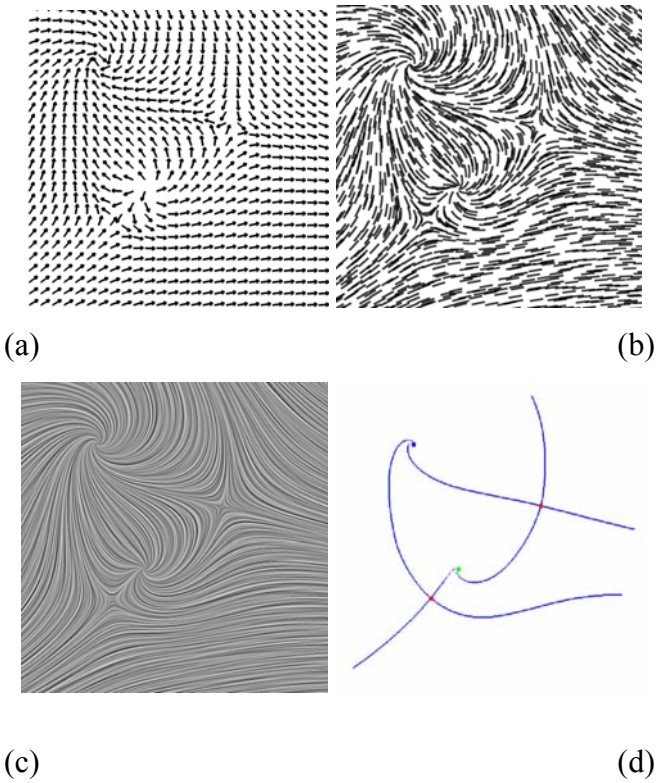


Figura 2.2 – Comparação das técnicas de visualização aplicadas ao mesmo fluxo 2D: (a) plotagem de setas, (b) *streamlets*, (c) *Line Integral Convolution* (LIC), (d) baseado na topologia (fonte: Weiskopf e Erlebacher, 2004).

Representações esparsas para técnicas de rastreamento de partículas. Outra classe de visualização abordada é baseada nas linhas características obtidas pelas trajetórias de partículas. Entre elas estão as referidas *pathlines, streamlines* e *streaklines*. Além destas, podem ser usadas as *time lines*, construídas a partir de partículas liberadas, ao mesmo tempo, a partir de diferentes pontos ao longo de uma curva. Todas estas linhas são bastante intuitivas porque elas representam algum tipo de transporte ao longo do fluxo.

A abordagem tradicional baseada em partículas (*particle-based)* calcula as curvas características e as desenha com linhas finas. Uma vez que muitos pesquisadores manipulam campos vetoriais independentes do tempo, a noção de streamlines é usada com frequência. Os conceitos de visualização muitas vezes podem ser generalizados a pathlines, streaklines, ou time lines, mesmo quando não explicitamente referidas. As streamlines apenas servem como

um modelo para as outras linhas características. Partículas traçadas durante um curto espaço de tempo geram streamlines curtas, ou *streamlets*.

As streamlines e streamlets podem ser usadas em um espaço 2D, em hipersuperfícies 2D de um fluxo subjacente em 3D, e para os fluxos 3D. Hipersuperfícies tipicamente são fatias seccionadas através de um volume ou superfícies curvas, como fronteiras ou outras superfícies características. É importante notar que o uso do traçado de partículas para campos vetoriais projetados sobre fatias pode ser enganoso, mesmo dentro de um fluxo permanente: a streamline sobre uma fatia pode retratar um circuito fechado, embora nunca uma partícula fosse atravessar o loop. O problema é causado pelo fato de que os componentes do fluxo, ortogonais à fatia, são negligenciadas durante a integração do fluxo. Para os fluxos 3D, problemas perceptivos poderão surgir devido a distorções resultantes da projeção no plano da imagem.

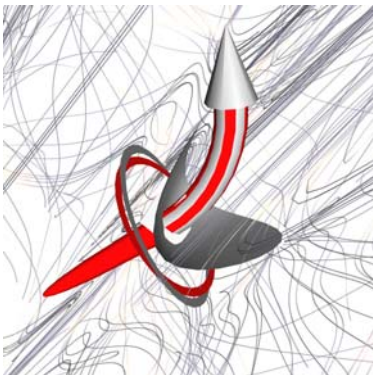


Figura 2.3 – Visualização 3D *Glyph-based* combinada com *streamlines* iluminadas (fonte: Weiskopf e Erlebacher, 2004)

Além disso, problemas de oclusão e desorganização têm que ser considerados (Fuhrmann e Gröller, 1998). Uma solução adequada é a de encontrar posições de origem para o traçado de partículas que mostram características importantes do fluxo, mas não superlotam o volume; por exemplo, uma corda de streamlets, ao longo de estruturas características de fluxo em 3D, pode ser usada. O método de streamlines iluminadas (Zöckler et al., 1996), baseado na iluminação em diversas codimensões (Banks, 1994), melhora a percepção dessas linhas, aumenta a profundidade de informação e aborda o problema da oclusão, tornando a streamline parcialmente transparente. Um exemplo é mostrado na figura 2.3.

Em 2D, o traçado de partículas é geralmente representado por linhas finas, embora a largura de uma linha seja por vezes alterada de modo a representar mais informações. A figura 2.2 (b) mostra um exemplo de uma coleção de streamlets. Em aplicações 3D, no entanto, a dimensão espacial adicional permite obter mais informações a serem colocadas na

representação gráfica, utilizando objetos geométricos de extensões finitas, perpendicularmente à trajetória da partícula. Exemplos de tal extensão de streamlines em 3D são os *streamribbons* e *streamtubes*.

Um streamribbon é a área varrida por um segmento de linha deformável ao longo de uma streamline. A forma de um streamribbon exibe o comportamento rotacional de um fluxo 3D. A figura 2.4 mostra uma visualização em 3D de uma simulação de fluxo combinando streamribbons, streamlines, setas e cores codificadas (Schulz et al., 1999).

Um streamtube (Ueng et al., 1996) é uma streamline na forma de tubo espesso cuja medida radial mostra a variação da velocidade do fluxo. Como uma extensão de streamtubes, os *dash tubes* (Fuhrmann e Gröller, 1998) proporcionam tubos com opacidade mapeada e animação. *Stream polygons* (Schroeder et al., 1991) traçam uma geometria arbitrária de uma seção poligonal transversal ao longo de uma streamline e, portanto, estão intimamente relacionadas aos streamtubes e streamribbons. As propriedades dos polígonos, tais como o tamanho, a forma ou a orientação, refletem as propriedades do campo vetorial, incluindo deformação, deslocamento, e rotação.

As *streamballs* (Brill et al., 1994) usam seus raios para visualizar as acelerações do fluxo. Em vez de esferas, outros objetos geométricos como tetraedros (Teizel e Ertl, 1999) podem ser utilizados.

Outra extensão das streamlines são *stream surfaces*, que estão em toda parte tangente ao campo vetorial. Uma stream surface pode ser modelada por uma superfície implícita (van Wijk, 1993) ou aproximada por ligação explícita de um conjunto de streamlines ao longo das time lines. Stream surfaces apresentam desafios relacionados com a oclusão, complexidade visual e de interpretação, que podem ser abordados por uma escolha apropriada de colocação e orientação baseada nas stream surfaces principais (Cai e Heng, 1997) ou pela interação do usuário (Hultquist, 1990).

Ray casting pode ser usado para renderizar várias stream surfaces em diferentes profundidades (Frühauf, 1996).

Stream arrows (Löffelmann et al., 1997) corta porções em forma de seta a partir de uma stream surface e então fornece informações adicionais sobre o fluxo, tais como a direção do fluxo e a convergência ou divergência. Stream surfaces também podem ser visualizadas e computadas com base nas partículas de superfície (van Wijk, 1993), que estão menos sujeitas a oclusão do que uma superfície completamente preenchida.

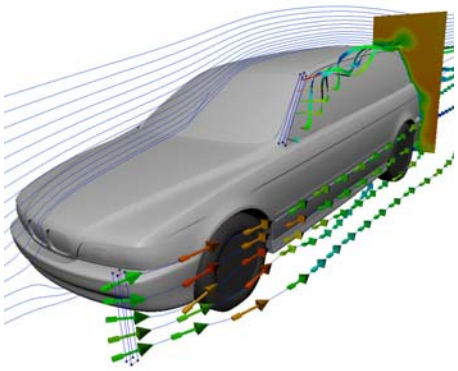


Figura 2.4 – Combinação de streamlines, streamribbons, setas e codificação de cores para um fluxo 3D (fonte: Weiskopf e Erlebacher, 2004).

A generalização do conceito de partículas libertadas por pontos únicos discretos (para streamlines ou streaklines) ou a partir de vários pontos em uma linha 1D (para stream surfaces) leva ao fluxo de volume (*flow volume)* (Max et al., 1993). Um fluxo de volume é um domínio de uma região de um fluxo 3D traçado por um anexo 2D ao longo do tempo. O volume resultante pode ser visualizado por técnicas de renderização de volumes. Uma vez que nenhum fluxo de volume pode ser (pelo menos aproximadamente) representado por uma coleção de células tetraédricas, as técnicas de renderização de volumes para malhas não estruturadas podem ser aplicadas, assim como a projeção de células aceleradas por hardware (Röttger et al., 2000) (Shirley e Tuchman, 1990). O fluxo de volumes pode ser estendido aos fluxos não permanentes (Becker et al., 1995), dando origem analogamente às streaklines. Por último, *time surfaces* estendem as time lines para superfícies que sejam construídas a partir de partículas libertadas por um anexo 2D. A evolução das time surfaces pode ser manipulada por uma abordagem *level-set* (Westermann et al., 2000).

Uma questão fundamental de todas as técnicas baseadas em partículas é a escolha adequada de condições iniciais - posicionamento dos pontos de origem (*seeds)* - a fim de capturar todas as características relevantes do fluxo. Duas estratégias principais podem ser identificadas:

a) Colocação interativa dos pontos de origem;
b) Colocação automática dos pontos de origem.

A abordagem interativa deixa o problema para o usuário e, nesse sentido, o que simplifica o problema a partir de um ponto de vista de algoritmos. No entanto, o sistema de visualização deve ser concebido para ajudar o usuário a identificar pontos de origem adequados. Por exemplo, o túnel de vento virtual (Bryson e Levit, 1992) é uma implementação breve de

realidade virtual de um sistema de visualização de fluxo interativa onde partículas podem ser liberadas pelo usuário.

Uma abordagem útil para a colocação automática de pontos de origem é a de construir uma distribuição uniforme das streamlines, que pode ser conseguido para campos vetoriais 2D (Jobard e Lefer, 1997) (Turk e Banks, 1996), ou para superfícies de fronteira com grades curvilíneas de um fluxo 3D (Mao et al., 1998). O foco de uma distribuição uniforme das streamlines é que uma distribuição agradável muito provavelmente não irá perder importantes características do fluxo. Por conseguinte, esta abordagem pode ser considerada como um passo no sentido de uma representação totalmente densa, que é discutido a seguir. As streamlines igualmente espaçadas podem ser estendidas a hierarquias multidefinidas que sustentam uma mudança interativa da densidade da streamline, quando se avança ou recua-se no zoom dentro do campo vetorial (Jobard e Lefer, 2001). Além disso, com esta técnica, a densidade das streamlines pode ser determinada pelas propriedades do fluxo, tais como a magnitude da velocidade ou a vorticidade. As streamlines uniformemente espaçadas para um fluxo não permanente podem ser realizadas correlacionando visualizações instantâneas de streamlines em tempos subsequentes (Jobard e Lefer, 2000). As estratégias das origens podem também ser baseadas na topologia do campo vetorial, por exemplo, a estrutura do fluxo na proximidade de pontos críticos podem ser visualizados por meio da escolha correta das condições iniciais para o traçado das partículas (Verma et al., 2000).

Uma vez que todas as técnicas de traçado de partículas são focadas em resolver a equação diferencial para o transporte de partículas, as questões de precisão numérica e velocidade devem ser abordadas. Diferentes técnicas numéricas podem ser aplicadas para o problema do valor inicial de equações diferenciais ordinárias. Em muitas aplicações, esquemas de integração explícita são utilizados, tais como os métodos adaptativos, ou não, de Runge-Kutta. A precisão exigida para o traçado de partículas depende da técnica de visualização, por exemplo, a integração de primeira ordem de Euler poderia ser aceitável para streamlets, mas não para streamlines. Uma comparação entre as diferentes formas de integração (Teizel et al., 1997) contribui para julgar o balanço entre o tempo de computação e a precisão. Além do sistema de integração real, a malha onde o campo vetorial é dado é muito importante para a escolha de uma técnica para o traçado de partículas. A posição de um ponto e a interpolação dependem fortemente da malha e, por conseguinte, afetam a velocidade e a precisão do traçado das partículas. Ambos os aspectos são detalhados em Nielson et al. (Nielson et al., 1997) juntamente com uma abordagem sobre a comparação entre C-espaço (espaço computacional) e P-espaço (espaço físico). O estudo numérico do traçado de partículas é

discutido, por exemplo, para a decomposição tetraédrica de malhas curvilíneas (Kenwright e Lane, 1996), especialmente para a decomposição de células distorcidas (Sadarjoen et al., 1998), para malhas não estruturadas (Ueng et al., 1996), para soluções analíticas em interpolações lineares discretas em malhas tetraédricas (Nielson e Jung, 1999), para equações diferenciais provenientes de cisalhamento de fluxos (Teizel et al., 1997) e para malhas esparsas (Teizel et al., 1998).

Representações densas para técnicas de rastreamento de partículas. Outra abordagem para as classes de visualização é baseada em uma representação do fluxo por uma densa cobertura através de estruturas determinadas pelas trajetórias das partículas. Normalmente, as representações densas são construídas sobre técnicas baseadas em texturas, que fornecem imagens de alta resolução espacial. Um resumo sobre as pesquisas no domínio das representações densas podem ser encontrados na pesquisa de Sanna (Sanna et al., 2000).

A distinção entre as técnicas densas e esparsas não deve ser tomada de modo demasiado rígido, pois ambas as classes de técnicas estão estreitamente relacionadas com o fato de que formam estruturas visuais baseadas na trajetória de partículas. Entretanto, as representações densas conduzem igualmente à mesma compreensão intuitiva do fluxo. Frequentemente, uma transição entre as duas classes é possível (Verma et al., 1999); por exemplo, as técnicas baseadas em textura, com poucos elementos visuais distintos, podem lembrar uma coleção de algumas streamlines e, por outro lado, igualmente streamlines originadas esparsamente podem ser utilizadas com uma elevada densidade de linhas.

Uma das primeiras técnicas de textura-sintetizada para visualização de campos vetoriais é o *spot noise* (van Wijk, 1991), que produz uma textura pela geração de um conjunto de sinais no domínio espacial. Cada sinal representa uma partícula em movimento durante um curto período de tempo e resulta em uma faixa no sentido do fluxo a partir da posição do sinal. O spot noise aprimorado (de Leeuw e van Wijk, 1995) acrescenta a visualização da magnitude da velocidade e é permitido a sinais curvos. O spot noise também pode ser aplicado sobre superfícies e fronteiras (de Leeuw e Pagendarm, 1995) (Telea e van Wijk, 1999). Uma estratégia de dividir-e-conquistar (*divide-and-conquer strategy*) (wiseGEEK, 2010) torna possível uma implementação do spot noise para ambientes interativos (de Leeuw, 1997). Como um exemplo de aplicação, um spot noise foi aplicado para a visualização de escoamentos turbulentos (de Leeuw et al., 1996).

Integral de convolução (LIC – *Line Integral Convolution*) (Cabral e Leedom, 1993) é uma técnica amplamente utilizada para a representação densa de streamlines em campos

vetoriais estacionários. Em vez de desenharmos uma coleção de curvas integrais para estudar um campo bidimensional, a idéia deste método é fazer a convolução de uma função ruído (textura) ao longo da linha de corrente (Giraldi, G., Feijóo, R., 2008). Um exemplo é mostrado na figura 2.2 (c). LIC toma como entrada um campo vetorial e uma textura de ruído branco. O ruído é localmente polido ao longo da streamline pela convolução com um filtro digital. Esta filtragem conduz a uma alta correlação ao longo das streamlines e pouca ou nenhuma correlação perpendicular às streamlines. O contraste e qualidade das imagens LIC podem ser melhorados através de técnicas de pós-processamento, tais como a equalização do histograma, filtragem passa-alta, ou um segundo passo da LIC (Okada e Kao, 1997). Tanto o ruído no local como a LIC são baseados na representação densa de texturas e trajetória de partículas e são, de um ponto de vista mais abstrato, estritamente relacionados uns aos outros (de Leeuw e van Liere, 1998). A técnica original LIC não revela a orientação e a magnitude do campo de velocidades, uma questão que é resolvida por variantes da LIC. Filtros de movimento periódicos podem ser utilizados para animar a visualização do fluxo, e uma troca da fase kernel pode ser aplicada para revelar a direção e magnitude do campo vetorial (Forssel e Cohen, 1995). A *Integral de Convolução Orientada* (*Oriented Line Integral Convolution* – OLIC) (Wegenkittl et al., 1997) explora a existência de manchas distintas e separadas em uma textura especialmente escassa e juntam estas manchas na direção do campo de velocidades local pela convolução com um filtro digital assimétrico para mostrar a orientação do fluxo. Sacrificando alguma precisão, uma versão rápida de OLIC (*FROLIC*) é viável (Wegenkittl e Gröller, 1997). Em outra abordagem, a orientação é visualizada combinando animação e adicionando advecção tingida (Shen et al., 1996). Ruído multi-frequência para LIC (Kiu e Banks, 1996) visualiza a magnitude da velocidade por meio da adaptação da frequência espacial de ruído.

Outras técnicas de visualização executam imagens parecidas com LIC através da aplicação de métodos não baseados na integral de convolução. Por exemplo, texturas parecidas com pêlos (Khouas et al., 1999) podem ser utilizadas, especificando a orientação, comprimento, densidade e cor dos filamentos dos pêlos de acordo com o campo vetorial. A integração e desenho (Risquet, 1998) aborda depósitos aleatórios de valores na escala de cinza ao longo das streamlines. *Pseudo LIC* (PLIC) (Verma et al., 1999) é um termo médio entre LIC e representações esparsas baseadas em partículas e, por conseguinte, permite uma mudança gradual entre visualizações densas e esparsas. PLIC utiliza a LIC para gerar um modelo de textura em um passo de pré-processamento. Para visualizações reais, o modelo é mapeado sobre finas ou "grossas" streamlines, preenchendo assim o domínio com estruturas

parecidas com LIC. A idéia de texturas LIC aplicadas a streamlines pode ser estendida a um método hierárquico de erro controlado numa abordagem em níveis de detalhes num acelerador gráfico (Bordoloi e Shen, 2002).

LIC pode ser estendido a grades não uniformes e superfícies curvas, por exemplo, a grades curvilíneas (Forssel e Cohen, 1995), malhas 2D triangulares ou não estruturadas (Mao et al., 1997) (Teizel et al., 1997), e superfícies arbitrárias em 3D (Battke et al., 1997). Ruídos multigranulares como alimentação para a LIC (Mao et al., 1998) compensa o mapeamento não-isométrico do espaço de texturas para as células de uma grade curvilínea que diferem em tamanho. A projeção da componente normal do campo vetorial precisa ser levada em conta para os tipos de visualizações LIC em hipersuperfícies (Scheuermann et al., 1999).

Unsteady Flow LIC (UFLIC) (Shen e Kao, 1998) ou a sua versão acelerada (Liu e Moorhead, 2002) incorpora o tempo em convolução para visualizar fluxos não permanentes. A questão de coerência temporal é resolvida por sucessivas atualizações no resultado da convolução ao longo do tempo. Forssel e Cohen, (1995), apresentam uma visualização de fluxos dependentes do tempo em superfícies curvilíneas é apresentado. *Dynamic LIC* (DLIC) (Sundquist, A., 2003) é outra extensão da LIC, que pode ser usada para campos vetoriais dependentes do tempo, como campos elétricos. Uma imagem semelhante à LIC de um fluxo não permanente também pode ser gerada por um método de visualização adaptativo que utiliza streaklines, onde a geração de streaklines é controlada pela vorticidade (Sanna et al., 2000).

Uma vez que a LIC tem de realizar uma integral de convolução para cada elemento de uma textura de alta resolução, os custos computacionais consistem um problema. Uma solução para este problema utiliza a coerência ao longo das streamlines para acelerar o processo de visualização (Hege e Stalling, 1998) (Stalling e Hege, 1995). Implementações paralelas são outra forma de lidar com elevados custos computacionais (Cabral e Leedom, 1995) (Zöckler et al., 1997). Por último, implementações baseadas em hardware gráfico podem melhorar o desempenho da LIC (Heidrich et al., 1999).

A partir de um ponto de vista conceitual, uma extensão da LIC para 3D é simples. A convolução ao longo das streamlines é realizada dentro de um volume; o volume, em escala de cinza, resultante pode ser representado por técnicas de visualização volumétrica, tais como a renderização volumétrica baseada em textura. No entanto, os custos computacionais são ainda mais elevados do que em 2D e, portanto, implementações interativas do processo de filtragem são difíceis de alcançar. Ainda mais importante, possíveis questões severas de oclusão têm de ser consideradas: em uma representação densa, há uma boa chance de se

esconder características importantes atrás de outras linhas de partículas. Uma combinação de cortes interativos e intervenção de usuários é uma solução possível (Rezk-Salama et al., 1999). Alternativamente, volumes 3D LIC podem ser representados por regiões de interesse, seletivamente enfatizadas, no fluxo, aumentando a profundidade de percepção e melhorando a orientação da percepção (Interrante, 1997) (Interrante e Grosch, 1997) (Interrante e Grosch, 1998); como ilustrado na figura 2.5.

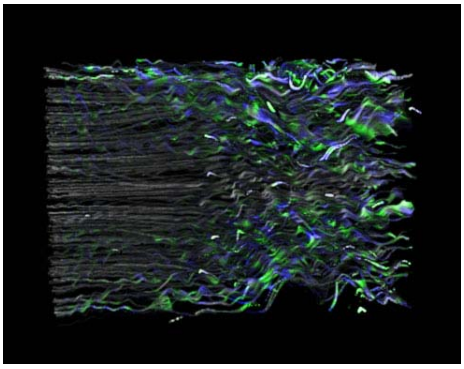


Figura 2.5 – 3D LIC com percepção de profundidade melhorada (fonte: Interrante e Grosch, 1998).

Outra representação densa é a baseada na advecção de texturas. O objetivo é representar uma coleção densa de partículas em uma textura e transportar essa textura de acordo com o movimento das partículas (Max e Becker, 1995) (Max et al.,1992). Por exemplo, as coordenadas Lagrangianas para o transporte de texturas podem ser computadas por um esquema numérico para equações de convecção (Becker e Rumf, 1998). O mapa de movimento (Jobard e Lefer, 1997) é uma aplicação do conceito de advecção de texturas para a animação de fluxo permanente 2D. O mapa de movimento contém uma representação densa do fluxo e as informações necessárias para a animação. A advecção Lagrangiana-Euleriana (LEA) (Jobard et al., 2001) é um sistema para visualização de fluxos permanentes através da integração da posição das partículas instáveis (ou seja, a parte Lagrangiana) e advectando a cor das partículas baseando-se numa representação de texturas (ou seja, o aspecto Euleriano). LEA pode ser estendido para a visualização de um movimento vertical em um fluxo 3D por meio de superfícies temporais (Grant et al., 2002). A advecção de textura está diretamente relacionada às capacidades de mapeamento de texturas dos hardwares gráficos e, por conseguinte, torna implementações eficientes possíveis (Becker e Rumpf, 1998) (Jobard et al., 2000) (Weiskopf et al., 2002) (Weiskopf et al., 2001). Outra vantagem da advecção de textura é o fato de que tanto a advecção de ruído como a de cor podem ser tratadas no mesmo quadro.

A advecção de texturas também pode ser aplicadas a fluxo 3D (Kao et al., 2001) (Weiskopf et al., 2001).

A imagem baseada na visualização de fluxo (*Image based flow visualization* - IBFV) (van Wijk, 2002) é uma variante, recentemente desenvolvida, de advecção de texturas 2D. Não só é a textura (ruído) transportada ao longo do fluxo, mas também uma segunda textura é misturada com a textura advectida em cada passo de tempo. IBFV é uma ferramenta flexível que pode imitar uma ampla variedade de estilos de visualização. Outra abordagem para o transporte de uma quantidade densa de partículas baseia-se na difusão não linear (Diewald et al., 2000). Uma primeira imagem de ruído é suavizada ao longo das integrais de linha de um fluxo constante por difusão, enquanto que a imagem está apontada na direção ortogonal. A difusão não linear pode ser estendida para a visualização multi-escalar do transporte nos fluxos dependente do tempo (Bürkle, 2001).

Por último, algumas técnicas de visualização de fluxos 3D adotam a concepção de splatting (uma única partícula fina achatada pulverizada), originalmente desenvolvido para renderização de volumes (Westover, 1990). Mesmo que algumas técnicas vetoriais de *splatting* não contam com o traçado de partículas, elas são aqui incluídas dado que a sua aparência assemelha-se com estruturas de curvas densas. "Arranhões" anisotrópicos podem ser modelados para texturas splats que são orientados ao longo do fluxo para mostrar a direção do campo vetorial (Crawfis e Max, 1993). Feixes de linhas (*line bundles)* (Max et al., 1994) usam a analogia do splatting para desenhar cada um dos pontos dos dados com um conjunto de segmentos de linhas pré-computados. Esses feixes de linhas semitransparentes são compostos em uma ordem de trás para frente a fim de atingir um resultado de renderização volumétrica anisotrópica. Para fluxos dependentes do tempo, a animação de um grande número de partículas mapeadas por textura ao longo das linhas de trajetórias pode ser usada (Guthe et al., 2002). Para todas as abordagens splatting, a densidade de representação depende do número de splats.

Em outra classificação apresentada por Hauser (Hauser et al., 2002), a visualização de fluxo (FlowViz) pode ser descrita da seguinte forma:

FlowViz direta – A técnica da visualização direta de fluxos procura apresentar os dados de maneira direta com o mínimo de computação entre a aquisição dos dados e renderização. Essas técnicas constituem, talvez, a estratégia mais intuitiva de visualização, apresentando os dados como realmente são. As dificuldades surgem quando o comportamento de longo prazo, induzido pelo fluxo de dados, é investigado. Se for utilizada a visualização direta, tornar-se necessária uma integração cognitiva dos resultados de visualização.

A FlowViz direta divide-se em:

a) FlowViz direta em 2D:

- Codificação de cores em 2D.
- Contorno em 2D.
- Representações de setas em 2D.
- FlowViz híbrida direta em 2D.

b) FlowViz direta sobre fatias ou fronteiras :

- Codificação de cores e contornos em fatias ou fronteiras.
- Setas 2D em fatias ou superfícies de fronteira.

c) FlowViz direta em 3D :

Em contraste com técnicas mencionadas anteriormente, aqui a representação gráfica torna-se uma questão mais crítica. A oclusão e a complexidade tornam difícil de obter uma visão geral sobre todos os dados de um fluxo em 3D. A figura 2.26 traz um exemplo da visualização direta de fluxos 3D.

Pode ser dividida em:

- Renderização volumétrica para FlowViz 3D.
- Iso-superfícies para FlowViz 3D.
- Representação gráfica de setas em 3D.

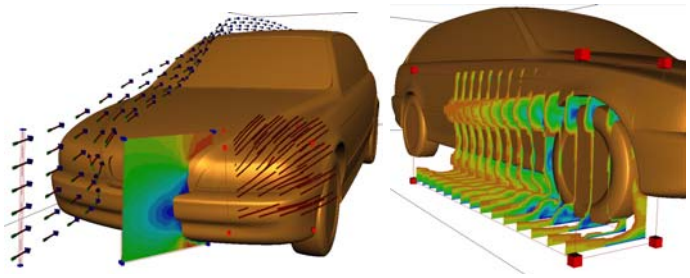


Figura 2.6 – Exemplos de visualização direta de fluxos (fonte: Schulz et al., 1999).

FlowViz usando objetos integrais. A visualização direta de fluxo utilizando representações de ouriço concentra-se em pontos individuais no campo de fluxo. No entanto, esquemas mais elaborados são introduzidos quando esses pontos, ou objetos similares, são deslocados dentro de um pequeno passo de tempo. Um glifo na forma de ouriço aproxima o movimento de um ponto para o período de tempo indicado pelo glifo em si. Uma extensão lógica dessa técnica é a de retratar o movimento de um ponto sobre mais de um passo de cada vez. O caminho resultante é expresso matematicamente como uma integral. Esse caminho instantâneo pode

ser retratado como uma stream line (no caso é considerado campo de fluxo permanente). Exemplos são mostrados nas figuras 2.7 e 2.8.

A técnica de FlowViz usando objetos integrais divide-se em:

a) FlowViz 2D usando objetos integrais.

- Streamlets em 2D.
- Streamlines em 2D.
- Streaklines, time lines (linhas de tempo), e pathlines (linhas de trajetória).
- Origem de stream line em 2D.

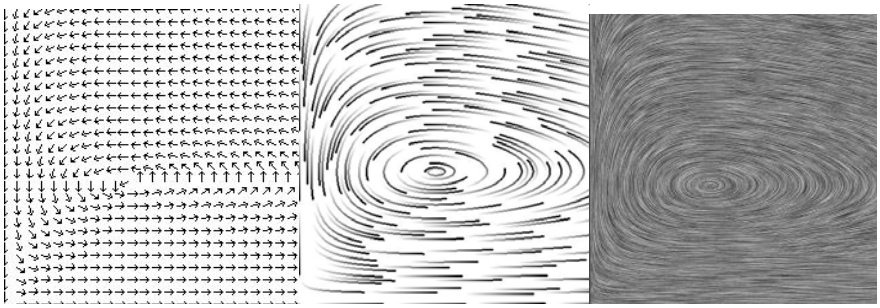


Figura 2.7 - Comparação de técnicas de FlowViz - FlowViz pelo uso de setas (esquerda) é comparada a FlowViz baseada na integração de objetos (meio) e FlowViz com espaço preenchido pelo uso da LIC (direita) (fonte: Hauser et al., 2002).

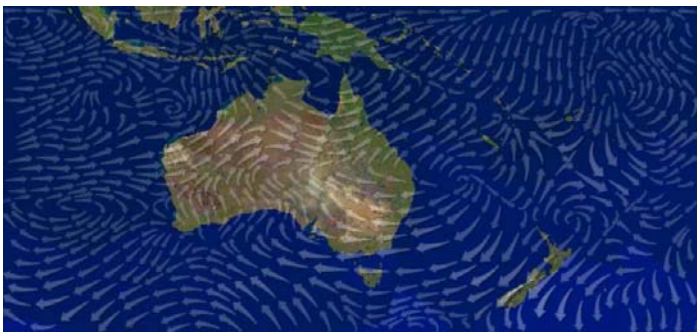


Figura 2.8 - As cadeias de setas indicam a magnitude e direção do vento ao longo da Austrália. As setas foram depositadas ao longo de stream lines, criadas pela otimização. O campo de dados vetoriais foi calculado utilizando um modelo numérico meteorológico. As altas velocidades são mostradas com setas mais largas e mais opacas (fonte: Turk e Banks, 1996).

b) FlowViz utilizando objetos integrais em fatias ou fronteiras

- Tufos Integrados.
- Objetos integrais em fatias ou fronteiras.
- Origem de stream line sobre superfícies de fronteira.

c) FlowViz 3D usando objetos integrais

Ao lidar com o fluxo em 3D, uma rica variedade de objetos integrais está disponível para a visualização dos fluxos, desde streamlets até fluxos de volumes, dimensionados com relação às técnicas estendidas a cada um:

- Streamlets em 3D.
- Streamlines em 3D.
- Streamlines iluminadas (figura 2.9).
- Pathlines em 3D.
- Streamribbons (fitas) e stream tubes (tubos)
- Streampolygons.
- Streamballs e streakballs (figura 2.10).
- Streamsurfaces.
- Time Surfaces (superfícies temporais) em 3D.
- Flow Volumes (fluxo de volumes).

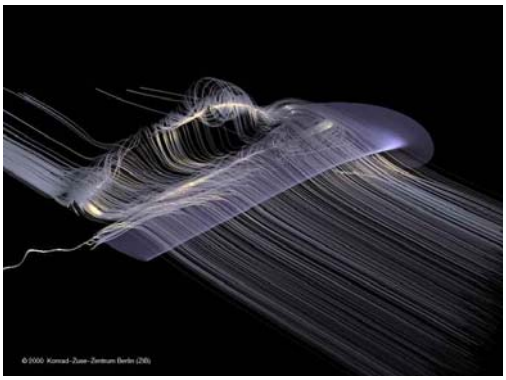


Figura 2.9 – Exemplo de FlowViz usando stream lines iluminadas (fonte: Hauser et al.,2002)

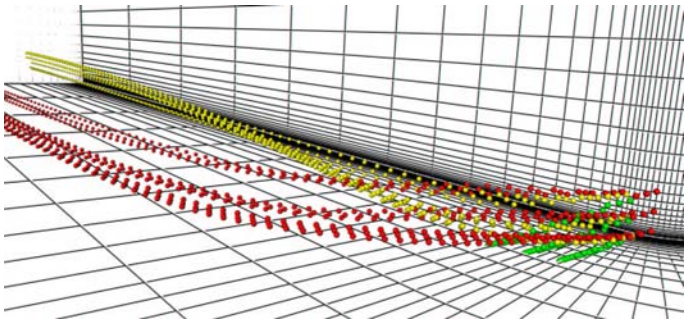


Figura 2.10 – As streak balls mostram o fluxo em um conjunto de dados de uma aleta. As bolas vermelhas são computadas num grid curvilíneo esparso de nível 4, as amarelas em nível 3 e as verdes em nível 2. (fonte: Teitzel e Ertl, 1999)

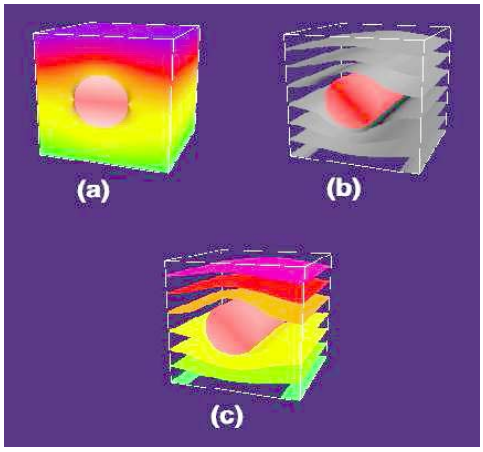


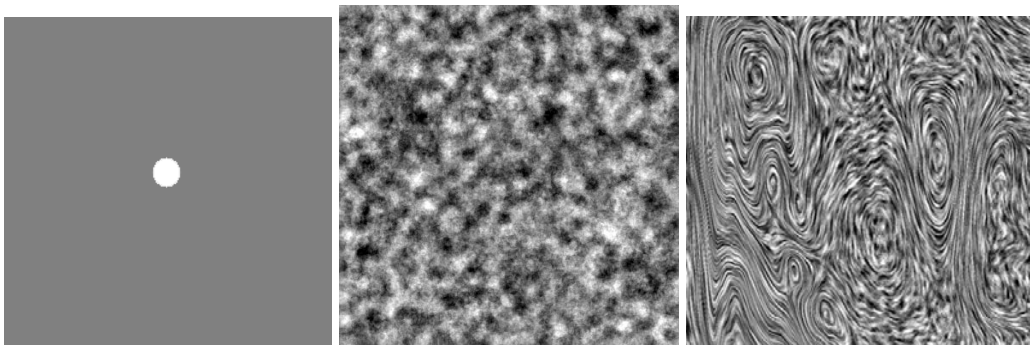
Figura 2.11 – Um campo de fluxo passando através de um cilindro circular. (a) Função principal do fluxo (b) Superfícies principais do fluxo (c) superfícies principais utilizando o mapeamento de cores. (fonte: Cai e Heng, 1997)

FlowViz densa, baseada em integração. Aqui se faz uma distinção entre a visualização de fluxo usando objetos integrantes e visualização de fluxo densa, baseada em integração. No entanto, esses dois temas estão intimamente atrelados: conceitualmente, o caminho da utilização de objetos integrantes para visualização densa, baseada na integração, é obtido por meio de uma estratégia de semeadura densa. Em outras palavras, semeando densamente objetos integrados resultará em uma imagem semelhante à obtida pelas técnicas baseadas em integração. As técnicas de visualização de fluxos, baseadas em integração podem fornecer imagens de resolução espacial densa. Os algoritmos densos, baseados em texturas são eficazes, versáteis e utilizáveis a um amplo espectro de aplicações. Sanna, Montrucchio, e Montuschi (Sanna et al., 2000) apresentam um excelente resumo desta pesquisa em seu artigo.

A FlowViz densa, baseada em integração, divide-se em:

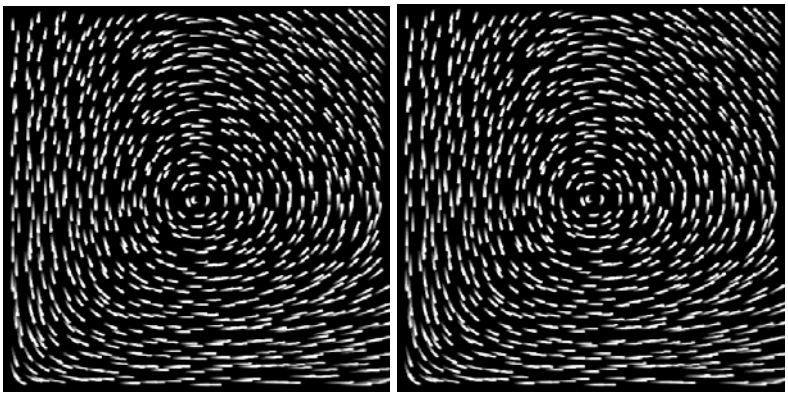
a) FlowViz densa, baseada em integração em 2D. Aqui estão incluídas soluções para FlowViz densa, baseada em integração de dados de fluxos 2D, ou seja, o ruído no local (spot noise), convolução linear integral (LIC) e outra abordagens relacionadas.

- *Spot Noise* (ruído local) (em 2D) (figura 2.12).
- Convolução Linear Integral (LIC) em 2D
- OLIC para FlowViz 2D (figura 2.13).
- Advecção de textura em 2D.
- FlowViz 2D densa baseada em streak lines.



(a) (b) (c)

Figura 2.12 – Princípio do *Spot Noise* : (a) mancha única, (b) textura de spot noise e (c) spot noise usado para visualizar um campo vetorial (fonte : Leeuw et al., 1996)



(a) (b)

Figura 2.13 – Fluxo circular com (a) OLIC e (b) FROLIC (fonte: Wegenkittl e Gröller, 1997).

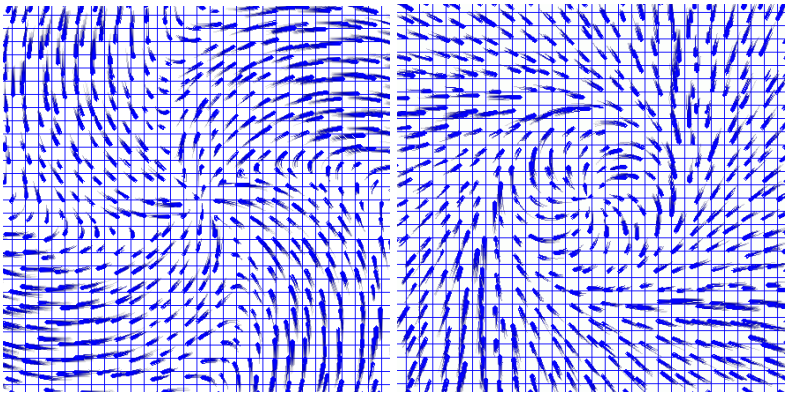


Figura 2.14 – Dois sistemas dinâmicos visualizados pelo uso do método das gotas de tinta virtuais. Um papel quadriculado é usado como fundo. (fonte: Löffelmann et al., 1997)

b) FlowViz densa baseada em integração em superfícies ou fronteiras. As técnicas de visualização densa, baseadas em integração são, em geral, melhores métodos de transmissão da informação sobre cortes transversais que as técnicas usando (longos) objetos integrais. Isto

acontece porque a ligação ao longo do caminho do que seria uma stream line é perdida com as técnicas densas baseadas em integração. Assim, a representação do fluxo não é enganosa em termos de um potencial de sugestões dos caminhos das partículas. Lembrando que o vetor componente ortogonal à fatia é removido quando se usa objetos integrais para a visualização dos resultados.

- *Spot Noise* nas fronteiras ou fatias (figura 2.15).
- LIC para fluxos de fronteira (figura 2.16).
- UFLIC, PLIC, etc.

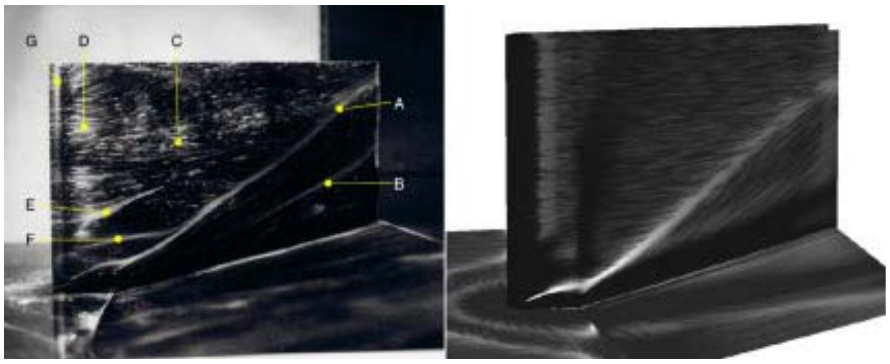


Figura 2.15 – Lado a lado uma comparação entre visualização experimental e spot noise (fonte: Leeuw et al., 1995).

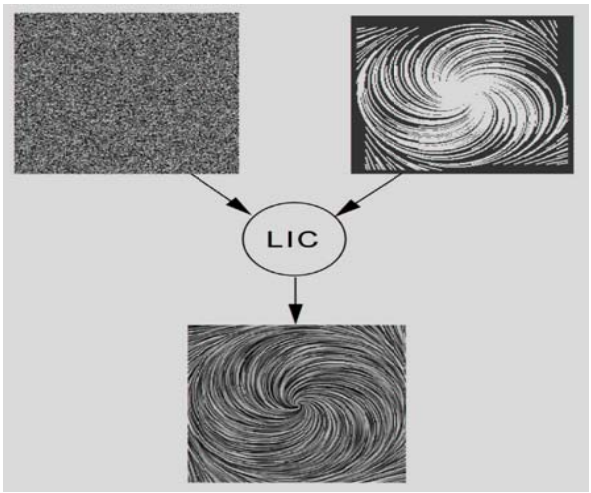


Figura 2.16 – O processo da convolução LIC (fonte : Shen e Kao, 1997).

c) FlowViz 3D densa, baseada em integração. O alto custo computacional, exigentes requisitos de memória, oclusão e complexidade visual podem ser fatores limitantes para a visualização 3D densa de fluxos, baseada em integração.

- LIC em 3D (figuras 2.17, 2.18 e 2.19).
- Advecção de textura em 3D

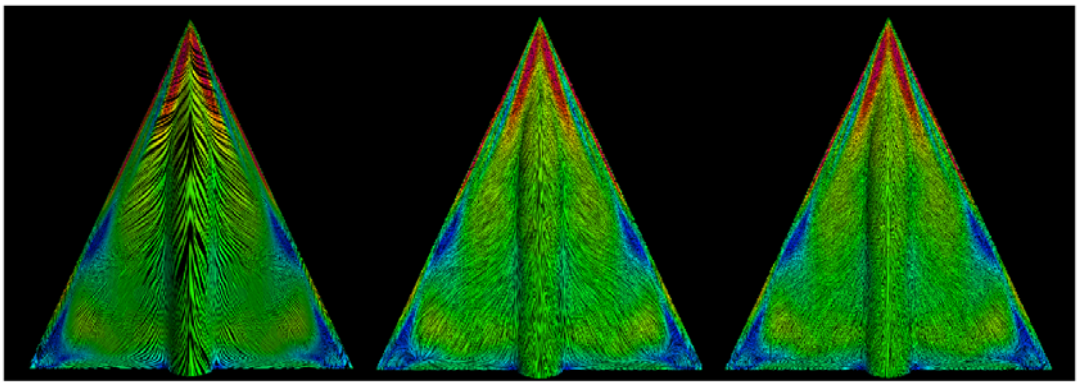


Figura 2.17 – Uma comparação entre três técnicas de LIC: UFLIC (esquerda), ELIC (centro), PLIC (direita) (fonte: Okada e Kao, 1997).

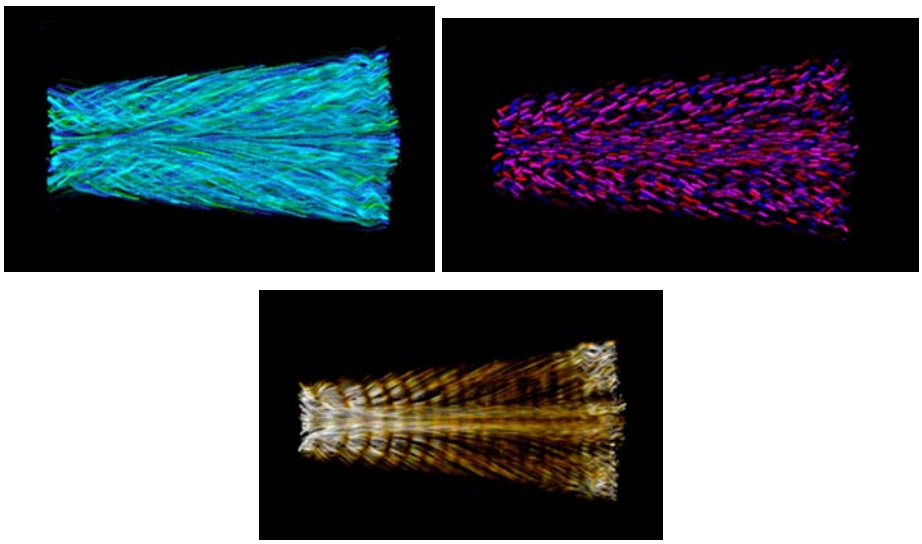


Figura 2.18 – Imagens volumétricas de um fluxo de dados 3D geradas usando LIC 3D. Fonte (Interrante e Grosch, 1997)

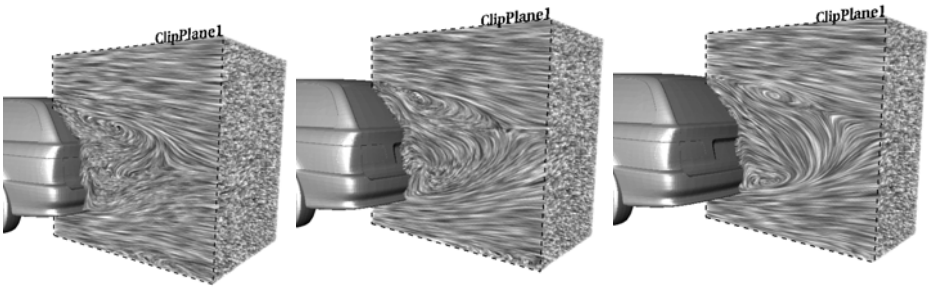


Figura 2.19 – Visualização de LIC volumétrica: O campo de fluxo é explorado usando um plano de recorte, que é mudado de posição interativamente (fonte: Rezk-Salama et al., 1999).

FlowViz baseada em dados derivados. Cálculos computacionais são executados sobre os dados de entrada para a aquisição de dados adicionais sobre a entrada. Estes dados derivados (dependendo da aplicação) podem ser topologia de fluxo, atributos de fluxos, fluxo de dados agregados (através de clustering), dados de fluxo de meta-nível ou outros. Dados originais, bem como os dados obtidos são então utilizados para a visualização, permitindo o reforço da visualização dos dados de fluxo. Em associação com os dados derivados vem a complexidade. As vantagens destas técnicas refletem no lado do usuário: simplificando, o trabalho maior é feito pelo software de visualização e o usuário fica com o menor, por exemplo, menor trabalho com a interpretação.

A FlowViz baseada em dados derivados divide-se em:

a) FlowViz 2D baseada em dados derivados.

- Agrupamento do campo vetorial (figuras 2.20 e 2.21).
- Visualização de fluxos baseada em características (*features)*.
- Visualização de fluxo baseada em topologia.
- Topologia de fluxo multi-resolução.
- Visualização de vórtices (figuras 2.22 e 2.23).
- A precisão da visualização do fluxo baseada em características

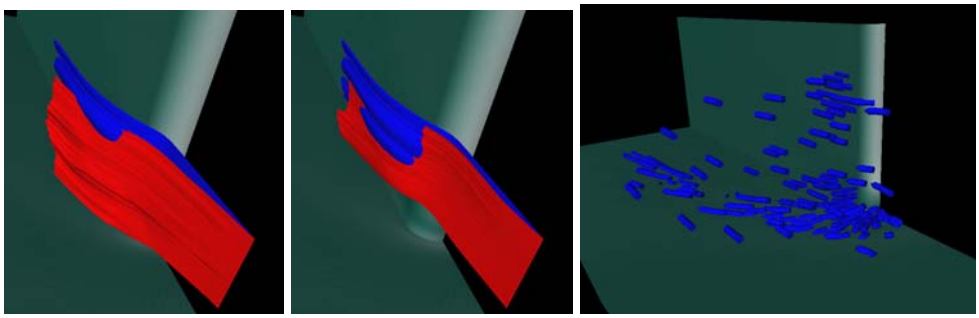


Figura 2.20 – Construção de Hierarquias do campo vetorial (fonte: Hecker et al., 1999)

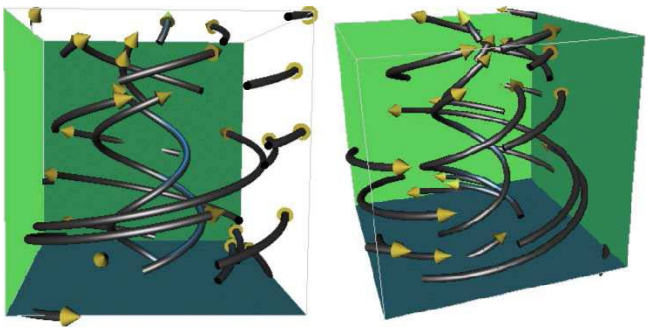


Figura 2.21 – Um campo vetorial 3D visualizado pelo método do agrupamento (fonte: Garcke et al., 2000).

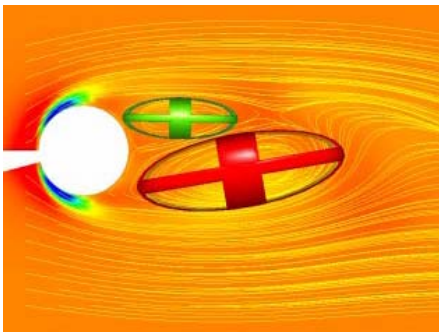


Figura 2.22 - Fluxo passando um cilindro estreito com vórtices aproximados por elipses e streamlines mostrados em uma fatia (fonte: Sadarjoen e Post, 1999).

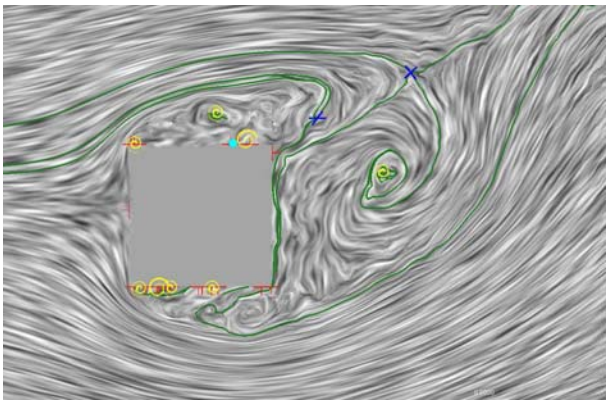


Figura 2.23 – Visualização de uma estrutura de fluxo global de um quadrado. O filtro de distância par é usado com uma distância de 0.0001*H – onde H é a altura do conjunto de dados (fonte: de Leww e van Liere, 1999).

b) Dados derivados para FlowViz em fatias ou fronteiras. Assim como os fluxos de fronteira frequentemente são primeiramente interessantes para a visualização, a extração de dados derivados para visualização de fluxos também é feita por subconjuntos como esse. Um exemplo é a extração automática da união e separação de linhas através das fronteiras dos fluxos, como apresentado por Kenwright (Kenwright, 1998). Wong et al. (Wong et al., 2000) aplicam uma técnica de agrupamento do campo vetorial para fatias 2D sobre dados de um modelo climático regional que abrangem a Ásia Oriental. Eles também mostram como combinar os resultados de várias fatias para obter visualização 3D. Seu objetivo é eliminar os pontos críticos esporádicos e menos interessantes em uma maneira multi-resolução.

c) FlowViz 3D baseados em dados derivados. Quanto mais extenso for um conjunto de dados de um fluxo, a solução mais adequada de FlowViz se baseia em dados derivados. Os dados de um fluxo 3D, especialmente se forem dependente do tempo, muitas vezes são compostos de centenas de milhares de amostras de valores, que (no caso de fluxos não permanentes) são

dados em centenas de etapas no tempo. Claramente, dados de grande extensão representam desafios especiais para a fase de exploração. Com soluções de FlowViz baseadas em dados derivados, o usuário pode ser ajudado a investigar, de forma mais rápida, partes dos dados nas quais está realmente interessado. de Leeuw e van Wijk (de Leeuw e van Wijk, 1993), por exemplo, apresentam uma técnica de análise de fluxo interativo em 3D, que se baseia na visualização dos dados obtidos localmente. Atributos locais do fluxo, tais como velocidade, aceleração, convergência ou divergência, rotação, etc., são visualizados através da utilização de glifos, que mudam sua aparência (geométrica) de acordo com os dados a serem visualizados.

- FlowViz 3D baseada em extração de características.
- Monitoramento de características.
- Foco e contexto da visualização em 3D.
- Visualização de fluxo 3D baseada na topologia.
- Visualização de Vórtices em 3D.
- FlowViz 3D utilizando mapas de Poincaré.

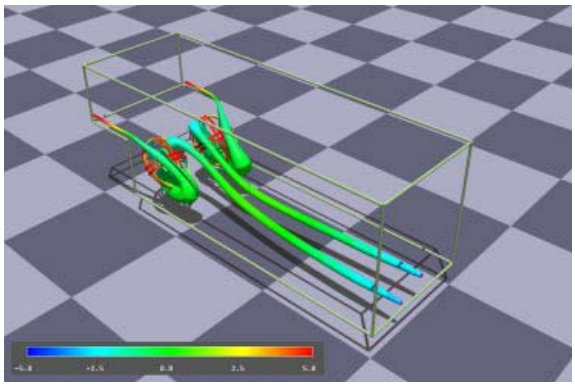


Figura 2.24 - Regiões com elevada densidade helicidal normalizada e stream tubes através dessas regiões (fonte: Van Walsum et al., 1996).

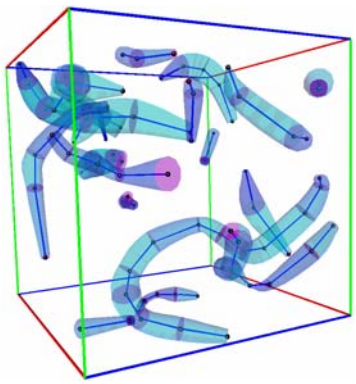


Figura 2.25 – Reconstrução de uma superfície esqueletal (fonte: Reinders et al., 2000).

Figura 2.26 – Simulação de um modelo de turbina Kaplan. Este estudo de caso descreve a utilização da visualização do resultado de simulação CFD para o projeto de turbinas (fonte: (Roth e Peikert, 1996).

Para uma representação apropriada de um fluxo é necessário uma compreensão profunda do problema físico para que se possa escolher, dentro da classificação apresentada pelos autores (Weiskoof e Erlebacher, 2004) (Hauser et AL., 2002), a que melhor visualiza o fenômeno que se deseja observar, seja em duas ou três dimensões.

2.2 Softwares CDF Comerciais Existentes

A visualização de escoamentos sobre objetos por ser obtida a partir de aplicativos existentes no mercado, de código aberto ou em pacotes comerciais. Para avaliar um software CFD como ferramenta são várias as considerações que devem ser tomadas, entre elas, a amplitude dos problemas físicos que o pacote pode resolver, a facilidade de uso, a capacidade de montar a geometria e gerar malhas, a eficiência e a robustez das técnicas numéricas implementadas. Esses fatores dependem muito do grau de desenvolvimento e do investimento colocado no aplicativo.

Sem dúvida alguma, atualmente os pacotes CFD mais desenvolvidos são os comerciais, como CFX (CFX, 2010), FLUENT (Fluent, 2010) e PHOENICS (Cham, 2010), justamente devido ao seu apelo comercial. Existem muitos códigos CFD disponíveis gratuitamente, contudo normalmente estes possuem uma área de aplicação limitada. Uma ferramenta CFD nunca será geral suficiente para atender, por exemplo, o fluxo de estrelas em galáxias e os fluxos de uma mistura laminar em nano-canais. Os modelos, utilizados pelas diversas ferramentas, representam e modelam uma parte da realidade e, com uma boa escolha

se pode reproduzir processos importantes para melhorar um desenho técnico ou fazer previsões de tempo.

Os pacotes CFD, pagos ou não, tipicamente requerem do usuário um grande esforço dedicado à compreensão e ao seu uso correto. Os códigos CFD irão fornecer uma resposta à maioria dos problemas quando estes forem apropriadamente colocados, porém é necessário paciência e experiência para produzir resultados razoáveis. A interface de integração com o usuário oferecida pelos pacotes CFD é algo que sempre deve ser considerado ao escolher um software. Além disso, os códigos para solução de problemas CFD necessitam de ferramentas de geração de malha e visualização científica (pré e pós-processamento) de modo a se tornarem úteis. A maioria dos códigos CFD livres não inclui estas duas ferramentas, mas existem vários programas específicos para geração de malhas e visualização de dados, contudo o usuário deve despender de um grande esforço para interligar estes com o código CFD.

De acordo com (Cook, 1998) os recursos de um software são divididos em seis grupos:

- **Capacidade de pré-processamento** – É a interface através da qual o usuário define a simulação. Inclui a geometria e malha e permite a especificação das condições de contorno, os modelos físicos e parâmetros numéricos utilizados no processo de solução. As características procuradas no pré-processador são:

  - Modelagem 2D e 3D;
  - Sistemas de coordenadas cartesianas e cilíndricas;
  - Capacidade de malha de grande porte (acima de 100000 células). Este limite é muitas vezes ditado pelo hardware disponível;
  - Especificação das condições de contorno complexas, tais como condições transientes;
  - Acesso às rotinas de programação para adicionar ou modificar modelos físicos.

- **Capacidade física de modelagem** - Identifica os métodos numéricos desejados para representar os vários aspectos do fluxo de ar. Além da transferência de massa, momento e energia, os seguintes recursos são procurados:

  - Modelagem do estado transiente e permanente;
  - Seleção de modelos de turbulência, incluindo o k-ε padrão - e modelo de Launder e Spalding;
  - Rastreamento de partículas - partículas de fumaça, partículas de poeira, e "nuvens" de partículas;
  - Modelagem multi-fase, por exemplo, evaporação da água no ar;

- Tratamento preciso da camada limite.

- **O solver** - Uma vez que os vários modelos físicos foram especificados e as condições de contorno definidas, é o trabalho do solver a "organizar" toda essa informação e resolver as equações que regem o fluxo para encontrar valores para todas as variáveis em cada célula da malha definida pelo usuário de tal forma que os modelos físicos e condições de contorno são simultaneamente satisfeitas. Ao resolver as equações é necessário reduzi-las a um formato numérico que pode ser entendida por um computador. Essa técnica é chamada de discretização. Existem três métodos principais para isso: o Método de Diferenças Finitas (FDM), o Método dos Elementos Finitos (MEF) e o Método dos Volumes Finitos (MVF).
- **Recursos para pós-processamento** - Este é o aspecto do pacote CFD dedicado à análise dos resultados. Os códigos CFD produzem grandes quantidades de dados em arquivos de seus resultados. A forma mais rápida e eficaz para ver estes dados é graficamente. A maioria dos pacotes CFD podem produzir contornos (contornos de linha e os contornos sombreados), gráficos de linhas e vetores. Alguns pacotes também permitem a sobreposição de duas dessas variáveis, por exemplo, vetores de velocidade e contornos de temperatura. Embora a análise gráfica seja provavelmente o método mais útil para ver os resultados, é importante verificar se há ou não o pacote dá acesso aos resultados em formato ASCII. Isto é muito útil para a comparação dos resultados com dados experimentais ou outros códigos CFD. Também pode ser útil se o software pode interagir com outros pacotes de visualização.
- **Facilidade de utilização** - A facilidade com que o usuário é capaz de operar o software é determinada pelo design da interface do usuário. Uma boa linguagem de comando oferece aos usuários mais liberdade na definição de geometrias, malhas, condições de contorno, na definição de novos modelos físicos, na especificação das condições de contorno complexas, definição de variáveis de saída adicionais etc.
- **Suporte ao usuário** - Ao usar o software CFD para resolver problemas difíceis, é útil se um bom suporte ao usuário está disponível através de um método imediato de resposta, como e-mail, fax ou telefone. Alguns fornecedores também criam grupos de e-mail que permitem aos usuários enviar solicitações e conselhos para todos os usuários atuais do referido código CFD. A maioria das empresas oferecem cursos de formação para novos e potenciais usuários que lhes permitem ultrapassar algumas das dificuldades iniciais e obter o máximo do software em um curto período de tempo.

Alguns pacotes comerciais foram escolhidos para comparação e estão listados a seguir:

- *ANSYS FLUENT Flow Modeling Software*

De acordo com seu fabricante o software ANSYS FLUENT contém grandes capacidades de modelagem física necessária para modelo de fluxo, turbulência, transferência de calor e reações para aplicações industriais que vão de fluxo de ar sobre uma asa de avião a combustão em um forno, a partir de colunas de bolhas de plataformas de petróleo, de fluxo de sangue para semicondutores fabricação e design de sala limpa para estações de tratamento de águas residuais. Os modelos especiais que dão ao software a capacidade de modelar a combustão de cilindros, aeroacústica, turbomáquinas, sistemas multifásicos têm servido para ampliar o seu alcance.

Ainda, conforme é relatado no site, milhares de empresas em todo o mundo se beneficiam do uso do software como parte integrante de seu projeto e otimização de fases de desenvolvimento do produto. A avançada tecnologia de resolução proporciona resultados rápidos e precisos para CFD, com malhas flexíveis, móveis e deformáveis. Funções definidas pelo usuário permitem a implementação de modelos de usuário novo e a ampla personalização dos já existentes. O conjunto interativo de resolução, solução e pós-processamento tornam mais fáceis a interrupção de um cálculo, análise dos resultados com pós-processamento integrado, alteração de qualquer configuração, e depois continuar o cálculo dentro de uma única aplicação. Processos e arquivos de dados também podem ser lidos em ANSYS CFD-Post para posterior análise com ferramentas avançadas de pós-tratamento e comparação dos resultados de diferentes casos lado a lado (Fluent, 2010).

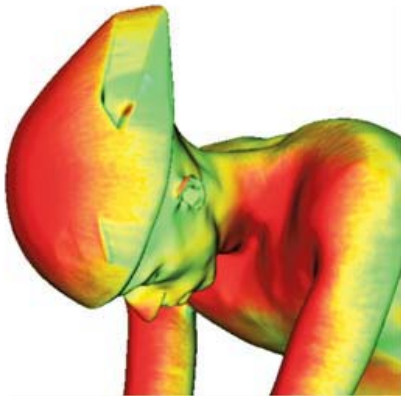


Figura 2.27 - Contornos de pressão estática em um capacete de bicicleta (fonte: Fluent, 2010).

- *ANSYS CFX*

A empresa ANSYS cita que seu pacote CFX é um programa de alto desempenho e de propósito geral CFD que é aplicado para resolver uma ampla variedade de problemas de fluxos de fluidos. Na proposta geral está uma tecnologia avançada de resolução, chave para alcançar soluções confiáveis e precisas de forma rápida e robusta. De acordo com a empresa, a solução moderna e altamente paralelizada é a base para uma escolha abundante de modelos físicos para capturar praticamente qualquer tipo de fenômenos relacionados com a dinâmica de fluidos: de laminar a turbulento (incluindo os de transição), incompressível até plenamente compressível, subsônico até trans e supersônico, isotérmicos ou com transferência de calor por convecção e/ou radiação, sem reação à combustão, estacionário e/ou rotativo, fluidos simples e misturas de fluidos em uma ou mais fases (incluindo superfícies livres) (CFX, 2010).

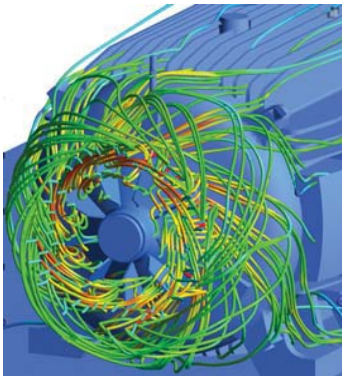


Figura 2.28 - Previsões de fluxo de ar de refrigeração para otimizar a transferência de calor em torno de motor elétrico (fonte : CFX, 2010)

- *ANSWER*

Em texto apresentado em seu site, a empresa ACRi – Analytic &Computational Research, Inc. cita que desenvolve há 25 anos o programa ANSWER. Este tem evoluído de um modelo matemático simples de fluxo e transferência de calor para uma ferramenta computational de Dinâmica de Fluidos para análise de uma vasta gama de problemas. Ele prevê o transporte de líquido acoplado, calor e várias espécies químicas em uma complexa geometria tridimensional. É capaz de simular o comportamento do estado transiente ou estacionário de fluxo laminar ou turbulento de fluidos compressíveis ou incompressíveis com reações químicas, combustão, sprays líquidos, queima de gotas, a formação de fuligem e radiação. Passou de um código de computador simples, com alguns módulos e poucas opções

a um pacote de software com quase 1000 módulos e um conjunto versátil de opções que podem acomodar quase todas as necessidades dos utilizadores.

De acordo com a empresa o software tem sido usado para analisar problemas tão diversos como a deposição de filmes de baixa pressão (microeletrônica), a refrigeração de componentes eletrônicos, dinâmica dos fluidos nos automóveis, a ventilação dos túneis, análise de tanques de processo, projeto de ramjets e motores de aeronaves, simulações de mísseis em tubo de lançamento, lubrificação dos rolamentos e desempenho de mísseis com elevado número de Mach.

No processo, a empresa relata que o software ANSWER tem evoluído com as necessidades. O usuário pode alterar os esquemas numéricos, método de solução, algoritmos de inversão de matrizes, ou qualquer das características físicas ou matemáticas.

As informações fornecidas pelo site cita que o aplicativo distingue-se pela diversidade de seus usuários. Ele está sendo usado por uma pesquisa comercial e organizações educacionais em 6 países. Entre seus usuários estão Aerospatiale (França), Allison turbina a gás, ASCI, SA (Espanha), BAe-Sema (UK), CNIM (França), Ministério da Educação (México), GERPY (FRANÇA), General Electric Company, James M. Montgomery, Lam Research Corporation, Marquardt Empresa Nacional de Aeronáutica e Espaço, a Renault (França), a SNECMA (França), Universidade de Califórnia, E.U. Força Aérea, Watkins-Johnson, WS Atkins (Reino Unido), e uma série de outras organizações comerciais. Mais de 100 publicações e relatórios do projeto na aferição, verificação e aplicação de ANSWER estão atualmente disponíveis (ANSWER, 2010).

O software ANSWER gera resultados numéricos que podem ser visualizados por outro programa da empresa ACRi, o CFDStudio (figura 2-29).

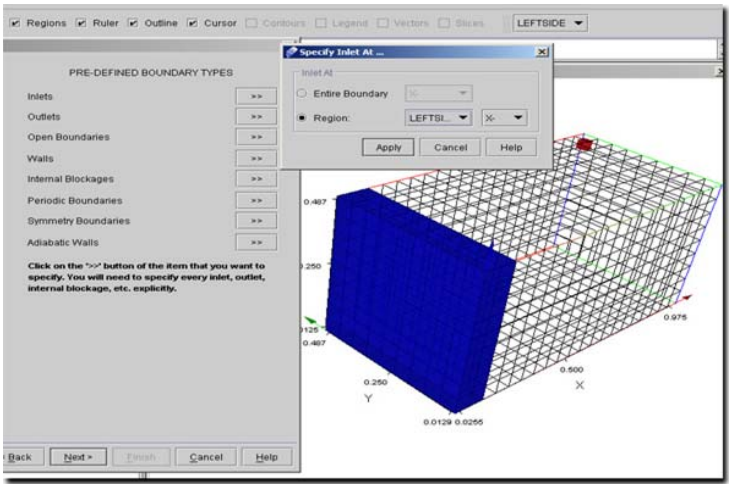


Figura 2.29 – Ambiente de visualização CFDStudio (fonte: ANSWER,2010).

- *Flowmaster*

Flowmaster é uma empresa internacional de desenvolvimento de softwares e serviços de simulação de sistemas termo-fluidos que proporciona a simulação da Dinâmica dos Fluidos Computacional (CFD) em nível de sistema, dando aos engenheiros, segundo a empresa, uma visão desde o estágio conceitual do processo de desenvolvimento até toda a extensão do projeto.

As informações contidas no site do fabricante citam que Flowmaster V7 é uma ferramenta de software para simulação utilizado pelas empresas através de uma ampla gama de indústrias para reduzir o tempo de desenvolvimento e custos de seus sistemas de termo-fluidos. Ela permite aos engenheiros simular rapidamente sistemas de fluidos compressíveis e incompressíveis para entender como as alterações de design, as condições de funcionamento, dimensionamento de componentes e seleção e outras variáveis que afetam o desempenho geral do sistema. As empresas podem maximizar seu retorno sobre o investimento através da integração Flowmaster V7 em todas as fases do processo de desenvolvimento, aproveitando-se da gestão de dados e capacidades de colaboração desta ferramenta analítica (Flowmaster, 2010).

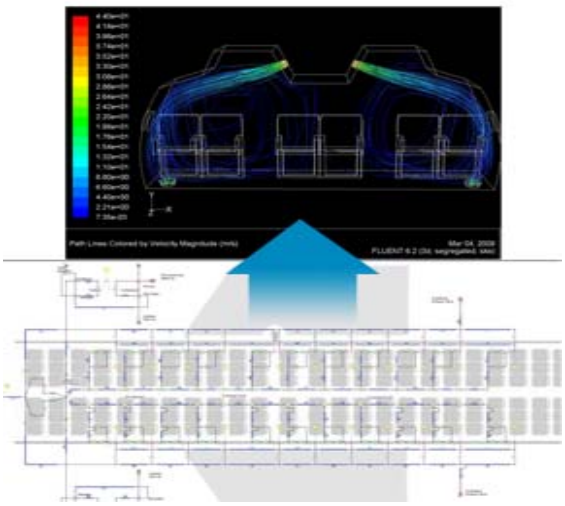


Figura 2.30 Flowmaster 1D e Co-simulação 3D (fonte: Flowmaster, 2010)

- *PHOENICS*

CHAM é uma empresa localizada em Wimbledon, na Inglaterra. Seu nome completo é Concentration Heat and Momentum Ltd. De acordo com sua propaganda, seu negócio é, e tem

sido desde a década de 1970, o fornecimento de software e serviços relacionados, preocupados com a simulação computacional de processos envolvendo o fluxo de fluidos, em conjunto com as tensões e temperaturas em sólidos em contato com eles.

PHOENICS é o seu principal produto de software. É um pacote de uso geral, lançado primeiramente em 1981 e ainda pioneiro de novos recursos (figura 2-31). Dentro das informações fornecidas pela empresa o software se diferencia de seus concorrentes pelo destaque que dá à RDI, ou seja, Relational Data Input (entrada de dados relacionais).

Conforme a empresa cita, para habilitar os usuários a explorar o poder da RDI, PHOENICS foi fornecido com um novo módulo front-end, PRELUDE, e este, em conjunto com um setor especial "Gateways"', permite aos utilizadores introduzirem facilmente os dados que de seu especial conhecimento que entendem necessários, sem ter que atender às questões de seu desinteresse. Além disso, ainda de acordo com a empresa, considerando que convencionalmente tem sido, para problemas de interação fluido-estrutura, necessariamente obrigatório contratar dois pacotes de software distintos, um para o fluxo de fluido e o outro (muitas vezes baseado em elementos finitos) para as tensões em sólidos, PHOENICS manipula as duas tarefas simultaneamente (Cham, 2010).

Figura 2.31 Fluxo através de edifícios do complexo Juberiah Beach – Dubai (fonte: Cham, 2010).

- *Fluidyn*

De acordo com a Transoft International, organização de 20 anos, a empresa fornece uma gama de produtos e serviços de consultoria. A teoria de Fluidyn é que todos os acidentes podem ser compreendidos se considerarmos relação entre o material, energia e movimentos.

Embora se saiba que esses três podem ser convertidos para qualquer um dos outros dois, é possível avaliá-los individualmente a qualquer instante. Qualquer mudança que prejudique o interesse dos homens ou a vida pode ser considerada um acidente. Para explicar essas mudanças usa-se o conhecimento da Fluidodinâmica, Aeronáutica, Hidráulica, Engenharia Civil, Mecânica e Química.

A empresa cita que o software Fluidyn (figura 2-32) tenta definir um acidente, enquanto ele ocorre, por causa dos movimentos no ar, água ou sólidos dentro de um determinado intervalo de tempo. Uma vez que a posição atual é entendida, usando algumas leis da física computacional (FisComp, 2010), tenta-se prever o futuro.

Áreas de provável acidente, onde Fluidyn pode ser útil para efeito de previsão, variam desde edifícios ou incêndios florestais até explosões, impactos ambientais (ar/água/subsolo), poluição rodoviária, risco industrial, aplicação aeronáutica, motores de automóveis/aeronaves, energia nuclear e de transferência de calor, etc. (Fluidyn, 2010).

A Fluidyn possui em torno de 25 pacotes aplicados a cada área de interesse onde a fluidodinâmica pode ser aplicada.

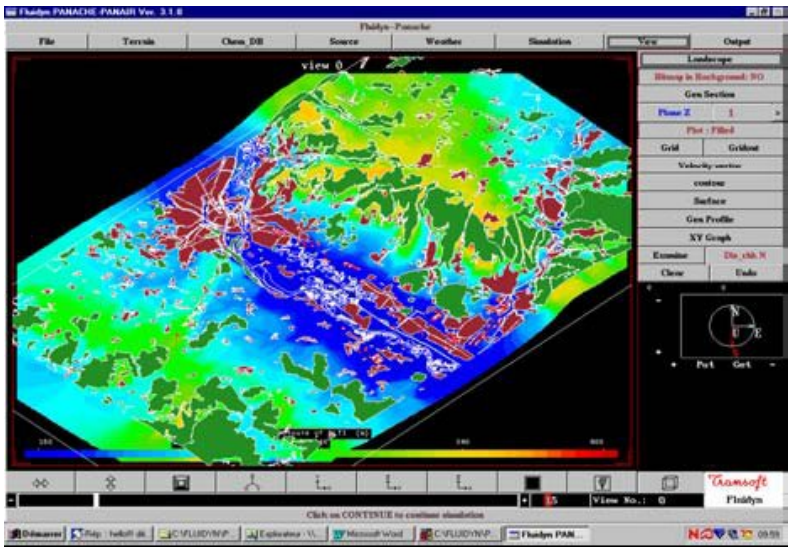


Figura 2.32 Simulação da dissipação da poluição atmosférica urbana usando o Fluidyn PANAIR (fonte Fluidyn, 2010).

- *OpenFOAM*

A OpenFOAM (Open Field Operation e Manipulation) CFD Toolbox é um pacote de software de código aberto de CFD produzido por uma empresa comercial, OpenCFD Ltd. Ela tem uma grande base de usuários das áreas da engenharia e da ciência, das organizações

comerciais e acadêmicas. Conforme relata a empresa, o software OpenFOAM possui uma extensa gama de recursos para resolver tudo, desde fluxos complexos de fluidos que envolvam reações químicas, turbulência e transferência de calor, a dinâmica de sólidos e eletromagnetismo. O núcleo da tecnologia do OpenFOAM é um conjunto flexível de módulos C++ eficientes. Estes são usados para construir uma riqueza de: *solvers* para simular problemas específicos em engenharia mecânica; *utilitários*, para realizar pré e pós-processamento de tarefas que vão desde simples manipulações de dados para visualização e processamento de malha; *bibliotecas*, para criar caixas de ferramentas que sejam acessíveis aos solvers/utilitários, tais como bibliotecas de modelos físicos.

OpenFOAM é fornecido com vários solvers, utilitários e bibliotecas pré-configuradas e assim pode ser usado como qualquer outro pacote típico de simulação. No entanto é aberto, não só em termos de código-fonte, mas também na sua estrutura e concepção hierárquica, de modo que seus solvers, utilitários e bibliotecas são totalmente extensíveis. Ele usa o método numérico de volumes finitos para resolver sistemas de equações diferenciais parciais atribuídas em qualquer malha 3D não estruturada de células poliédricas. Os solvers de fluxo de fluidos são desenvolvidos dentro de uma estrutura robusta, implícita, solução iterativa para pressão-velocidade, entretanto algumas técnicas alternativas são aplicadas a outros solvers de mecânica do contínuo. O paralelismo da decomposição do domínio é fundamental para o projeto do OpenFOAM, integrado a um nível baixo de modo que solvers geralmente podem ser desenvolvidos sem a qualquer necessidade de codificação paralela específica (FOAM, 2010).

- *Gerris Flow Solver*

Gerris (figura 2-33) é um programa de software livre para a solução das equações diferenciais parciais que descrevem o fluxo de fluido. O código fonte está disponível gratuitamente sob a licença Free Software GPL. Gerris é apoiado por NIWA (National Institute of Water and Atmospheric Research).

De acordo com o fabricante, como um breve resumo das suas principais características, tem-se: Resolve equações de Euler, Stokes e Navier-Stokes, dependentes de tempo, incompressíveis e de densidade variável; resolve as equações superficiais lineares e não lineares; refinamento da malha adaptativa: a resolução é dinamicamente adaptada às características do fluxo; geração totalmente automática de malha em geometrias complexas;

suporte paralelo utilizando a biblioteca MPI, balanceamento de carga dinâmica; modelo preciso modelo de tensão superficial (Gerris, 2010).

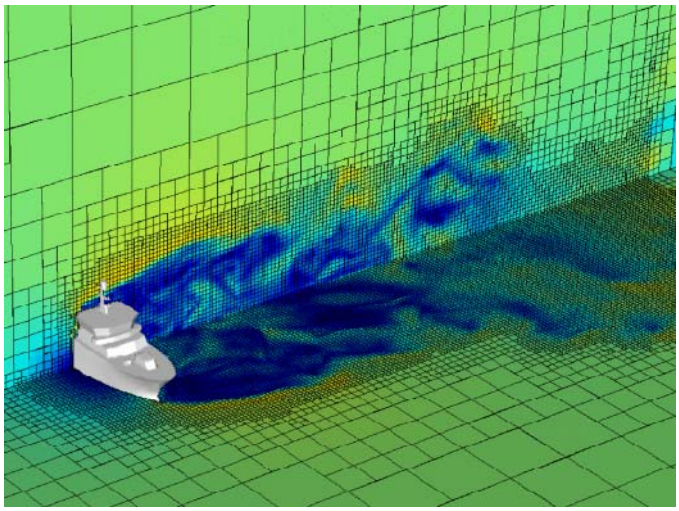


Figura 2.33 Quadro de filme MPEG da malha adaptativa de um fluxo de vento turbulento. As duas seções são coloridas de acordo com a norma do vetor de velocidades. Tempo de processamento: 7 horas (fonte: Gerris, 2010).

2.2.1 Breve comparação entre os programas existentes

A tabela 2.1, apresentada a seguir, mostra uma breve comparação entre os aplicativos mostrados neste capítulo, citando algumas características de cada um. Um dos pontos principais é o fato de que, apesar de possuírem métodos semelhantes de resolução, com modelos avançados de malhas e cálculos de fluxos. Nenhum deles apresentam a possibilidade de interação em tempo real.

	Fluent	**CFX**	**ANSWER**	**Flowmaster**	**PHOENICS**	**Fluidyn**	**OpenFOAM**	**Gerris Flow Solver**
Fabricante/ Desenvolvedor	ANSYS	ANSYS	ACRi	Flowmaster	CHAM	Transoft International	OpenCFD Ltd.	Gerris
Plataforma	Windows e Linux (arquiteturas x86 e x86-64)	Windows e Linux (arquiteturas x86 e x86-64)	Windows	Windows	Windows	Windows, LINUX, SCO Unix. Silicon Graphics workstations (SG IRIX) Sun (Ultra) SPARC workstations (Sun Solaris) IBM workstations (IBM AIX) Digital workstations (DEC alpha) HP workstations (HPUX) CRAY supercomps	32 bit Linux, 64 bit Linux.	Ubuntu, debian, Open SUSE, Fedora, Mac OSX.
Interfaces geométricas	CAD	CAD	ANSYS (via conversor); CFDStudio	ND	ND	Fluidyn CAD	ND	ND
Malha Dinâmica	sim	sim	ND	ND	Não	ND	Sim	Não
Turbulência	modelos k–epsilon, k–omega e Reynolds stress model (RSM)	modelos k-ε, k-ω, SST, RSM, scaleresolving LES e DES	ND	ND	baixos nº de Reynolds; LVEL, k-ε, RNG, de duas escalas e duas camadas.	Sim	modelos RAS para fluidos compressíveis e incompressíveis. Large eddy simulation (LES) turbulence.	Não
Pós Processamento	ANSYS CFD-Post	ANSYS CFD-Post	ND	Sim	ND	Fluidyn CADGEN	foamDataToFluent, foamToEnsight, foamToFieldview9, foamToGMV, foamToTecplot360, foamToVTK.	GfsView
Fluxo permanente	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Fluxo Transiente	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Fluxo 2D	Sim	Não	Sim	ND	Sim	Sim	Sim	Sim
Fluxo 3D	Sim	Sim	Sim	ND	Sim	Sim	ND	Sim
Visualização 3D	Sim	Sim	Não	Não	Sim	Sim	Para View	Sim
Interação em tempo real	**Não**	**Não**	**Não**	**Não**	**Não**	**Não**	**Não**	**Não**
Gratuito	Não	Não	Não	Não	Não	Não	Sim	Sim

Tabela 2-1 Breve comparação entre softwares comerciais de cálculo de CFD (Legenda: ND – não divulgado).

3 MODELO CFD NUMÉRICO PARA A EQUAÇÃO DE LAPLACE 3D

Tem-se a seguir um detalhamento do esquema numérico a ser adotado para a solução da equação tridimensional linear estudada (equação de Laplace).

3.1 Dinâmica dos Fluidos Computacional (CFD)

Há muitos séculos o movimento dos fluidos vem sendo estudado. Os egípcios possuíam relógios de água; Aristóteles descreveu o princípio da continuidade; Arquimedes definiu as condições para que um corpo, quando mergulhado em um fluido, flutuasse ou não; os romanos construíram aquedutos para transportar água para as suas cidades; Leonardo da Vinci sugeriu formas que reduziam o arrasto de barcos na água. Em 1586, Simon Stevin publicou *Estática e Hidrostática*, um tratado matemático sobre a mecânica dos fluidos como era conhecida até então (Fortuna, 2000).

Inicialmente a mecânica dos fluidos foi estudada de forma experimental, surgindo a hidráulica e, posteriormente, com a matemática, surgiu a hidrodinâmica. Leonard Euler foi quem deduziu, primeiramente, as equações de movimento dos fluidos. Mas somente no Século XIX, Claude Navier (1822), Simeon Poisson (1829) e George Stokes (1845) deram força às descrições matemáticas do comportamento dos fluidos e publicaram as *equações de Navier-Stokes* (Fox e McDonald, 2001).

As soluções analíticas para as equações de Navier-Stokes foram determinadas para poucos casos, devido ao fato de que são equações diferenciais parciais, não lineares, e a teoria matemática não está suficientemente desenvolvida para permitir a obtenção de resultados em regiões arbitrárias e condições de contorno gerais.

Por esta razão, utilizam-se ensaios experimentais como testes em túneis de vento e tanques de água no estudo do movimento dos fluidos. Porém, devido às limitações de custo, tempo e equipamentos costuma-se reduzir o estudo a apenas alguns pontos da região em que ocorrem os fenômenos de interesse. Aliado a isso, nem sempre um fenômeno é passível de reprodução em laboratório, mesmo em escala reduzida, como, por exemplo, a previsão do tempo.

O uso de uma solução numérica tomou força, a partir dos anos 1950, com o avanço da computação. Problemas reais de engenharia muitas vezes requerem um tratamento computacional, por ser a forma mais prática, ou única, de se obter dados de um escoamento.

A *Dinâmica dos Fluidos Computacional*, ou CFD (*Computacional Fluid Dynamics*, em inglês) é a área da computação científica que estuda métodos computacionais para a simulação de fenômenos que envolvem fluidos em movimento com ou sem troca de calor.

O objetivo básico da CFD é reduzir o número de experimentos e explorar fenômenos que não poderiam ser estudados em laboratório de forma prática.

Apesar de toda a flexibilidade que a técnica computacional oferece, ela ainda não pode resolver muitos problemas reais que envolvem o escoamento de fluidos. O exemplo mais direto é a modelagem de escoamentos turbulentos, em que o uso de diferentes modelos de turbulência, em um mesmo tipo de escoamento, pode fornecer resultados distintos. As próprias equações de Navier-Stokes, quando tratadas numericamente, têm comportamentos diferentes daqueles que se esperam das equações originais. Essas alterações, se não forem levadas em consideração, podem distorcer a solução do problema.

Uma classificação da Mecânica dos Fluidos pode ser vista na figura 3.1:

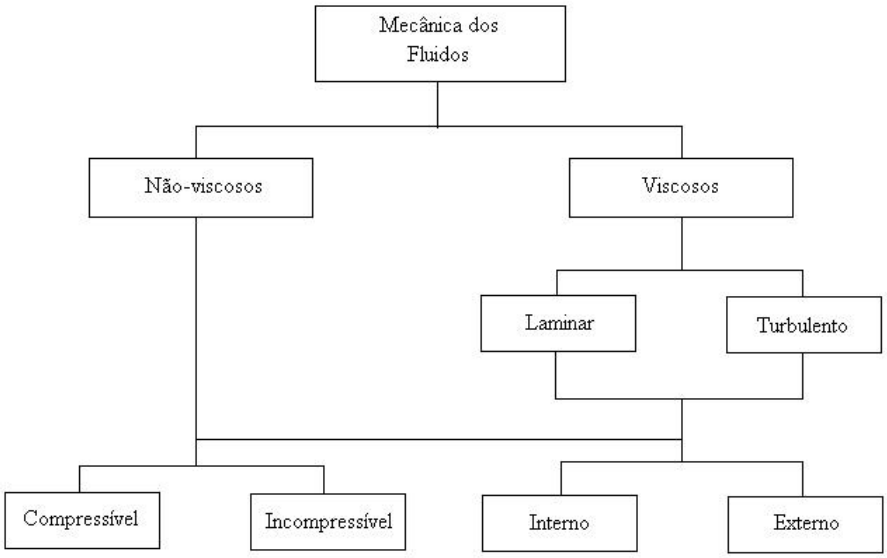


Figura 3.1 – Classificação possível da Mecânica dos Fluidos (fonte: Fox e McDonald, 2001)

Os escoamentos nos quais as variações das densidades dos fluidos são desprezíveis denominam-se *incompressíveis.* Os gases comportam-se como fluidos incompressíveis desde que as velocidades dos escoamentos sejam pequenas em relação à velocidade do som. A razão entre velocidade, V, e a velocidade local, c, do som, no caso dos gases, é denominada número Mach, $M \equiv \frac{V}{c}$. Para valores de M<0,3 as variações das densidades são menores do que 5 por cento. Para M=0,3 estando o ar nas condições padrão, a velocidade é de, aproximadamente, 100 m/s (Fox e McDonald, 2001).

Uma grandeza adimensional importante na mecânica dos fluidos é o *número de Reynolds (Re)*. Este expressa a razão entre as forças inerciais (devido à velocidade) e as forças viscosas em um escoamento. A magnitude do número de Reynolds indica a importância para o escoamento das forças inerciais (Re>1) e das forças viscosas (Re <1). Quando Re>>1, as forças viscosas são importantes somente nas regiões adjacentes às superfícies sólidas, devido à presença da camada limite. Os escoamentos abaixo de um valor $Re_{crítico}$ são laminares; acima deste valor os escoamentos se tornam turbulentos (Fortuna, 2000).

No escoamento de fluidos não-viscosos, a viscosidade, μ, do fluido é supostamente nula. Tal fluido não existe. Contudo há vários problemas nos quais a hipótese de μ=0 simplifica a análise e, ao mesmo tempo, conduz a resultados satisfatórios (Fox e McDonald, 2001).

Este trabalho baseia-se nos conceitos de Dinâmica dos Fluidos Computacional em três dimensões. Para reduzir o esforço computacional, o desenvolvimento numérico do sistema foi feito considerando um fluxo em regime permanente, não-viscoso, incompressível e irrotacional. Isto simplifica as equações de Navier-Stokes e torna o processamento mais rápido. Na análise numérica também não foram considerados os efeitos de turbulência e variações de temperatura (Carvalho et al., 2005). Desse modo, a equação simplificada que governa a simulação tridimensional é (Fortuna, 2000):

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2} = 0 \qquad (1)$$

Sendo que:

ϕ é o potencial de velocidade do fluxo;

x, y e *z* são as coordenadas cartesianas.

Essa equação é conhecida na literatura como *Equação de Laplace*, que representa a lei da conservação da massa. Trata-se de uma equação diferencial parcial elíptica escrita, neste trabalho, em coordenadas cartesianas. Em geral, equações desse tipo representam problemas de equilíbrio, em que a propriedade de interesse não se altera com o passar do tempo. De uma maneira geral, qualquer escoamento irrotacional e incompressível é governado pela equação de Laplace (Greenberg, 1998).

O potencial de velocidade se relaciona com as componentes de velocidade v_x, v_y e v_z através das equações:

$$v_x = \frac{\partial \phi}{\partial x} \tag{2}$$

$$v_y = \frac{\partial \phi}{\partial y} \tag{3}$$

$$v_z = \frac{\partial \phi}{\partial z} \tag{4}$$

A solução única para esse e outros problemas envolvendo equações diferenciais parciais é obtida especificando-se condições para a variável dependente na fronteira ∂R da região *R*, em que se quer resolver o problema. Os problemas que exigem condições ao longo da fronteira (contorno) ∂R de toda a região são denominados *Problemas de Valor de Contorno* (PVC) (Fortuna, 2000).

Computacionalmente, no caso das técnicas numéricas de solução, não é possível tratar a região *R* como contínua, pois o método numérico obtém a solução em pontos (*x, y, z,* por exemplo) por cálculos como adição e multiplicação. Pode-se, então, escolher alguns pontos dentro de *R* e *somente* neles calcular a solução do problema. Este é o processo de discretização.

Este trabalho foi desenvolvido utilizando o Método das Diferenças Finitas (MDF) para encontrar a solução discreta da equação (1) (Sperandio et al., 2003). O MDF foi um dos primeiros métodos numéricos desenvolvidos e é aplicado, até a atualidade, a uma extensa gama de problemas. Nesse método, utiliza-se uma malha, *grid*, sobre todo o domínio físico do problema, que contém determinados pontos nos quais são efetuadas as aproximações envolvidas. Neste trabalho foi utilizada uma malha contendo 96.000 pontos (60x40x40). Representações das derivadas em diferenças finitas são baseadas na expansão em série de Taylor. Discretizando a equação (1) por diferenças finitas chega-se à seguinte equação:

$$\frac{\phi_{i+1,j,k} - 2\phi_{i,j,k} + \phi_{i-1,j,k}}{(\Delta x)^2} + \frac{\phi_{i,j+1,k} - 2\phi_{i,j,k} + \phi_{i,j-1,k}}{(\Delta y)^2} + \frac{\phi_{i,j,k+1} - 2\phi_{i,j,k} + \phi_{i,j,k-1}}{(\Delta z)^2} = 0 \tag{5}$$

Para simplificar, consideram-se Δx, Δy *e* $\Delta z = 1.0$. A equação (5) torna-se:

$$\phi_{i+1,j,k} + \phi_{i-1,j,k} + \phi_{i,j+1,k} + \phi_{i,j-1,k} + \phi_{i,j,k+1} + \phi_{i,j,k-1} - 6\phi_{i,j,k} = 0 \tag{6}$$

Reorganizando a equação (6) resulta:

$$\phi_{i,j,k} = \frac{\phi_{i+1,j,k} + \phi_{i-1,j,k} + \phi_{i,j+1,k} + \phi_{i,j-1,k} + \phi_{i,j,k+1} + \phi_{i,j,k-1}}{6} \tag{7}$$

Utiliza-se a equação (7) discretizada para determinar o potencial em todas as células internas da malha. As velocidades são determinadas a partir do potencial através das equações (2), (3) e (4). Para obter a solução do problema é necessário especificar as condições de contorno.

Neste trabalho foi utilizado um potencial fixo no grid, de coordenadas x=0, com um valor pré-determindado, proporcionando uma simulação de velocidade inicial constante *Vx* igual a 20 m/s. Desta maneira o comportamento do escoamento e suas velocidades locais dependerão das condições de contorno que são modificadas em tempo quase-real sempre que um objeto é inserido ao grid e possui suas dimensões alteradas. Estas condições de contornos serão determinadas a seguir.

3.2 Condições de Contorno

As condições de contorno ou condições de fronteira são um conjunto de restrições que complementam a Equação de Laplace. A condição de contorno de Dirichlet indica um valor fixo, neste caso o potencial nulo na fronteira dos objetos geométricos selecionados e a condição de Neumann a velocidade nula na fronteira (Fortuna, 2000). Desta forma, neste trabalho, as formas geométricas, representadas por paredes sólidas, impõem condições de contorno de Dirichlet e Neumann e as fronteiras de simetria são representadas pelo limite do grid computacional.

Paredes sólidas. Para os pontos adjacentes a fronteiras sólidas, no cálculo das diferenças finitas, é usada a técnica da reflexão, ou a do ponto fantasma, por ser de implementação simples. A figura 3.2 representa 26 possibilidades de fronteira para um ponto no grid espacial 3D, levando-se em conta seus pontos adjacentes. Desta maneira, o algoritmo utilizado neste trabalho determina, para cada ponto no grid, uma condição de contorno individual, o que representa uma solução particular da equação de Laplace. Esta condição de contorno possibilita uma aplicação direta e simples no cálculo computacional, representada pela

implementação demonstrada no capítulo 3.4. A cada iteração feita, todos os pontos da malha são analisados e uma solução particular é utilizada.

Na figura 3.2 representa-se o potencial do ponto a ser calculado, $\phi_{i,j,k}$, pela cor amarela e a fronteira (barreira – ponto sólido) é representada pela cor preta.

Figura 3.2 - Representação dos 26 casos de fronteiras de um ponto dentro de uma malha computacional, usados para o cálculo do potencial.

Para o caso 1 da figura 3-2, o valor de $\phi_{i+1,j,k}$ passa a ser igual a $\phi_{i-1,j,k}$ e a solução da equação (1) neste ponto vem a ser:

$$\phi_{i,j,k} = \frac{2\phi_{i-1,j,k} + \phi_{i,j+1,k} + \phi_{i,j-1,k} + \phi_{i,j,k+1} + \phi_{i,j,k-1}}{6} \tag{8}$$

Nas condições de fronteiras sólidas duplas, como o caso 7, a solução da equação (1) é:

$$\phi_{i,j,k} = \frac{2\phi_{i-1,j,k} + 2\phi_{i,j-1,k} + \phi_{i,j,k+1} + \phi_{i,j,k-1}}{6} \tag{9}$$

E, para condições de fronteiras triplas, como no caso 19, a solução da equação (1) será:

$$\phi_{i,j,k} = \frac{2\phi_{i-1,j,k} + 2\phi_{i,j-1,k} + 2\phi_{i,j,k-1}}{6} \tag{10}$$

Fronteira de simetria. Considera-se como fronteira de simetria os limites do grid computacional, paralelos ao eixo X. Nesse tipo de fronteira a velocidade tangencial não está definida, então se pode escrever:

$$\frac{\partial \phi}{\partial x} \Big|_{sim} \cong \frac{\phi_{i,j,k} - \phi_{i-1,j,k}}{\Delta x} = 0 \tag{11}$$

O que implica $\phi_{i-1,j,k} = \phi_{i,j,k}$.

3.3 Velocidades

A solução das componentes da velocidade, equações (2), (3) e (4), pelo Método das Diferenças Finitas Centrais apresenta-se da seguinte forma:

$$v_x = \frac{\partial \phi}{\partial x} \cong \frac{\phi_{i+1,j,k} - \phi_{i-1,j,k}}{2\Delta x} \tag{12}$$

$$v_y = \frac{\partial \phi}{\partial y} \cong \frac{\phi_{i,j+1,k} - \phi_{i,j-1,k}}{2\Delta y} \tag{13}$$

$$v_z = \frac{\partial \phi}{\partial z} \cong \frac{\phi_{i,j,k+1} - \phi_{i,j,k-1}}{2\Delta z} \tag{14}$$

Para o caso 1, da figura 3.2 a componente da velocidade no eixo x, v_x é igual a zero. No caso 7 $v_x = 0$ e $v_y = 0$. Do caso 19 até o caso 26, $v_x = 0$, $v_y = 0$ e $v_z = 0$.

3.4 Implementação Computacional

O cálculo do potencial dos pontos da equação (7) se traduz computacionalmente da seguinte forma:

```
// ******** Início do cálculo dos pontos  *****
for (i=1; i<a; i++)
for (j=0; j<=b; j++)
for (k=0; k<=c; k++)
{
      vx[i][j][k] = -(u[i+1][j][k]-u[i-1][j][k])/2;
      vy[i][j][k] = -(u[i][j+1][k]-u[i][j-1][k])/2;
      vz[i][j][k] = -(u[i][j][k+1]-u[i][j][k-1])/2;
      ua = 1;
      ub = 1;
      uc = 1;
      ud = 1;
      ue = 1;
      uf = 1;

 // *** Condições de contorno ***

      if ( u[i+1][j][k]==0 ) {
            ub = 2;
            vx[i][j][k] = 0;
      }
      if ( u[i-1][j][k]==0 ) {
            ua = 2;
            vx[i][j][k] = 0;
      }
      if ( u[i][j+1][k]==0 ) {
            ud = 2;
            vy[i][j][k] = 0;
      }
      if ( u[i][j-1][k]==0) {
            uc = 2;
            vy[i][j][k] = 0;
      }
      if ( u[i][j][k+1]==0 ) {
            uf = 2;
            vz[i][j][k] = 0;
      }
      if ( u[i][j][k-1]==0 ) {
            ue = 2;
            vz[i][j][k] = 0;
      }

//*** Condição geral ***

u[i][j][k] = (ua*u[i+1][j][k]+ub*u[i-1][j][k]+uc*u[i][j+1][k]+ud*u[i][j-1][k]+ue*u[i][j][k+1]+uf*u[i][j][k-1])/6.;
…
```

Onde a, b e c são as fronteiras do domínio computacional.

O algoritmo analisa cada ponto e sua vizinhança espacial (figura 3.2), utiliza os índices ua, ub, uc etc para transformar a condição geral (equação 7) em uma das condições

particulares. Ao mesmo tempo, sempre que encontra alguma condição de barreira faz com que a velocidade relacionada se anule.

O método computacional utilizado realiza o cálculo constante do potencial, armazenando os resultados de forma vetorial, atualizados a cada iteração. Estes valores são utilizados no algoritmo de visualização, onde os dados numéricos são transformados em representações visuais.

4 MODELO DE VISUALIZAÇÃO DA EQUAÇÃO DE LAPLACE 3D

A solução numérica traz uma grande quantidade de dados que podem ser analisados individualmente ou de forma coletiva. O modelo visual permite uma visão global ou seletiva dos resultados, proporcionando a interpretação do fenômeno de forma mais abrangente.

A visualização dos valores das velocidades, traduzidos pela representação em cores, induz a uma interpretação rápida e clara de suas variações. As linhas de fluxo se tornam visíveis através da sua representação geométrica e não apenas pelos seus dados numéricos.

Existem várias linguagens de programação que podem ser utilizadas para a visualização de dados, como o C, Basic, Pascal e Fortran. Algumas delas são especiais, a exemplo da Cg (*C for Graphics)*,desenvolvidas para o hardware gráfico. Esta, apesar de baseada na linguagem C, é diferente de C, C++ ou Java, porque é bem específica. Não é conveniente escrever uma planilha ou um processador de texto em Cg, para tal existem formas mais apropriadas. Em vez disso, Cg tem como meta a capacidade de controlar programaticamente a forma, aparência e movimento de objetos processados usando hardware gráfico. Em termos gerais, este tipo de linguagem é chamada linguagem de sombreamento (Fernando e Kilgard, 2003).

Um computador, pessoal ou não, que tenha uma unidade de processamento gráfico (*GPU - Graphics Processor Unit*), deve ter uma unidade central de processamento (*CPU*) que executa o sistema operacional e programas aplicativos. CPUs são, pelo projeto, de objetivo geral. As CPUs executam aplicativos (por exemplo, processadores de texto e pacotes de contabilidade) escritos em linguagens de propósito geral, como C++ ou Java. Devido ao design das GPUs ser especializado, são muito mais rápidas em tarefas de gráficos, tais como cenas de renderização em 3D, do que um processador de propósito geral seria. As novas GPUs processam dezenas de milhões de vértices por segundo e rasterizam centenas de milhões ou até bilhões de fragmentos por segundo. No entanto, as GPUs não podem executar os mesmos programas arbitrários, de propósito geral, que pode uma CPU. Por causa da natureza especializada de alto desempenho da GPU é que existem linguagens como a Cg. As linguagens de programação de propósito geral são muito abertas para a tarefa especializada de processamento de vértices e fragmentos. Em contraste, a linguagem Cg é inteiramente dedicada a esta tarefa. Cg é uma linguagem auxiliar projetada especificamente para GPUs.

Como o objetivo deste trabalho não é de desenvolver pacotes de visualização em linguagens específicas, mas a união da visualização com o processamento numérico em tempo quase-real, foram adotados pacotes já desenvolvidos e otimizados em linguagens de propósito geral, como o C++. Assim, para a visualização do modelo computacional descrito no capítulo anterior e a construção do simulador descrito neste trabalho, foi adotada a linguagem de programação C++ (Alves, 2002) e bibliotecas de visualização gratuitas disponíveis na net, como o VTK (VTK, 2010). A apresentação gráfica foi desenvolvida utilizando programas de desenvolvimento de software como o Borland C++ Builder (Alves, 2020), o Visual C++ (Visual C++, 2009) e o Visual C# (Visual C#, 2009) da Microsoft.

A programação específica para o uso de GPUs estará como trabalho futuro, com o objetivo de aumentar o grid computacional, sem prejudicar o desempenho geral do aplicativo.

Sendo totalmente orientado a objeto, o C++ Builder foi utilizado neste trabalho para facilitar a criação da interface do simulador. Com ele foi possível criar toda a programação visual, adicionando objetos gráficos aos formulários, como botões, caixas de edição, legendas e caixas de combinação (Alves, 2002).

A versão utilizada foi a Borland C++ Builder 6.

Dentro do ambiente de trabalho do C++ Builder é possível utilizar funções de repetição e decisão como *if, switch, for, while, do/while, break e continue*, além de matrizes e strings. O componente *Timer* foi utilizado como atualizador para a função de tempo quase-real dos cálculos matriciais.

Com o propósito de testar o desempenho do algoritmo desenvolvido com o pacote de visualização utilizado, em mais de uma ferramenta, algumas partes principais do trabalho, como o cálculo e a visualização das linhas de fluxo foram construídas usando o Visual C++.

Do mesmo modo, também, uma parte da programação, construída em C++, foi modificada e transformada na linguagem C#. Para a visualização, foi utilizada a versão *.NET Wrappers for VTK,* disponível na internet (VTK.NET, 2009).

Para testar uma possível diferença entre desempenhos devido às linguagens e compiladores foi simulado um escoamento em torno de um cubo. Iniciou-se com a visualização de 6 linhas de fluxo, incrementando-se 4 linhas a cada medição até chegar a 30 linhas de corrente visualizadas (figura 4.1). A medição do desempenho foi feita baseada na quantidade de “frames por segundo” (FPS) que o computador conseguia mostrar, a partir do número de linhas de fluxo apresentado. A construção de um quadro comparativo (fig. 4.2) foi executada de forma manual onde, a cada quantidade de linhas acrescidas, tomava-se o valor do desempenho. O processo foi executado de forma idêntica para cada linguagem, ou seja,

somente o aplicativo testado estava sendo rodado no ambiente Windows. Desta maneira, chegou-se a conclusão que a diferença entre as performances obtidas não teve significado quantitativo, apenas uma leve superioridade apresentada pelo compilador da Borland, representado pelo gráfico na figura 4.2. Portanto, optou-se pelo aplicativo Borland C++ Builder para o desenvolvimento do simulador, por ter sido inicialmente usado.

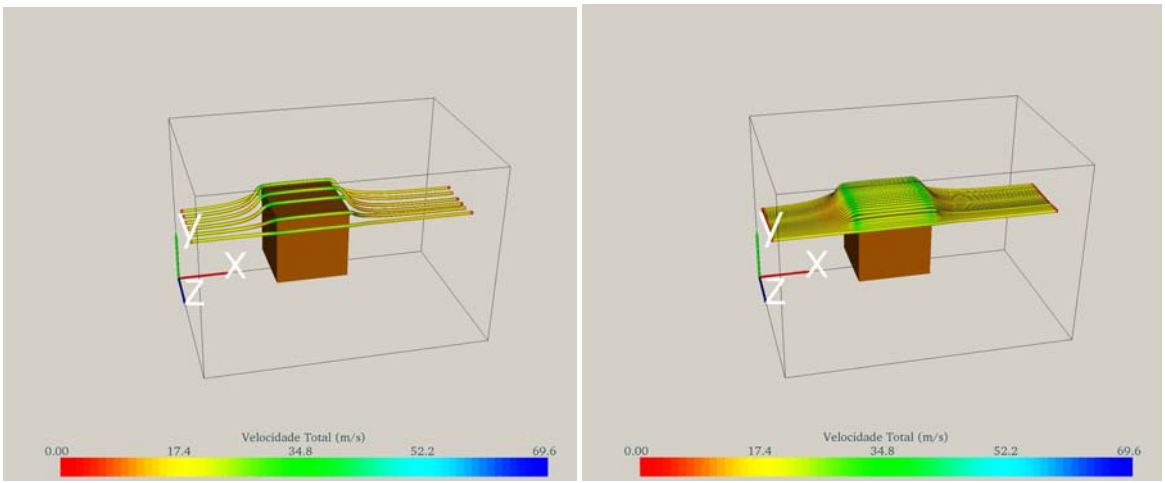


Figura 4.1 – Cubo com (a) 6 linhas de corrente e (b) 30 linhas de corrente usado como referência na comparação de velocidade de processamento entre a compilação Borland C++, Visual C++ e Visual C#.

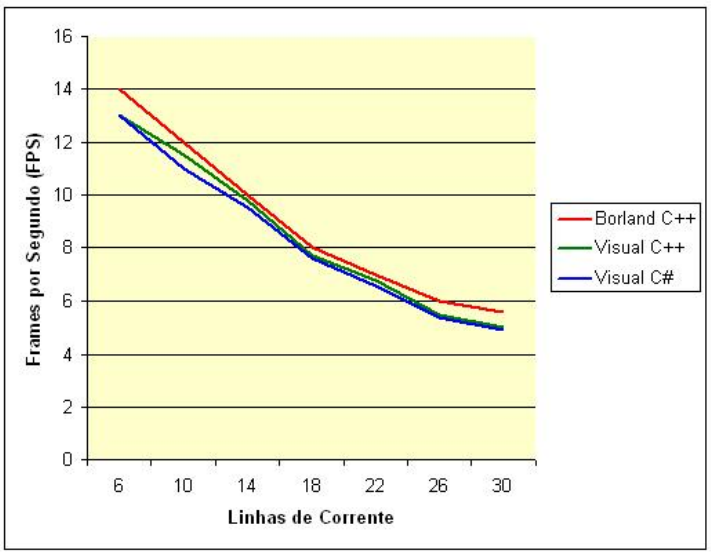


Figura 4.2 – Comparação entre Borland C++, Visual C++ e Visual C#. Nos três casos a performance, medida em FPS, diminui com o aumento das linhas de corrente visualizadas.

O pacote de bibliotecas de visualização VTK (*Visualization Tool* Kit) foi escolhido para apresentar as formas geométricas por possuir códigos já testados e otimizados e por

oferecer opções de visualização adequadas à proposta do trabalho. Além disso, de uma maneira inédita, foi utilizado o VTK para a visualização de CFD interativa em tempo quase-real.

No simulador desenvolvido foram utilizadas diversas classes para o processamento e a visualização dos gradientes de velocidades e linhas de corrente.

4.1 VTK e Borland C++ Builder 6

O conjunto de classes para a visualização, fornecidos pelo VTK, precisou ser integrado ao software Borland C++ Builder 6 para ser usado no desenvolvimento do simulador. O processo de integração entre o VTK e o Borland é descrito abaixo.

Com a instalação do Borland C++ Builder 6 é necessária a obtenção dos arquivos VTK: o download do código fonte (*VTK-X.X.X-LatestRelease.zip),* os dados necessários para os exemplos (*VTKDATA-X.X.X.zip)* e o Tcl/Tk 8.3 é feito a partir do site http://www.vtk.org. Também é necessário obter o *Cmake* para auxiliar na preparação para a compilação (http://www.cmake.org).

O primeiro passo é a instalação do Cmake, assim como a instalação do Tcl/TK 8.3.

O próximo passo é a descompactação dos arquivos com os códigos-fonte do VTK para o diretório C:\VTK (ou outro que escolher). Descompacta-se também o arquivo VTKData em um diretório, por exemplo C:\VTKData. É necessário converter a biblioteca TCL83.LIB de COFF para OMF, utilizando o utilitário *coff2omf* num "prompt de comando" como mostrado a seguir:

coff2omf ...\tcl*xx*.lib tcl*xx*.lib

Este mesmo processo é necessário para todos os arquivos de bibliotecas existentes no diretório pai (tcl*xx*, tclstub*xx*, tk*xx*, tkstub*xx,* onde *xx* indica a versão) A versão utilizada neste trabalho foi a 8.3.2.

Em seguida executa-se o *Cmake* de acordo com os seguintes parâmetros:

- Código de fonte: C:\VTK;
- Código Binário: C:\VTKBin;
- Compilador (*Build For*): Borland Makefiles;
- Mostre Opções Avançadas: Selecionado.

Clica-se em *Configure* para atualizar as opções. À medida que a configuração for estabilizando, as opções, em fundo avermelhado, são apresentadas em fundo cinza e o botão *OK* é habilitado (figura 4.3).

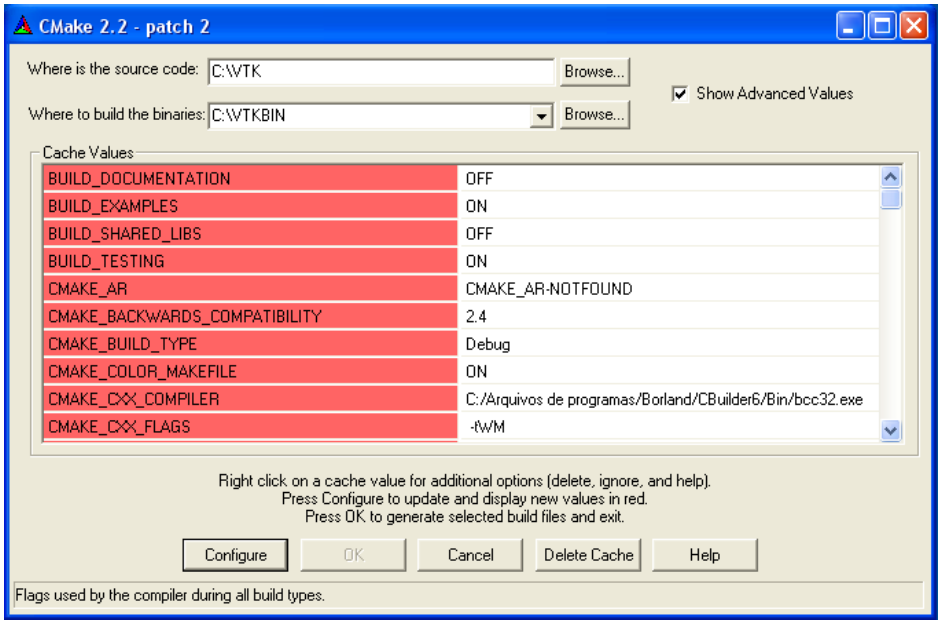


Figura 4.3 – Interface do Cmake.

Algumas opções devem ser configuradas de acordo com as necessidades específicas. Por exemplo, a opção USE_SHARED_LIBS quando *desligada* (OFF) serão geradas bibliotecas estáticas (lib), caso contrário (ON) serão geradas bibliotecas dinâmicas (dll). É necessário indicar a localização das bibliotecas requeridas:

- TK_LIBRARY = C:\Arquivos de Programas\Tcl\lib\bcb\tk83.lib;
- TCL_LIBRARY = C:\Arquivos de Programas\Tcl\lib\bcb\tcl83.lib;
- VTK_USE_RENDERING: ON;
- VTK_USE_HYBRID: ON;
- VTK_DATA_ROOT = C:\VTKData.

As bibliotecas aqui utilizadas são aquelas convertidas de COFF para OMF. Após clicar em *OK* serão criados os arquivos necessários para a compilação.

Finalmente, utilizando um prompt de comando executa-se o comando *make* dentro do diretório C:\VTKBin. Este processo pode durar mais de duas horas em um computador Pentium 4,2GHz, com 512Mb de RAM.

Após a compilação é possível utilizar o VTK em programas desenvolvidos no C++ Builder. Para facilitar esta utilização há um pacote do VTK chamado *VTKRenderWindow* que

pode ser instalado. O diretório C:\VTK\Examples\GUI\Win32\vtkBorland traz os arquivos e as instruções necessárias para a instalação deste pacote.

Um componente na palheta *Samples* do C++ Builder indica a instalação correta do VTK (figura 4.4).

Figura 4.4 - Componente *VTKRenderWindow* instalado no Borland C++ Builder 6.

4.2 Desenvolvimento e Implementação

A utilização da ferramenta C++ Builder juntamente com o VTK permitiu a combinação de dois facilitadores, a apresentação, ou lay out, do programa desenvolvido e a utilização de rotinas de visualização já otimizadas (classes).

O acesso às classes do VTK é feito de forma simples, exemplificado pela figura 4.5. A construção dos menus de acesso e janelas interativas, botões, assim como o design do programa foi feito utilizando objetos e componentes disponíveis no C++ Builder. Desta maneira foram criados acessos rápidos às formas de visualização das figuras geométricas e linhas de fluxo. Alguns menus na forma de "cortinas" são semelhantes aos que temos em programas tradicionais como o Word da Microsoft. Isto permite a colocação de várias funções de forma que o menu principal não se apresente com muitas informações.

Todas as janelas apresentadas podem ser redimensionadas a critério do usuário. Desta forma é possível adaptar o conteúdo visualizado ao tamanho do monitor ou projetor usado.

Criou-se uma conexão entre todos os *forms* (janelas ou moldes) onde, por exemplo, um valor dimensional de uma forma geométrica, mudado na janela de ferramentas, é visualizado instantaneamente na(s) janela(s) de visualização.

Como o propósito do simulador apresentado não é a de um produto comercial final, apenas uma forma simplificada de uma nova proposta, não foram estudados a ergonomia do software e a cognitividade da interface.

A figura 4.7 traz um pequeno fluxograma representativo do simulador desenvolvido.

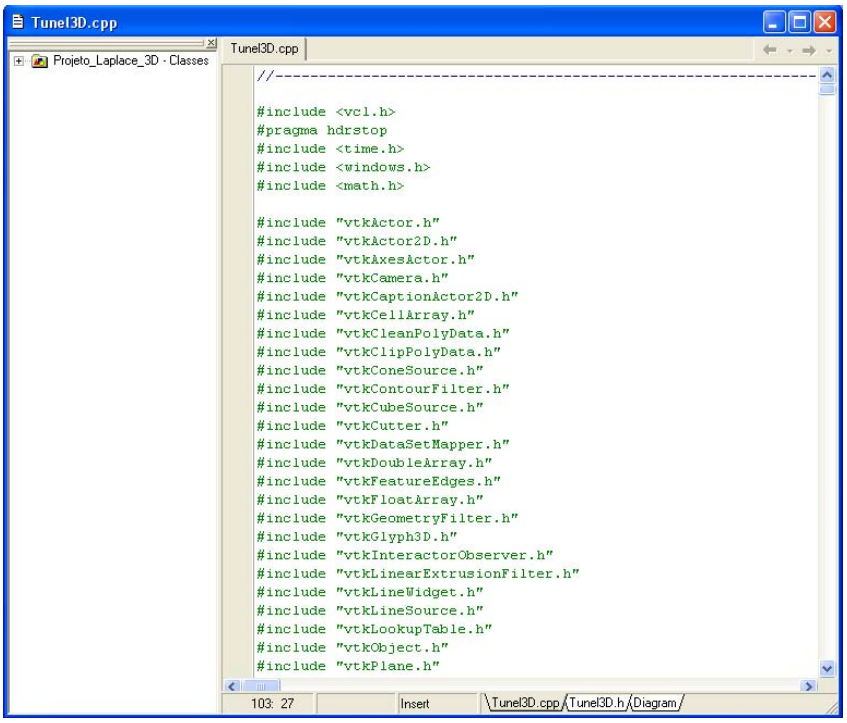


Figura 4.5 - Acesso às classes do VTK pelo Borland Builder C++.

O *menu* principal permite uma interatividade simples com botões de acesso rápido e cortinas em cascata com diversas opções configuradas (figura 4.6). Optou-se por apresentar uma *taxa de atualização* medida em *frames por segundo* (FPS) na primeira tela para que se possa avaliar a capacidade computacional e perceber o desempenho do software em função dos parâmetros utilizados, como a quantidade de linhas de fluxo ou gradientes de velocidades.

A partir da interface inicial é possível selecionar: duas janelas de visualização, uma onde podem ser observadas as linhas de fluxo e outra onde é possível visualizar gradientes e isolinhas em corte; uma janela onde se tem dados numéricos; uma janela de ferramentas onde é possível alterar dados de entrada; figuras geométricas pré-determinadas, que podem ser combinadas; a parada e recomeço dos cálculos para que se possa interromper o processamento numérico; outras opções em cortinas, como a seleção da escala de cores.

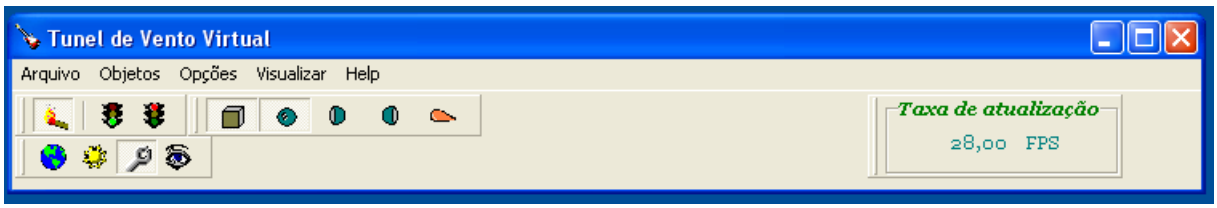


Figura 4.6 – Menu Principal

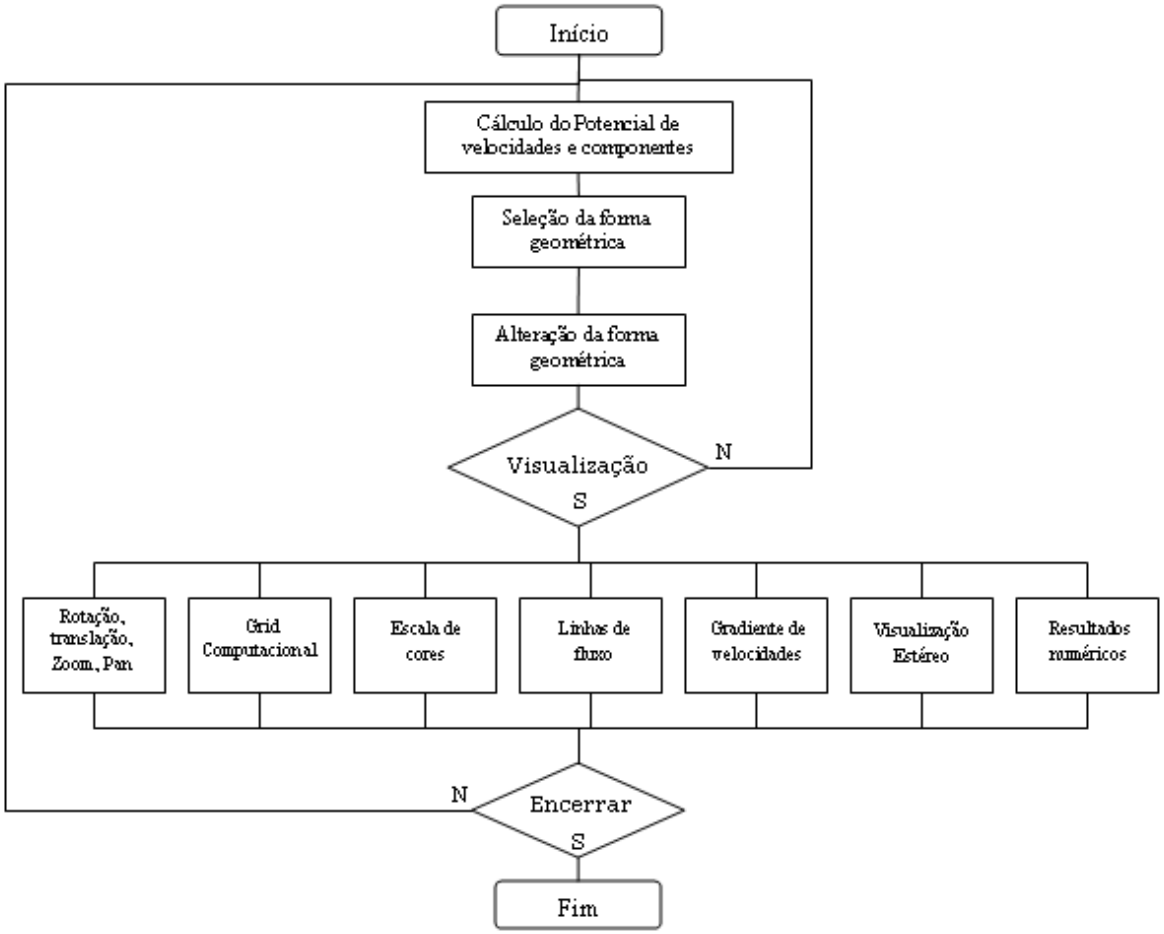


Figura 4.7 – Fluxograma do simulador desenvolvido

4.2.1 Figuras geométricas

As figuras geométricas, utilizadas inicialmente no simulador, são visualizadas utilizando-se classes pré-estabelecidas. Estes objetos podem ser manipulados de forma interativa, onde o usuário poderá interagir, alterando suas dimensões e posições. O VTK também possibilita o desenvolvimento e visualização de formas alternativas, permitindo a criação de infinitas geometrias. Neste trabalho, aproveita-se algumas formas geométricas padrão, como a esfera *vtkSphereSource* e o cubo *vtkCubeSource*. Além disso, foi criada uma forma geométrica semelhante ao perfil de uma asa. Um exemplo destas visualizações podem ser vistas na figura 4.8. O simulador desenvolvido permite a seleção de várias formas geométricas, individuais ou agrupadas, que podem ser modificadas e posicionadas individualmente. O cálculo e a visualização do gradiente de velocidades e/ou linhas de corrente é feito de maneira global, considerando todas as formas geométricas inseridas no

grid. Além das dimensões iniciais, pode-se modificar o paralelepípedo (cubo), a esfera, duas hemi-esferas e a forma semelhante a um perfil de asa, que pode ter o perfil alterado. A cada alteração pode-se verificar todas as velocidades e analisar a trajetória das linhas de corrente em tempo quase-real. Também é possível visualizar os campos de velocidades na forma de gradientes ou em isolinhas, simultaneamente.

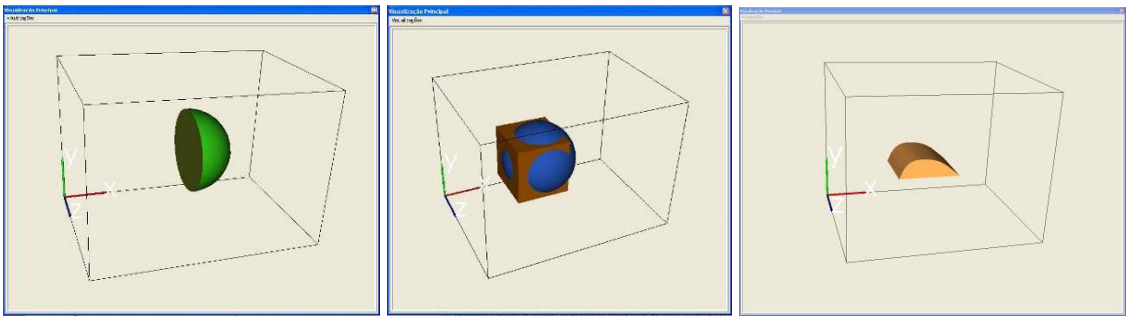


Figura 4.8 - Exemplos de formas geométricas: semi-esfera, esfera+cubo e perfil de asa, visualizados utilizando o programa desenvolvido neste trabalho.

Uma parte do código deste exemplo pode ser visto a seguir:

```
vtkSphereSource *sphere1 = vtkSphereSource::New();
   sphere1->SetRadius(raio-0.5);
vtkPlane *plane1 = vtkPlane::New();
   plane1->SetNormal( 1, 0, 0);
vtkClipPolyData  *clipper1 = vtkClipPolyData::New();
   clipper1->SetInput(sphere1->GetOutput());
…
vtkFeatureEdges *boundaryEdges1 =  vtkFeatureEdges::New();
 boundaryEdges1->SetInput(clipper1->GetOutput());
...
vtkPolyDataMapper *boundaryMapper1 = vtkPolyDataMapper::New();
 boundaryMapper1->SetInput(boundaryPoly1);
vtkActor *boundaryActor1 = vtkActor::New();
 boundaryActor1->SetMapper(boundaryMapper1);
 boundaryActor1->GetProperty()->SetColor(0.7, 0.8, 0.2);…
```

4.2.2 Linhas de Corrente

Na análise de problemas da Mecânica dos fluidos, frequentemente é vantajosa a representação visual do campo do escoamento. Tal representação é proporcionada pelas linhas

de emissão, pelas trajetórias, pelos filetes e pelas linhas de corrente (Fox e McDonald, 2001). O capítulo 2 deste trabalho demonstra as mais variadas formas dessas representações.

Linhas de corrente, ou streamlines em inglês, são linhas traçadas no campo de escoamento de modo que, em um dado instante, são tangentes à direção do fluxo em todos os pontos do campo. Como são tangentes aos vetores velocidades em cada ponto do campo do escoamento, não pode haver escoamento através das linhas de corrente.

De acordo com Martinez (1995), linha de corrente é a curva integral do campo vetorial de velocidade instantânea, que passa por um dado ponto do espaço num dado instante de tempo. Em outras palavras, são linhas tangentes em todos seus pontos ao campo de velocidade, num dado instante de tempo. Um exemplo de visualização das linhas de corrente aparece na figura 4.9.

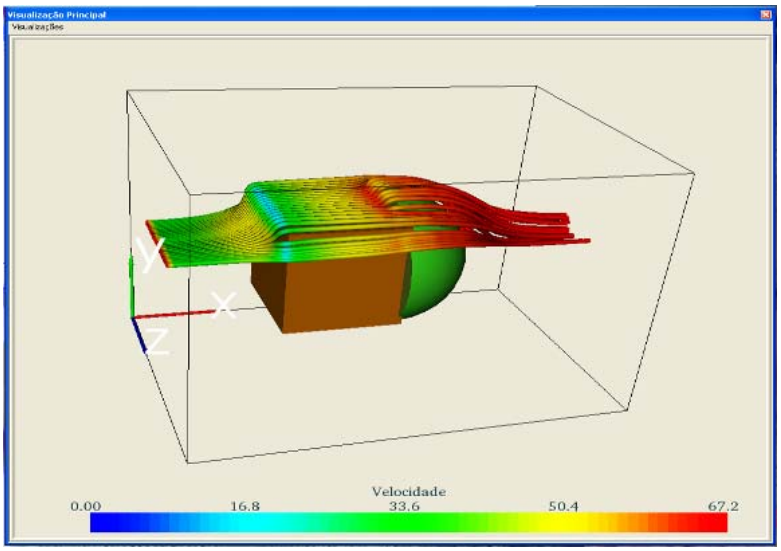


Figura 4.9 - Visualização das linhas de corrente em tempo quase-real, usando o simulador desenvolvido neste trabalho.

Determinam-se as linhas de corrente resolvendo-se um Problema de Valor Inicial (PVI), no qual o valor inicial de cada linha de corrente corresponde às coordenadas *(x, y e z)* da região *R,* onde se deseja iniciar o traçado.

Dentre os diversos métodos numéricos utilizados para determinar as trajetórias das partículas neste trabalho foi utilizado o método de Runge-Kutta (Albrecht, 1973). No caso de uma trajetória no espaço (3D), é necessário considerar as três componentes das velocidades (v_x, v_y e v_z) em cada ponto do grid (x_k, y_k e z_k). Para isso as equações são:

$$X_{k+1} = X_k + \frac{h}{6}\left(K_0 + 2K_1 + 2K_2 + K_3\right) \tag{15}$$

$$Y_{k+1} = Y_k + \frac{h}{6}\left(M_0 + 2M_1 + 2M_2 + M_3\right) \tag{16}$$

$$Z_{k+1} = Z_k + \frac{h}{6}\left(N_0 + 2N_1 + 2N_2 + N_3\right) \tag{17}$$

$$K_0 = V_x \tag{18}$$

$$M_0 = V_y \tag{19}$$

$$N_0 = V_z \tag{20}$$

$$K_1 = V_x\left(X_k + \frac{h}{2}K_0, Y_k + \frac{h}{2}M_0, Z_k + \frac{h}{2}N_0\right) \tag{21}$$

$$K_2 = V_x\left(X_k + \frac{h}{2}K_1, Y_k + \frac{h}{2}M_1, Z_k + \frac{h}{2}N_1\right) \tag{22}$$

$$K_3 = V_x\left(X_k + hK_2, Y_k + hM_2, Z_k + hN_2\right) \tag{23}$$

$$M_1 = V_y\left(X_k + \frac{h}{2}K_0, Y_k + \frac{h}{2}M_0, Z_k + \frac{h}{2}N_0\right) \tag{24}$$

$$M_2 = V_y\left(X_k + \frac{h}{2}K_1, Y_k + \frac{h}{2}M_1, Z_k + \frac{h}{2}N_1\right) \tag{25}$$

$$M_3 = V_y\left(X_k + hK_2, Y_k + hM_2, Z_k + hN_2\right) \tag{26}$$

$$N_1 = V_z\left(X_k + \frac{h}{2}K_0, Y_k + \frac{h}{2}M_0, Z_k + \frac{h}{2}N_0\right) \tag{27}$$

$$N_2 = V_z\left(X_k + \frac{h}{2}K_1, Y_k + \frac{h}{2}M_1, Z_k + \frac{h}{2}N_1\right) \tag{28}$$

$$N_3 = V_z\left(X_k + hK_2, Y_k + hM_2, Z_k + hN_2\right) \tag{29}$$

No aplicativo apresentado neste trabalho a posição inicial e a quantidade de linhas de corrente podem ser escolhidas pelo usuário, assim como sua forma de apresentação.

Com os dados gerados no algoritmo de processamento utilizam-se algumas classes do VTK para a visualização das linhas de corrente. Uma destas foi a *vtkStreamTracer*. Acopladas a esta classe, foram utilizadas classes que possuem a função de filtros, como *vtkRuledSurfaceFilter* e *vtkTubeFilter* que permitem a alteração no diâmetro da linha de corrente, a critério do usuário ou uma variação automática, em função do valor local da

velocidade. A diminuição do diâmetro das linhas de corrente poderá ser necessária quando se utiliza uma quantidade grande de linhas, evitando a sobreposição do desenho.

Para o início das linhas de corrente foram utilizadas as classes *vtkLineSource* e *vtkPlaneSource*, permitindo-se visualizá-las a partir de um plano ou de uma linha, conforme mostra a figura 4.10. Outra opção é a visualização das linhas de corrente na forma de tubos (*streamtubes*) ou de fitas (*streamribbons*), que são extensões das streamlines. A visualização das linhas de corrente na forma de fitas (streamribbons) pode ser interessante para se analisar o comportamento rotacional de um fluxo 3D, o que não é possível com streamlines sozinhas (Ueng et al., 1996) Neste trabalho, por se tratar de um fluxo irrotacional, a representação das streamribbons é meramente ilustrativa, não representando o comportamento do fluido. Um streamtube é uma streamline que pode ser dilatada ou comprimida para mostrar a expansão do fluxo. Os streamribbons e streamtubes oferecem vantagens sobre streamlines na forma de que eles podem codificar mais propriedades, como a divergência e convergência do campo vetorial, em propriedades geométricas do respectivo objeto integral.

Neste trabalho é possível escolher a origem, a quantidade e a forma das linhas de corrente.

A classe *vtkRungeKutta4* foi usada para fazer a integração numérica necessária à visualização das linhas de corrente.

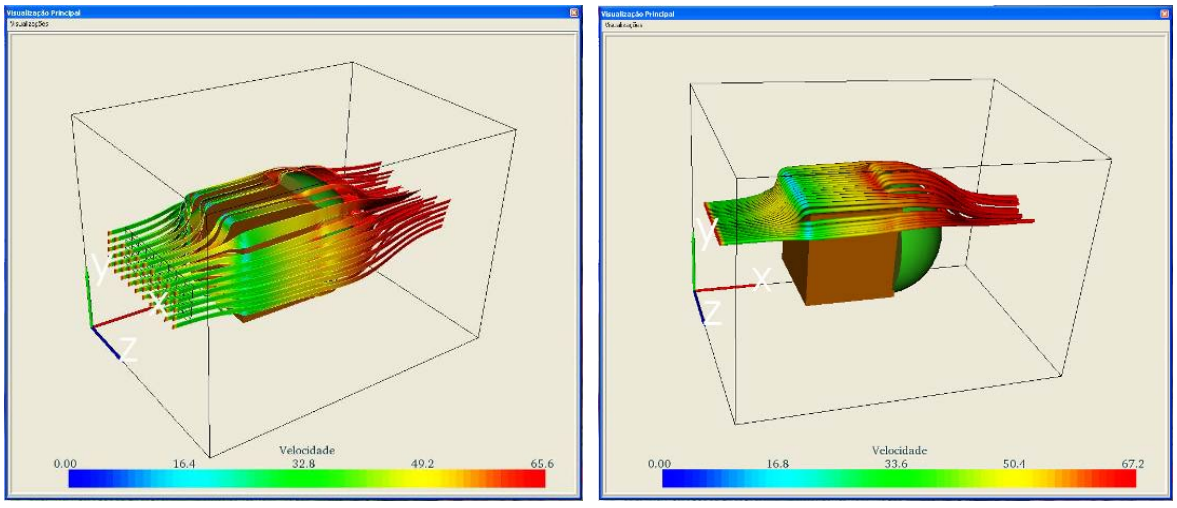
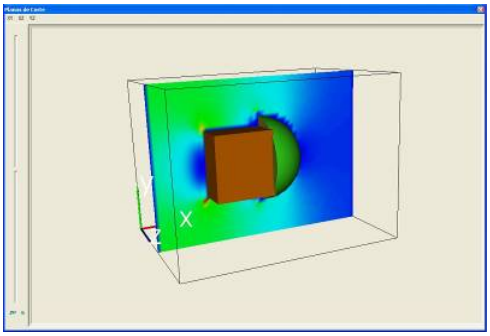


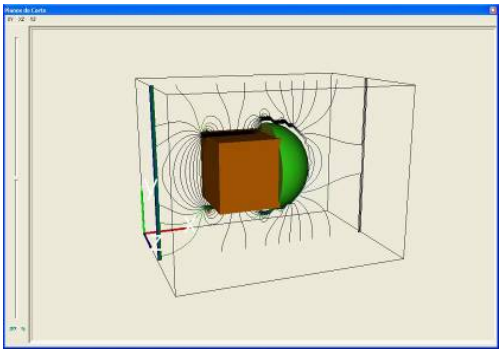
Figura 4.10 - Exemplos de visualização de linhas de corrente iniciadas a partir de um plano de origem, com representação por streamribbons, e a partir de uma reta de origem, com representação por streamlines.

4.2.3 Planos de Corte

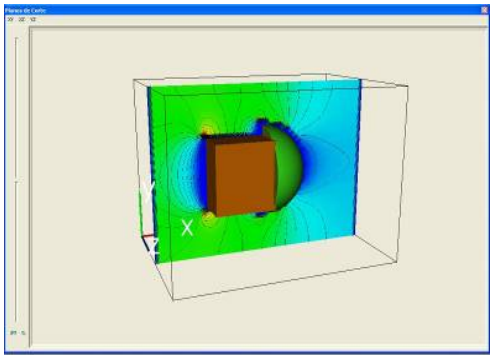
Para a visualização de planos de corte, o VTK possui algumas classes que podem ser aplicadas. Uma destas, utilizadas neste trabalho, foi a *vtkRectilinearGridGeometryFilter*. Ela permite fazer cortes no grid para se visualizar "fatias" selecionadas. Exemplos destes cortes podem ser vistos na figura 4.11. Este trabalho permite ao usuário escolher a visualização de gradientes, isolinhas ou ambos simultaneamente e selecionar a posição do plano a ser visualizado nos eixos x, y ou z.



(a)



(b)



(c)

Figura 4.11 - Visualização do gradiente de velocidades (a), de isolinhas (b) e ambos (c), feita a partir de planos de corte no sentido dos eixos xy.

4.2.4 Visualização Estereoscópica

O VTK permite programar um algoritmo de visualização com efeito estereoscópico (visualização estéreo). A figura 4.12 mostra uma das possibilidades deste tipo de visualização implantadas neste trabalho. Neste caso, para o usuário ter a sensação de profundidade é necessário o uso de óculos com filtros ciano e magenta. Outras opções disponíveis no trabalho são o estéreo *dresden* e o *entrelaçado.*

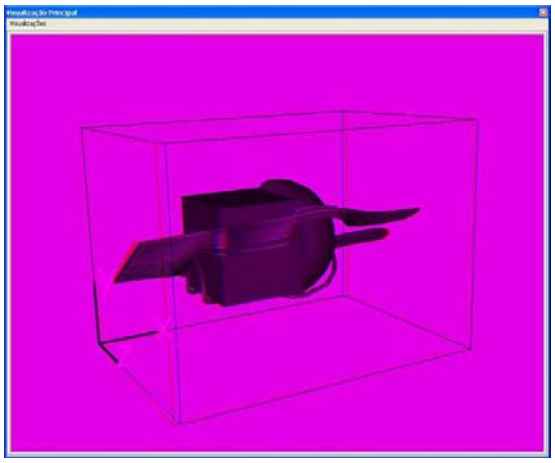


Figura 4.12 - Visualização estérea de linhas de corrente a partir do modo *cyan-magenta*.

4.2.5 Opções de Interatividade

Como as operações de cálculo estão sendo realizadas sem interrupção, geram-se dados que podem ser representados por imagens, ou mesmo valores numéricos, de diversas formas. Estas apresentações são exibidas a critério do usuário. Muitas formas de interação foram disponibilizadas no trabalho realizado e outras poderão ser implementadas de acordo com a necessidade e/ou evolução do programa.

O simulador apresentado permite a inserção de objetos em um domínio computacional, inicial, de 96000 pontos (60x40x40). Este valor se deu devido a um limite computacional apresentado pelo compilador utilizado. Um grid maior representava um erro de compilação. Não foram estudados a influência da alteração de parâmetros do compilador, da plataforma utilizada ou da quantidade de memória alocada.

Com um *menu de opções*, selecionado a partir do menu principal (figura 4.13), é possível selecionar ferramentas para a alteração das dimensões e da localização dos objetos geométricos. As operações de transformações geométricas como rotações, translações e escala

(*zoom)* podem ser realizadas de forma simples e rápida. Todos os objetos podem ser colocados ou subtraídos do grid permitindo a visualização da alteração dos gradientes de velocidades ou linhas de fluxo em tempo quase-real, referentes a estes movimentos.

Muitas opções foram colocadas a disposição do usuário para que este possa utilizar a ferramenta da maneira que achar mais conveniente. Ferramentas como a visualização da posição da maior ou menor velocidade do grid. A escolha de um ponto específico para se obter o resultado numérico e visual das velocidades também pode ser selecionado.

O processamento pode ser interrompido e recomeçado a qualquer momento, para uma melhor interpretação dos resultados numéricos ou visuais obtidos. Imagens instantâneas ou *screenshots* da visualização podem ser tirados e gravados a qualquer momento.

Uma janela adicional pode ser mostrada indicando os valores numéricos obtidos durante as transformações.

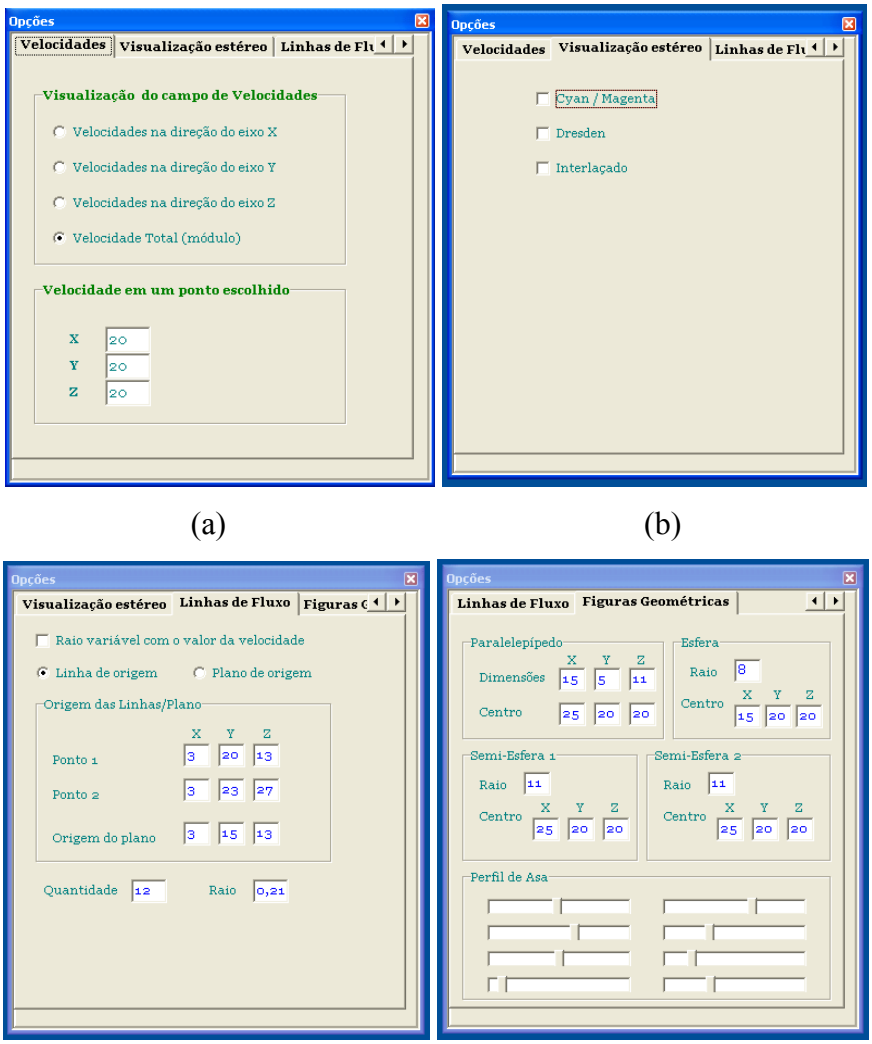


Figura 4.13 Menu *opções* para seleção interativa

Dentro da opção *Linhas de Fluxo* no menu *opções* existem campos que permitem escolher a quantidade de linhas a serem visualizadas, sua dimensão e uma seleção indicando a origem das linhas de fluxo como sendo uma linha ou um plano, assim como a posição de partida destas linhas. É possível, ainda, selecionar se deseja que o raio local da linha de fluxo varia com o valor da velocidade, transformando as streamlines em streamtubes (figura 4.13-c).

Outro tipo de interação permite efetuar a seleção do gradiente de velocidades (v_x, v_y ou v_z) referente ao eixo escolhido, assim como selecionar o local onde se deseja verificar valores das componentes das velocidades (figura 4.13 – a). Uma referência da posição deste ponto escolhido, na forma de uma pequena esfera, pode ser visualizada a partir da escolha no menu principal. Os valores numéricos são apresentados na opção *Dados de Saída.*

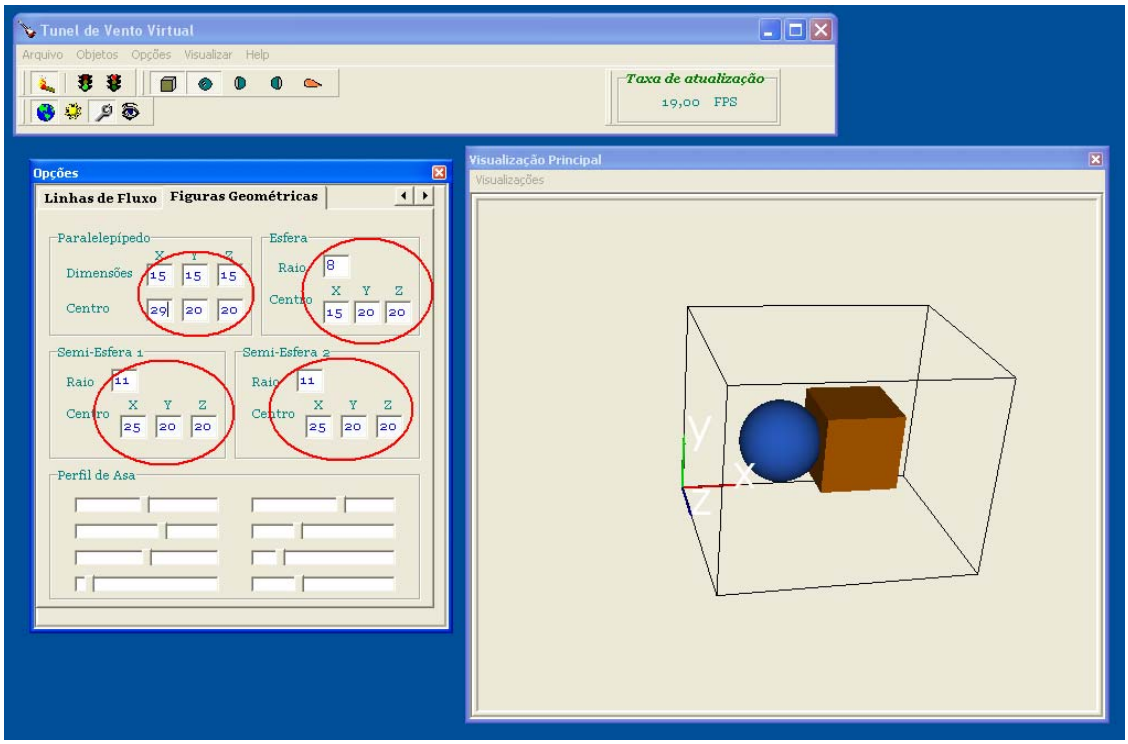


Figura 4.14 Ferramentas para alterar as dimensões, em tempo quase real, dos objetos.

A janela de *Saída* de dados traz valores numéricos como as velocidades em um ponto central, as velocidades máximas e mínimas com suas posições no grid e as velocidades num ponto escolhido qualquer (figura 4.15). Ao se alterar as formas geométricas, os novos valores de velocidades são gerados e visualizados instantaneamente.

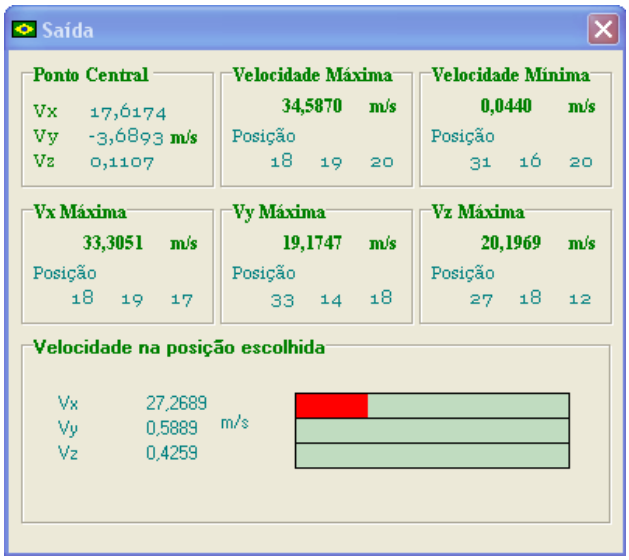
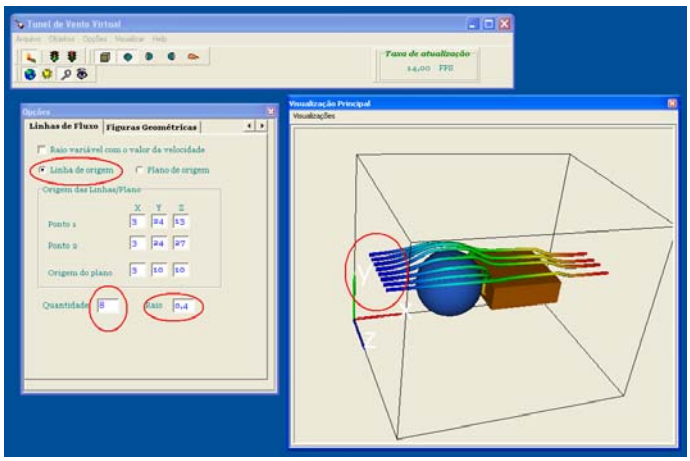
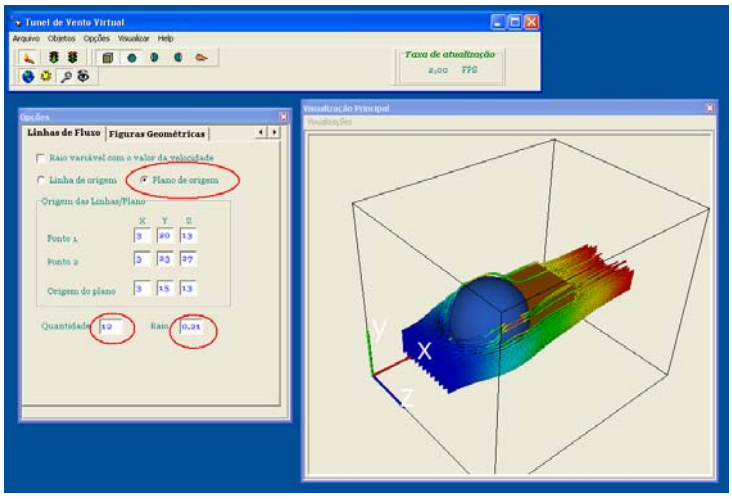


Figura 4.15 – Janela de saída de dados.



(a)



(b)

Figura 4.16 - Opções de visualização de linhas de fluxo (a) 8 streamlines iniciadas por uma linha de origem e (b) 144 streamlines iniciadas por um plano de origem.

A visualização estereoscópica permite uma melhor sensação de imersão, pois a percepção tridimensional se torna mais realista (figura 4.12).

Este trabalho contempla o uso de duas janelas de visualização, que podem ser dispostas simultaneamente permitindo, ao mesmo tempo, visualizar linhas de corrente e planos de corte, evitando sobrepor as imagens, o que torna a visualização muito poluída. As janelas podem ser manipuladas individualmente.

Quando selecionadas, as figuras geométricas podem ter suas dimensões e posição alteradas. O perfil de asa possui sua forma definida por uma interpolação polinomial na forma de diferenças divididas de Newton (Sperandio et al., 2003). Por intermédio de barras deslizantes é possível alterar os valores dos pontos e, com isso, mudar seu perfil em tempo quase-real (figura 4.14).

Uma barra com o valor das velocidades em escala de cores pode ser selecionada. A cor de fundo da janela de visualização também pode ser alterada para modificar o contraste dos objetos e linhas com o ambiente.

A figura 4.17 mostra de forma geral algumas das opções de visualização oferecidas pelo simulador desenvolvido.

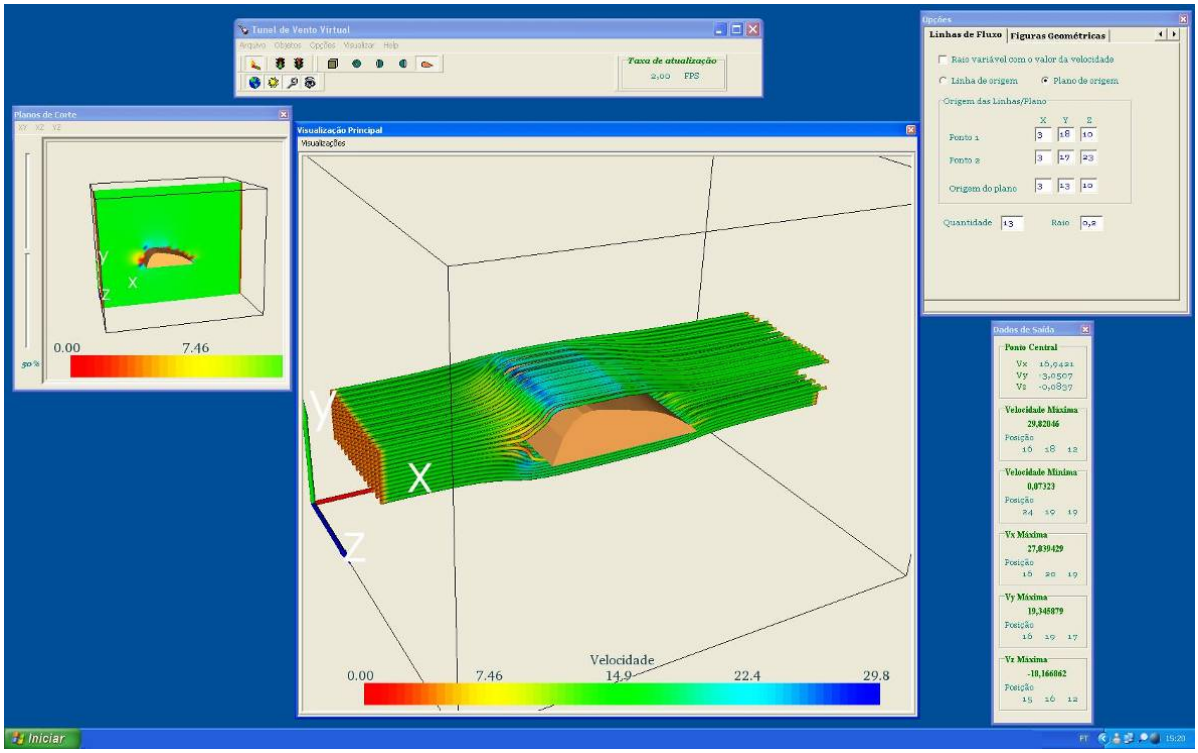


Figura 4.17 - Simulador desenvolvido neste trabalho para visualização 3D de escoamentos tridimensionais sobre objetos.

Até chegar ao simulador desenvolvido descrito neste capítulo passou-se por diversos testes e alterações. Mais de 150 versões foram feitas e testadas até se obter um programa que rodasse em qualquer computador com a plataforma Windows sem que se apresentasse erros de funcionamento. Desta maneira, mesmo não se tratando de um produto comercial final, procurou-se construir um aplicativo estável. A performance do simulador varia em função de cada máquina, devido às diferentes combinações possíveis de componentes (processadores, memórias etc).

5 VALIDAÇÃO E ESTUDOS DE CASO

Para fazer um estudo de um fenômeno físico qualquer se pode optar por métodos teóricos ou ensaios de laboratório mas, mesmo esses últimos, devem estar embasados em análises teóricas prévias. Portanto, ao modelar a física do problema, na qual se determina quais as grandezas físicas (como temperatura, pressão, velocidade) atuam sobre o sistema físico, é necessário expressar este modelo de forma matemática por intermédio de equações. Para tratar o modelo computacionalmente é preciso considerar de forma adequada as equações e a região (domínio) em que elas são válidas. Como não se podem obter soluções numéricas sobre uma região contínua, devido a seus infinitos pontos, discretiza-se o domínio em pontos onde as soluções serão obtidas. A distribuição adequada dos pontos no domínio é fundamental para se obter uma solução numérica representativa do escoamento. O resultado é um conjunto de equações algébricas, geralmente lineares, que podem ou não estar acopladas. Ao se resolver essas equações, a solução do problema está fornecida. Métodos experimentais e, mais recentemente, computacionais são utilizados para verificar a validade do modelo teórico. As técnicas de visualização científica tem papel fundamental nessa etapa. Comparando-se os resultados numéricos com dados experimentais, por exemplo, pode se ajustar o modelo matemático até que ele reflita a física do problema. Intuitivamente, percebe-se que, quanto maior for o número de pontos discretos, isto é, quanto mais fina for a malha, mais fiel ao modelo será o resultado numérico obtido. Obviamente, maior também será o custo computacional. No caso de problemas bidimensionais, ao dobrar o número de incógnitas em cada dimensão, o custo computacional se torna quatro vezes maior; em problemas tridimensionais, há um aumento de oito vezes (Fortuna, 2000).

Os resultados numéricos fornecidos pelo computador devem ser sempre analisados em relação à física do problema. Conforme a AIAA (American Institute of Aeronautics and Astronautics), a partir de um modelo para um dado fenômeno físico, existem duas etapas básicas de avaliação de um simulador:

- Etapa de verificação: Determina com que grau a implementação do modelo, representada por equações, parâmetros e métodos numéricos adotados, corresponde à sua descrição conceitual, isto é, se o modelo está corretamente implementado. Os resultados numéricos fornecidos pela implementação do modelo são comparados a

outras soluções, consideradas "de referência". Estas soluções podem ser analíticas, numéricas ou experimentais.

- Etapa de validação: Quantifica o grau de representatividade do modelo em relação ao fenômeno físico real. Esta análise é normalmente realizada por comparações sistemáticas com dados experimentais, representativos de fenômenos nos quais se espera utilizar o simulador.

Deve-se notar que, em geral, dados experimentais contém incertezas e erros de medidas aleatórios e/ou sistemáticos (Camões, 2001). A magnitude desses erros deve ser estimada para efeito de validação do simulador.

Os processos de verificação e validação não garantem que a previsão fornecida pelo simulador esteja correta. Eles apenas garantem que, a partir dos testes realizados com diversas combinações de fenômenos físicos representativos dos fenômenos que serão previstos pelo simulador, o simulador forneceu resultados compatíveis com aqueles encontrados no mundo real. O simulador deve ser considerado como uma ferramenta, cuja qualidade depende do uso que é feito dela (Fortuna, 2000).

Quando se resolve uma equação numericamente deve-se verificar se a solução se aproxima, de alguma forma, da solução real. Para que isso aconteça, deve-se determinar a sua consistência, estabilidade e convergência.

Para que uma discretização seja consistente com a equação diferencial, seu erro local de truncamento (ELT) deve tender a zero quando Δx, $\Delta t \rightarrow 0$. Quando a solução numérica no domínio de interesse se aproxima da solução exata, diz-se que o método numérico é convergente. Por fim, um método numérico estável é aquele no qual quaisquer erros ou perturbações não são amplificados sem limite. Os exemplos mais diretos são condições de fronteira ou iniciais aproximadas de forma incorreta, e acúmulo de erros de arredondamento cometidos pelo computador durante os cálculos.

A equação de Laplace utilizada neste trabalho (eq. 01) foi discretizada por meio de diferenças centrais de segunda ordem (eq. 06), o que nos coloca um ELT de $O(\Delta x)^2$.

Para uma primeira validação, onde não há objetos no interior do grid, a solução analítica para a equação de Laplace (eq. 01) pode ser obtida da seguinte maneira:

- Na figura 5.1 demonstra-se que não há variação de potencial nas direções y e z, mas apenas na direção x. Portanto, a equação de Laplace se reduz a:

$$\frac{d^2\phi}{dx^2} = 0 \tag{30}$$

- Para a segunda derivada de ϕ em relação a x ser zero, a primeira derivada deve ser igual a uma constante:

$$\frac{d\phi}{dx} = C_1 \tag{31}$$

ou:

$$d\phi = C_1 dx \tag{32}$$

Integrando:

$$\int d\phi = \int C_1 dx \tag{33}$$

ou:

$$\phi = C_1 x + C_2 \tag{34}$$

- Admitindo um valor potencial em x=0 igual a um valor arbitrário *P* e no final do grid ($x_{máx}$), ϕ=0, tem-se:

$$P = C_1 0 + C_2 \Rightarrow C_2 = P \tag{35}$$

e

$$0 = C_1 x_{máx} + C_2 \Rightarrow C_1 = -\frac{P}{x_{máx}} \tag{36}$$

- Introduzindo os valores de C_1 e C_2 na equação 34, tem-se:

$$\phi = -\frac{P}{x_{máx}} x + P \tag{37}$$

- Para um valor no meio do grid, ou seja, $0{,}5x_{máx}$ tem-se:

$$\phi = -\frac{P}{x_{máx}} \frac{1}{2} x_{máx} + P \Rightarrow \phi = \frac{P}{2} \tag{38}$$

Este é o resultado utilizado como referência inicial para o algoritmo usado na discretização.

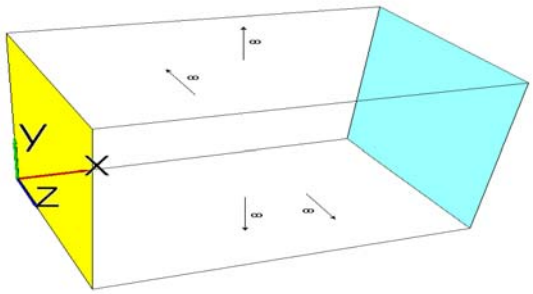


Figura 5.1 – Grid representativo para a resolução da equação de Laplace. A face amarela possui um potencial P e a face azul um potencial nulo.

Para um valor de potencial *P* qualquer, o método iterativo de solução utilizado para a eq. (7) tem seu erro de convergência reduzido a medida que se aumentam as iterações. A figura 5.2 mostra a redução do erro em função do número de iterações. Inicialmente, para um erro local de 0,1%, ou 0,001 no valor da velocidade *Vx*, foi necessário repetir 4647 vezes a iteração, onde se calculou, a cada iteração, 96000 pontos. O tempo necessário para chegar a este valor foi de 189 segundos. O cálculo foi executado em um computador Pentium Quad-Core, 2,4 GHz, com 4 GB de memória RAM. Para um erro de 1% foram necessárias 2186 iterações em 88 segundos.

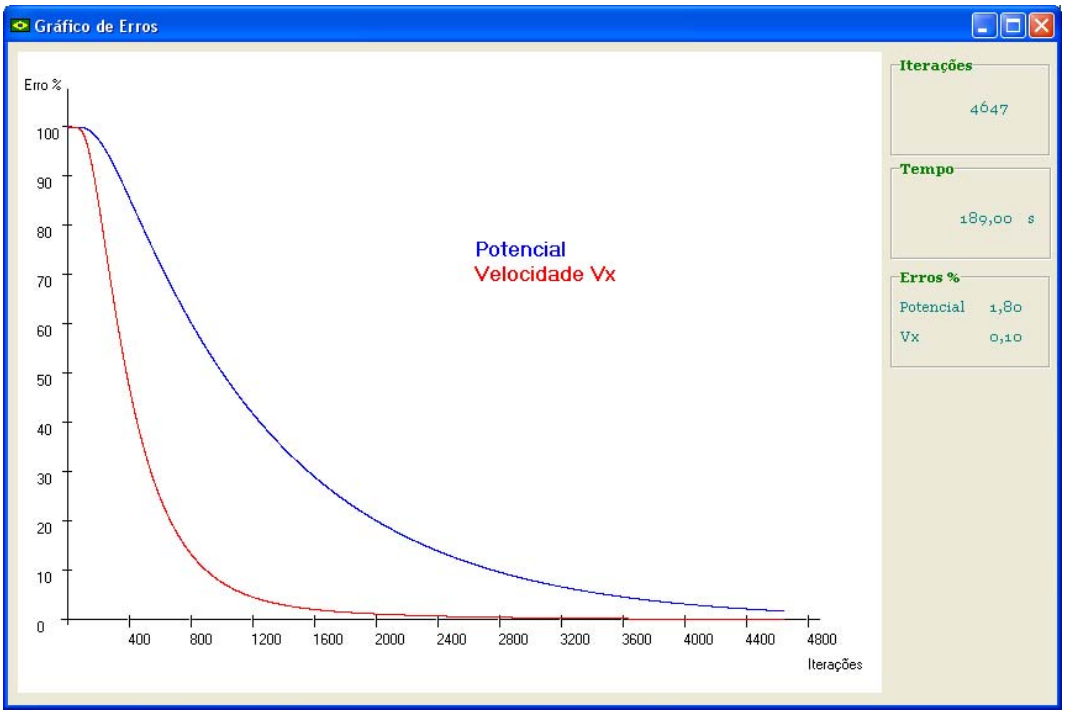


Figura 5.2 – Erro local relativo em função do número de iterações.

Uma característica interessante deste modelo foi a capacidade de restauração durante a colocação e retirada de elementos (figuras geométricas) no interior do grid. Ao se estabelecer, inicialmente um erro de 0,01 (1%) para a velocidade *Vx* (após 2186 iterações), inserir um objeto, iteragir 2186 vezes novamente e retirar esse objeto, o método levou apenas 29 segundos e 649 iterações para chegar ao mesmo erro novamente (1%) (figura 5.3).

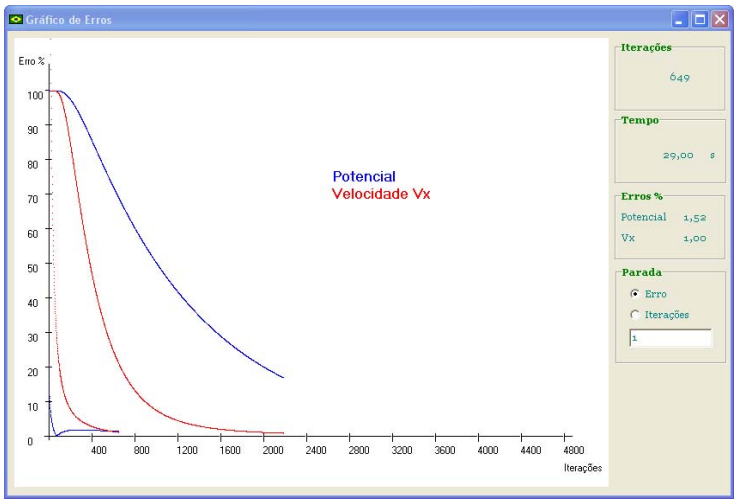


Figura 5.3 – Erro relativo após a inserção e retirada de figuras geométricas do grid.

Como já se possuía a solução analítica neste caso, optou-se por inicializar o grid com os valores reais dos potenciais, baseado na equação 38. Desta forma, para atingir um erro de 1% bastou 1 iteração, pois o valor de cada ponto coincidia com o valor da solução analítica. Repetindo a mesma experiência com a colocação de objetos, 2186 iterações e a sua retirada, o erro de 1% foi atingido com 658 iterações e 27 segundos, repetindo a mesma resposta anterior (figura 5.4). A experiência foi repetida com os objetos pré-disponíveis no simulador, como um cubo, uma esfera e um perfil de asa imersos no grid e com alterações no número de iterações aplicadas enquanto estes objetos estavam participando. Obteve-se a mesma resposta no número de iterações e tempo para se atingir o erro de 0,01 (1%). Desta maneira demonstra-se a consistência do método utilizado no simulador.

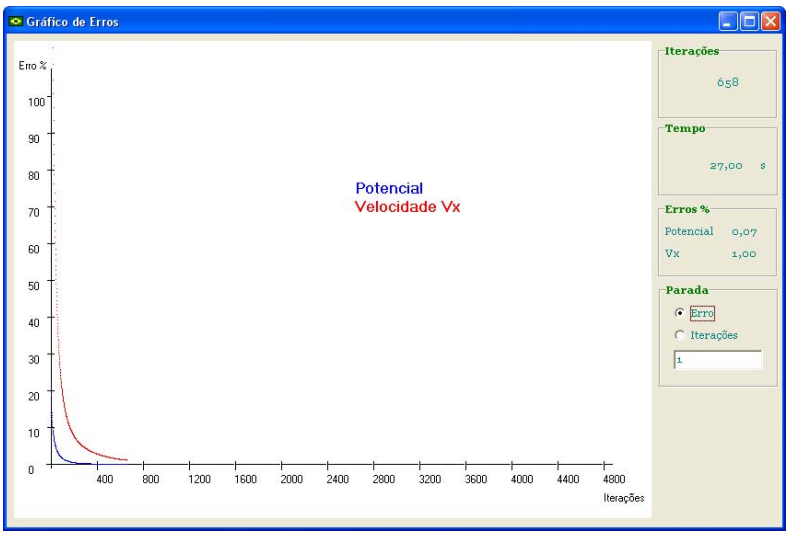


Figura 5.4 – Verificação do erro relativo.

Ao se inserir uma geometria dentro do domínio computacional, uma nova situação surge e, com as novas condições de contorno, o potencial dos pontos é recalculado imediatamente. Por se tratar de superfícies escorregadias, percebe-se, pela visualização que o algoritmo funciona perfeitamente (figuras 5.5 e 5.6). As linhas de fluxo circundam as figuras geométricas e, como era de se esperar, não se cruzam em nenhum momento.

A solução numérica de um fluxo passando por uma superfície semicircular pode ser vista em Greenberg, (1998), onde existe um exemplo resolvido. A representação visual é semelhante às obtidas no simulador, o que o valida novamente, e demonstra sua capacidade de resolução e visualização.

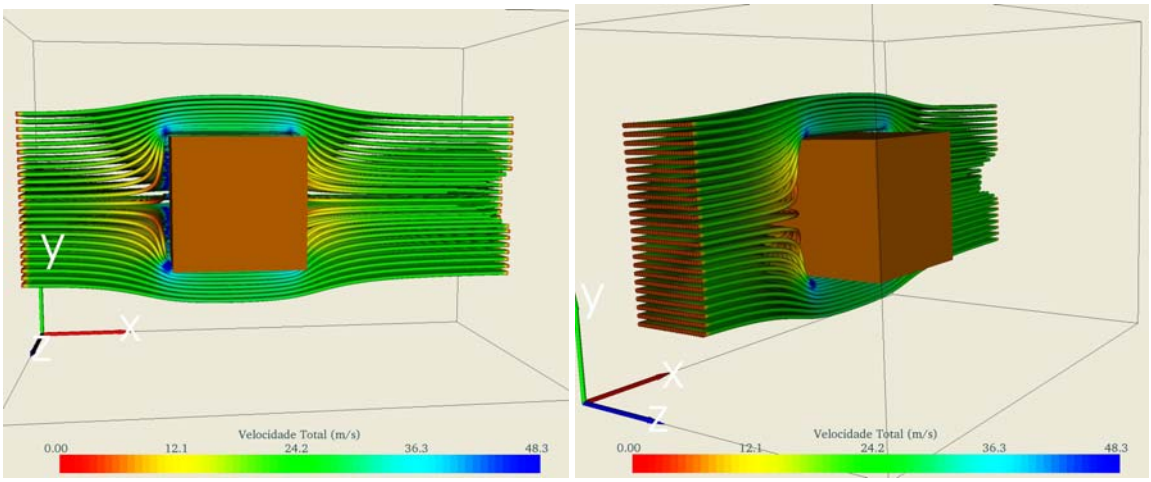


Figura 5.5 – 576 streamlines através de um cubo .

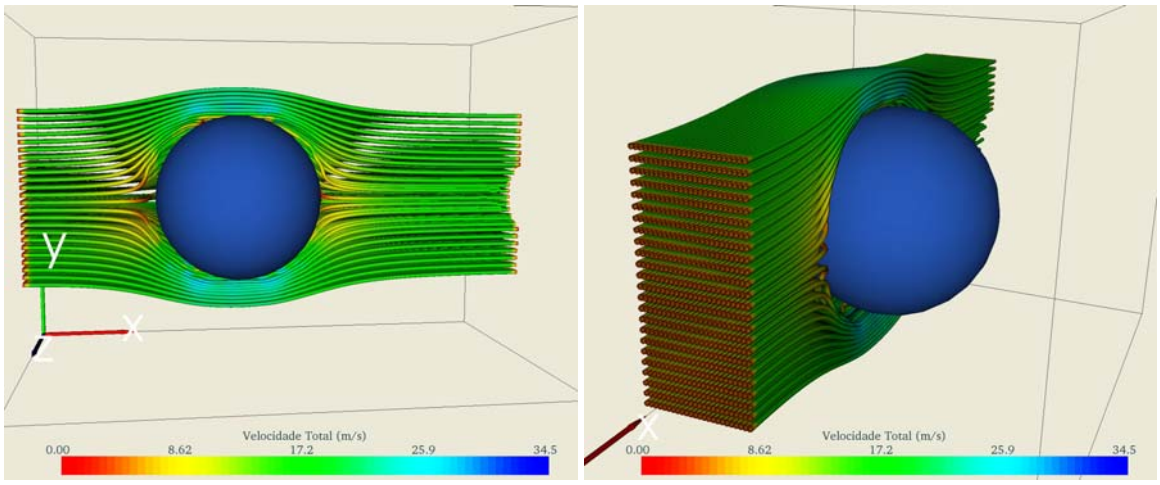
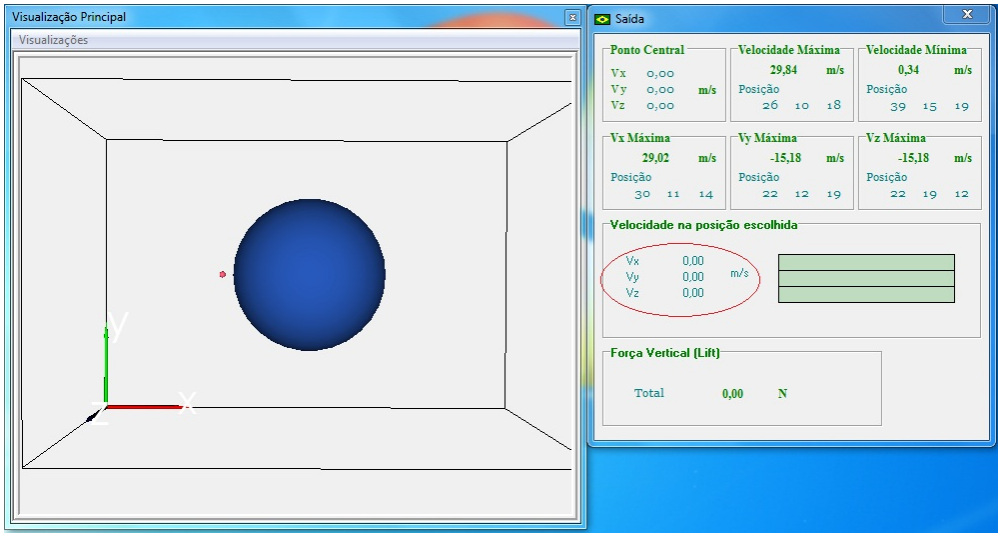


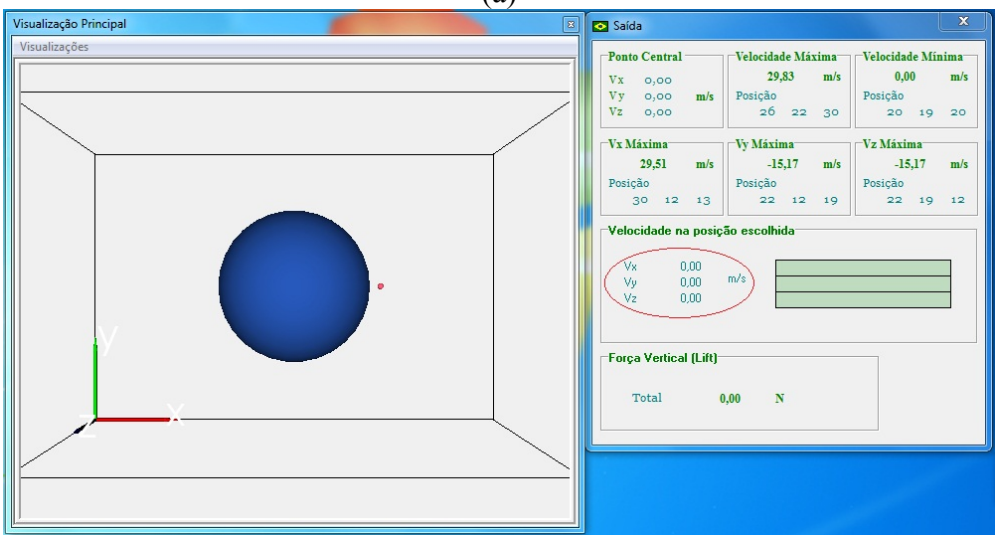
Figura 5.6 - 576 streamlines através de uma esfera

Para confirmar numericamente o resultado visual obtido, foram selecionado quatro pontos para a obtenção do resultado das velocidades. O primeiro e segundo ponto encontram-se imediatamente à frente e atrás da esfera, no sentido do fluxo (da esquerda para a direita) e o terceiro e quarto ponto acima e abaixo do centro geométrico, ligeiramente superior e inferior ao seu raio (figuras 5-8 e 5-8).

Percebe-se que os valores obtidos no primeiro e segundo casos representam o resultado esperado, ou seja, velocidade nula (figura 5.7). Para o segundo caso a componente horizontal da velocidade (*Vx*) apresenta um valor superior à velocidade inicial (20 m/s), o que era esperado, já as componentes *Vy* e *Vz* são nulas (figura 5.8). A simetria dos resultados também demonstra a consistência do método aplicado.



(a)



(b)

Figura 5.7 – Componentes das velocidades à frente (a) e atrás (b) de uma esfera.

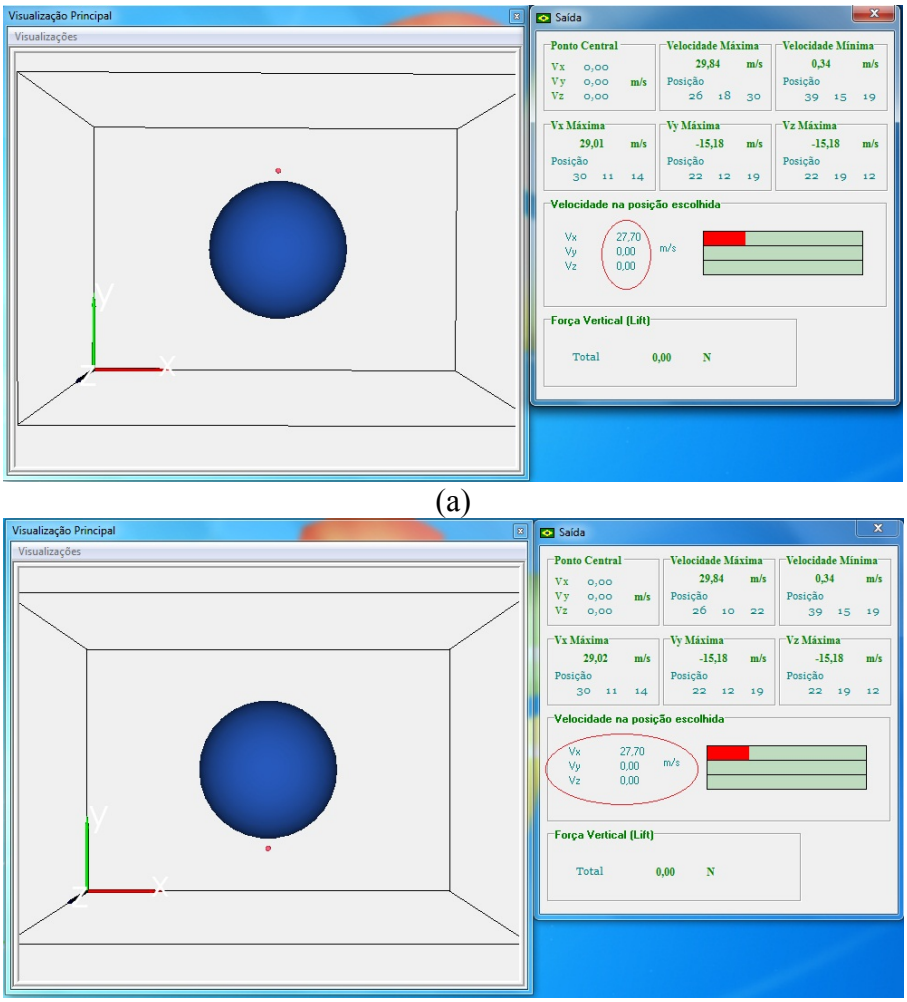


Figura 5.8 - Componentes das velocidades acima (a) e abaixo (b) de uma esfera.

5.1 Força de sustentação (Lift)

Sustentação é a componente da Resultante Aerodinâmica perpendicular ao vento relativo. A Resultante Aerodinâmica (RA) é uma força que surge em virtude do diferencial de pressão entre o intradorso e o extradorso do aerofólio e tende a empurrá-lo para cima, auxiliada ainda pela reação do ar (Terceira Lei de Newton) na parte inferior da mesma. Ela é representada como um vetor que, quando decomposto, dá origem a duas forças componentes que são: a força de sustentação e a força de arrasto. Graças a essa força o aerofólio é capaz de erguer-se. Se este for, por exemplo a asa de uma aeronave, esta alçará vôo. A sustentação é função da densidade do ar, do coeficiente de sustentação, da área da asa e da velocidade de vôo elevada ao quadrado, e seu símbolo é "L" (Lift, em Inglês).

Partindo da idéia da conservação da energia mecânica - característica encontrada mesmo em um líquido isento de forças viscosas – Danielis Bernoulli (1700 – 1782) mostrou que, em igualdade de nível, há uma diferença de pressões devida à diferente velocidade de

escoamento nos vários pontos de um fluido. Por exemplo, num dado ponto do fluido, no qual este último esteja em repouso, a pressão aí será maior, pois está associada a uma forma de energia potencial, ao passo que num outro ponto onde o fluido se move rapidamente a pressão é menor, pois nessa posição à velocidade do fluido corresponde uma dose de energia cinética. Dado que a energia total é a mesma em todos os pontos do filete líquido, nos pontos de maior energia cinética a pressão é menor e vice-versa. A própria força de sustentação dos aviões se deve à existência da diferença de pressões, que Bernoulli tão bem assinalou. De fato, como o trajeto que os filetes de ar devem percorrer na parte superior do perfil da asa é bem maior que na parte inferior, estabelece-se uma diferença de velocidade nos filetes, de forma que, onde a velocidade é maior, a pressão é menor. Essa diferença resulta numa força ascensional.

Como neste trabalho as velocidades são obtidas em todos os pontos do grid, ao se inserir uma forma geométrica é possível calcular as pressões que agem sobre elas, inferiores e superiores.

Um algoritmo, que identifica se existe um obstáculo acima ou abaixo de todos os ponto do grid, foi implementado e permitiu calcular e mostrar instantaneamente a força de sustentação existente. Devido ao escoamento ser irrotacional, invíscido e em regime permanente, foi possível aplicar a equação de Bernoulli, onde a pressão é proporcional ao quadrado da velocidade. Tendo os valores das pressões aplicadas e as áreas de aplicação, o cálculo da força de sustentação é feito de forma simples, somando-se todas as forças atuantes no objeto.

Inicialmente uma forma geométrica simétrica foi testada e, como era de se esperar, a força de sustentação foi nula (figura 5.9). Ao inserir um perfil de asa, obteve-se um resultado positivo, indicando uma força de sustentação. Este valor se altera em função das mudanças geométricas efetuadas no perfil superior da asa (figura 5.10).

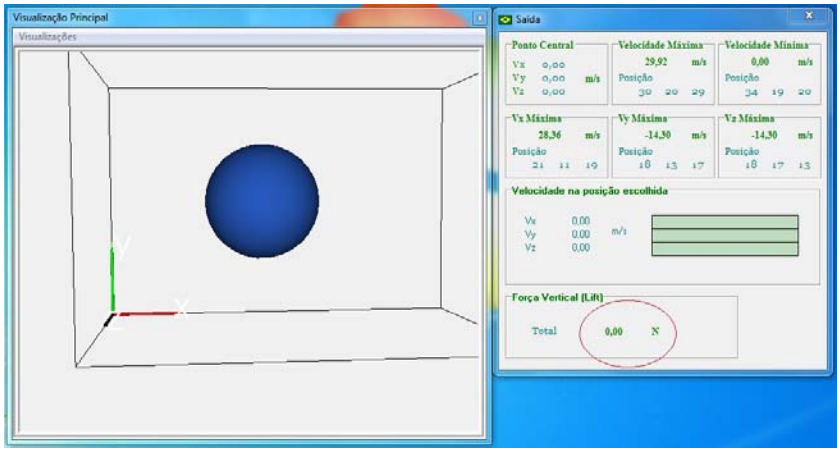
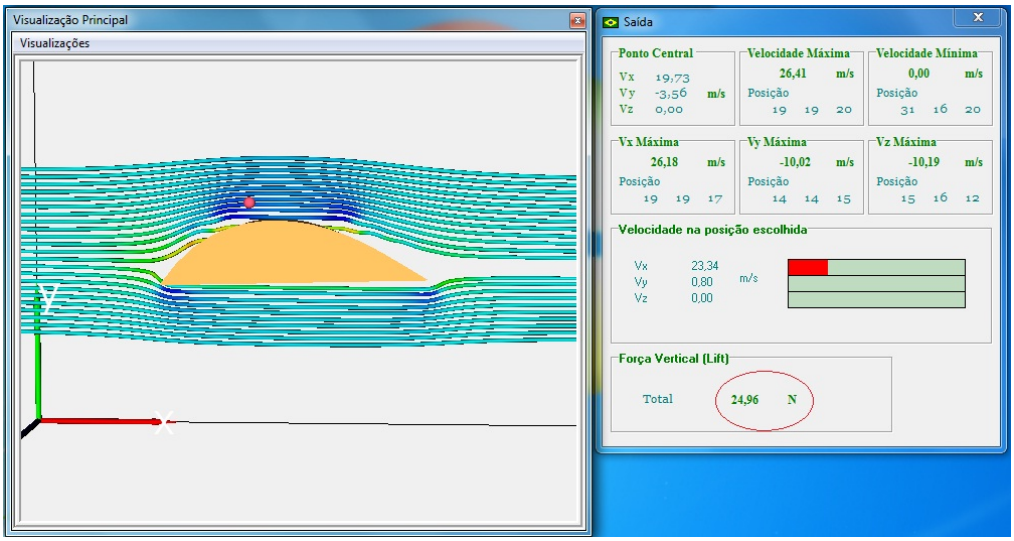
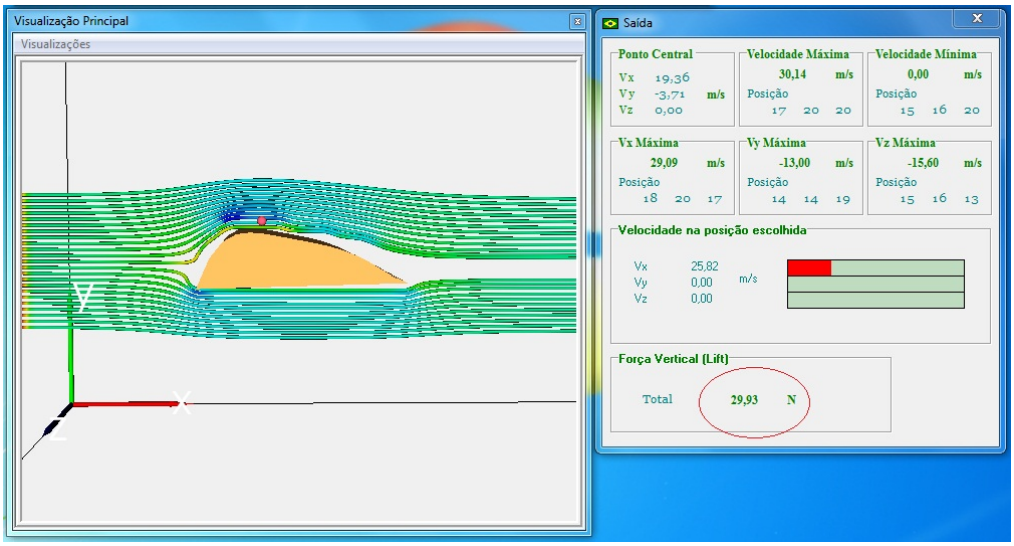


Figura 5.9 – Força de sustentação em uma esfera.



(a)



(b)

Figura 5.10 – Força de sustentação em um perfil de asa com fundo plano. (a) Força obtida igual a 24,96 N e (b) Força obtida igual a 29,93 N.

Ao se aplicar as dimensões do perfil de asa (área = 0,0255m², corda = 0,05m, ângulo = 2,88º) e valor de velocidade = 20m/s (72km/h) em outro simulador (*FoilSim II* 2D), disponível no site da NASA (http://www.grc.nasa.gov/WWW/K-12/airplane/foil2.html), o resultado da força de sustentação (Lift) encontrado foi de 26N. Com estes mesmos parâmetros foi encontrado o valor de 25,10N no simulador desenvolvido. Isto representa uma diferença de menos de 5%, o que demonstra coerência no modelo apresentado neste trabalho.

6 CONCLUSÃO

Segundo Maliska (1995), "A solução do escoamento turbulento supersônico sobre um aerofólio, usando computadores do tipo IBM 704, existente na década de 60, consumiria um tempo de computação de aproximadamente 30 anos, com custo de 10 milhões de dólares".

Hoje existem computadores com capacidade de processamento centenas de vezes superior aos do século passado, o que é um fator estimulante para o desenvolvimento de programas que requerem alto teor computacional.

O simulador descrito neste trabalho foi desenvolvido, inicialmente, para o cálculo e visualização de fluxos irrotacionais. Ele traz, de forma inédita e interativa, o uso da solução numérica da equação de Laplace, simultaneamente com um sistema de visualização 3D em tempo quase-real. Não existe sistema com a mesma natureza que permite grande interação e manipulação apoiada visualmente.

A potencialidade do algoritmo utilizado é demonstrada no cálculo de cada ponto do grid, onde se analisa toda sua vizinhança. Dependendo das condições de fronteira, descritas no capítulo 3, uma nova solução particular é utilizada para que se tenha o resultado adequado. Para a visualização das linhas de fluxo foi necessário usar uma integração dos valores obtidos. O método numérico utilizado foi o de Runge Kutta e o algoritmo desenvolvido permite a visualização de qualquer quantidade de linhas, a partir da origem, também escolhida pelo usuário. Como resultado positivo, as linhas de fluxo aparecem de forma realista e podem ser manipuladas interativamente.

Durante a construção do simulador foram testadas várias ferramentas de auxílio à programação, assim como, inicialmente, a visualização foi desenvolvida a partir de operações matriciais que geravam rotações e translações. Optou-se pela utilização de bibliotecas gráficas, escritas em C++, já otimizadas, encontradas gratuitamente na internet, como o VTK. Este procedimento induziu a novos estudos mas trouxe resultados visuais de maior qualidade.

Foram testados, também, alguns compiladores C++ e a linguagem C#, mas o resultado final praticamente não sofreu alterações, conforme é mostrado no capítulo 4.

As formas de apresentação da visualização de fluxos foram estudadas e estão presentes no capítulo 2. Procurou-se disponibilizar as que mais se mostram adequados aos fluidos irrotacionais, como streamlines, streamtubes, isolinhas e gradientes, para a seleção no

simulador. O usuário tem a liberdade de escolha da forma de visualização numa apresentação individual ou coletiva.

A proposta alcançada no trabalho desenvolvido é a qualidade da percepção visual, a possibilidade de manipulação de objetos e o cálculo e visualização de um fenômeno físico em tempo quase-real. Com um aplicativo desta natureza, desenvolvido inicialmente para fluxos irrotacionais mas com potencial de melhoramentos futuros devido ao constante crescimento computacional, um usuário simulará diversas formas geométricas e as adequará às suas necessidades de design e aerodinâmica. Tal procedimento permitirá às empresas reduzirem significativamente as horas de testes em túneis de vento reais, por exemplo, o que poderá acarretar a redução do custo do desenvolvimento de novos produtos.

Buscou-se a validação do programa, descrita no capítulo 5, onde os resultados numéricos e visuais obtidos comparam-se com soluções teóricas. Em um estudo de caso, foi utilizado o campo de velocidades para o cálculo da força de sustentação de algumas formas geométricas. O resultado também foi compatível com a realidade. Não se encontrou força de sustentação em formas geométricas simétricas e no perfil de asa o resultado se altera com a mudança da geometria.

Outra aplicação sugerida para o simulador desenvolvido é de caráter didático-pedagógico. A possibilidade de um estudante visualizar em tempo quase-real as linhas de fluxo em torno de um objeto e, ao fazer modificações dimensionais, visualizar imediatamente o novo caminho destas linhas, tornará o aprendizado mais interessante e atrativo.

Ainda, já pensando na utilização futura de monitores estereoscópicos, disponibilizou-se a opção da visualização estereoscópica, que pode ser feita com o auxílio de óculos polarizados ou diretamente nesses monitores. Este procedimento torna o trabalho mais imersivo e realista.

Por analisar espacialmente a vizinhança de cada ponto, o algoritmo computacional deste trabalho utiliza o cálculo na forma seqüencial, não sendo possível uma paralelização. Porém não é fator limitante para o desempenho. Para o cálculo de 96000 pontos, inicialmente realizado neste programa, um computador de dois núcleos atualiza cerca de 50 vezes por segundo. O algoritmo de visualização continua sendo o “gargalo” na qualidade de percepção. Ele foi desenvolvido usando as bibliotecas do VTK que, ainda, não são paralelizáveis. Ao utilizar a visualização das linhas de fluxo o desempenho reduz significativamente, caindo para 12 fps (frames por segundo) ao se utilizar 26 linhas e para 4 fps ao se utilizar 100 linhas de fluxo. Novas tecnologias, como a GPU, disponíveis, por exemplo, usando o software CUDA, da nVidia (2010) foram estudadas, mas ainda não são aplicáveis neste trabalho. Seria

necessária uma nova programação para visualização, utilizando a linguagem Cg, descrita no capítulo 4 e, desta maneira, todas as classes do VTK não seriam aproveitáveis. Ainda, durante o desenvolvimento, surgiram as bibliotecas de visualização vtkEdge (2010), que permitem o uso do cálculo paralelo para o uso das unidades de processamento paralelo (GPU) disponíveis nas placas gráficas. Embora as classes de visualização utilizadas neste trabalho, como a vtkStreamTracer, ainda não tenham sido atualizadas para a utilização desta tecnologia, os fóruns oficiais dos desenvolvedores do vtkEdge indicam que em breve ter-se-á estas bibliotecas. Desta maneira será possível aumentar significativamente o grid computacional ou inserir novos algoritmos de cálculo que contemplem fenômenos naturais como a turbulência.

Como trabalhos futuros, algumas implementações poderão ser desenvolvidas para obter-se um software com maiores recursos:

a) Importação de formas geométricas desenvolvidas em outros aplicativos CAD, para análise de fluxo e sustentação;

b) Exportação das formas geométricas estudadas num formato compatível com outros aplicativos;

c) Um novo algoritmo para o cálculo e a visualização que contemple modelos de turbulência;

d) Aumento do grid sem comprometer o rendimento computacional, com o uso de GPUs;

e) Exportação de dados numéricos, filmes e/ou fotografias;

f) Estudo da ergonomia do software e da cognitividade.

REFERÊNCIAS

Abraham, R. H., Shaw, C. D., 1992. *Dynamics – the Geometry of Behavior*. Addison-Wesley.

Albrecht, P., 1973. *Análise Numérica,* Livros técnicos e Científicos Editora S.A., Rio de Janeiro.

Alves, W. P., 2002. *C++ Builder 6: Desenvolva Aplicações para Windows,* Editora Érica, São Paulo.

ANSWER, 2010. http://www.acricfd.com/applications/answer/default.htm. Disponível em 19/02/2010.

Bancroft, G. V., Merritt, F. J., Plessel, T. C., Kelaita, P.G., McCabe, R. K., Globus, A., 1990. *FAST: A multiprocessed environment for visualization of computational fluid dynamics.* Proceedings of IEEE Visualization 90, p. 14–27.

Banks, D. C., 1994. *Illumination in diverse codimensions.* Proceedings of ACM SIGGRAPH 94, p. 327–334.

Batra, R. K., Hesselink, L., 1999. *Feature comparisons of 3D vector fields using earth mover's distance*. IEEE Visualization '99, p. 105–114.

Battke, H., Stalling, D., Hege H-C., 1997. *Fast line integral convolution for arbitrary surfaces in 3D.* Visualization and Mathematics, p. 181–195. Springer-Verlag, Heidelberg.

Becker, B. G., Lane, D|. A., Max, N L., 1995. *Unsteady flow volumes*. Proceedings of IEEE Visualization '95.

Becker, J., Rumpf, M., 1998. *Visualization of time-dependent velocity fields by texture transport.* Proceedings of Visualization in Scientific Computing '98, Eurographics, p. 91–102.

Berger, S., Gröller, E., 2000. *Color-table animation of fast oriented line integral convolution for vector field visualization.* WSCG 2000 Conference Proceedings, p. 4–11.

Bordoloi, U., Shen, H.-W., 2002. *Hardware accelerated interactive vector field visualization: A level of detail approach.* Proceedings of Eurographics '02, p. 605–614.

Boring, E., Pang, A., 1996. *Directional flow visualization of vector fields*. Proceedings of IEEE Visualization '96, p. 389–392.

Brill, M., Hagen, H., Rodrian, H.-C., Djatschin, W., Klimenko, S. V., 1994. *Streamball techniques for flow visualization.* Proceedings of the IEEE Conference on Visualization'94, p. 225–231.

Bryson, S., Levit, C., 1992. *The virtual wind tunnel*. IEEE Computer Graphics and Applications, vol 12, nº 04, p.25–34.

Bryson, S., 1992. *The Distributed Virtual Windtunnel,* http://citeseer.ist.psu.edu/images/f6/f6/59/90/1a8eca5b14a58084b5d27a3fbde8058e/1.png, (disponível em 08/05/2007).

Bürkle, D., Preußer, T., Rumpf, M., 2001. *Transport and anisotropic diffusion in time-dependent flow visualization.* Proceedings of IEEE Visualization '01, p. 61–67.

Cabral, B., Leedom, L., 1993. Imaging vector fields using line integral convolution. Proceedings of Computer Graphics, SIGGRAPH '93 , v. 27, p. 263–272.

Cabral, B., Leedom, C., 1995. *Highly parallel vector visual environments using line integral convolution*. Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing, p. 802–807.

Cai, W., Heng, P-A., 1997. *Principal stream surfaces*. Proceedings of the 8th Annual IEEE Conference on Visualization (VISU-97), p. 75–80.

Camões, M.F.G.F.C.; 2001. *Quantificação das Incertezas nas medicões analíticas.* http://www.eurachem.fc.ul.pt/Guia_Eurachem_P.pdf. Disponível em 05/10/2010.

Card, S. K., 1996. *Visualizing retrieved information: a survey.* IEEE Computer Graphics and Applications, vol 16(2), p. 63–67.

Carvalho, C.V.A., Martha L. F., Teixeira, W., 2005. *Fluxovento – Um Simulador Gráfico Interativo para o Estudo de Ventilação em Ambientes Construídos*, ENCAC-ELACAC 2005– VIII Encontro Nacional sobre Conforto no Ambiente Construído e IV Encontro Latino-Americano sobre Conforto no Ambiente Construído, UFAL, Maceió, AL, Out. 2005, ISBN85-89478-12-2, p. 11.

Castro, V., A. S., Ansejo, E. O. B., Silva, J., 2000. *Uma Ferramenta Interface na Simulação/Visualização de Fluidos.* Revista de Pesquisa e Pós-Graduação, Erechim, p 201-216.

CFX, 2010. http://www.ansys.com/products/fluid-dynamics/cfx/. Disponível em 18-02-1010.

Cham, 2010. http://www.cham.co.uk/default.php. Disponível em 22-02-2010.

Clyne, J., Dennis, J., 1999. *Interactive direct volume rendering of time-varying data*. Proceedings of Data Visualization ’99, Eurographics. p. 109–120.

Cook, M.J.,1998. http://www.lboro.ac.uk/departments/cv/staff/docs/227/appendices/appa.pdf. Disponível em 07/10/2010.

Crawfis, R., Max, N., 1993. *Texture splats for 3D scalar and vector field visualization*. Proceedings of IEEE Visualization ’93, p. 261–267.

Crawfis, R., Max, N., Becker, B., Cabral, B., 1993. *Volume rendering of 3D scalar and vector fields at LLNL.* Proceedings, Supercomputing ’93: Portland, Oregon, November 15–19, p 570–576.

Crawfis, R. A., Shen, H-W., Max, N., 2000. *Flow visualization techniques for CFD using volume rendering*. 9th International Symposium on Flow Visualization, p 64/1–64/10.

Deitel, H.M., Deitel, P.J., 2001. *C++ Como Programar,* Bookman, Porto Alegre.

Diewald, U., Preußer, T., Rumpf, M., 2000. *Anisotropic diffusion in vector field visualization on Euclidean domains and surfaces.* Proceedings of IEEE Transactions on Visualization and Computer Graphics, vol 6(2), p.139–149.

Doleisch, H., Hauser, H., 2002. *Smooth Brushing for Focus and Context Visualization of Simulation Data in 3D.* Proceedings of WSCG 2002, International Conference in Central Europe on Computer Graphics, Visualization and Digital Interactive Media, Plzen, República Checa.

Earnshaw, R. A.; Wiseman, N., 1992. *An introductory guide to scientific visualization.* Springer-Verlag, New York.

Ebert, D. S., Rheingans, P., 2000. *Volume illustration: Non-photorealistic rendering of volume models.* Proceedings of IEEE Visualization 2000, p. 195–202.

Ebert, D. S., Yagel, R., Scott, J., Kurzion, Y., 1994. *Volume rendering methods for computational fluid dynamics visualization*, Proceedings of the IEEE Conference on Visualization, Los Alamitos, CA, USA, October, p. 232– 239.

Fernando, R.; Kilgard, M., 2003. *The Cg Tutorial – The Definitive Guide to Programmable Real-Time Graphics.* Addison-Wesley, Boston, CA.

Fenlon, A.J., David, T., Walton, J.P.R.B.,2000. *An Integrated Visualization and Design Toolkit for Flexible Prosthetic Heart Valves.* Proceedings of the 11th Annual IEEE Conference on Visualization (Visualization 2000), p. 453-456.

FisComp, 2010. http://omnis.if.ufrj.br/~carlos/fiscomp/fiscomp.html. Disponível em 05/10/2010.

Flowmaster, 2010. http://www.flowmaster.com/index.html. Disponível em 22-02-2010.

Fluent, 2010. http://www.ansys.com/products/fluid-dynamics/fluent/. Disponível em 18-02-2010.

Fluidyn, 2010. http://www.fluidyn.com/Home_English/Home_English.htm. Disponível em 22-02-2010.

FOAM, 2010. http://www.openfoam.com/. Disponível em 22-02-2010.

Forssell, L. K., 1994. *Visualizing flow over curvilinear grid surfaces using line integral convolution.* Proceedings of the IEEE Conference on Visualization'94, p. 240–247, USA.

Forssell, L. K., Cohen, S. D., 1995. *Using line integral convolution for flow visualization: Curvilinear grids, variablespeed animation, and unsteady flows*. IEEE Transactions on Visualization and Computer Graphics, vol. 1(2), p.133–141.

Fortuna, A.O., 2000. *Técnicas Computacionais para Dinâmica dos Fluidos*: *Conceitos Básicos e Aplicações*, Edusp.

Fox, R.W., McDonald, A.T., 2001. *Introdução a Mecânica dos Fluidos.* LTC, Rio de Janeiro.

Frühauf, T., 1996. *Raycasting vector fields*. Proceedings of the IEEE Conference on Visualization, Los Alamitos, October 27– November 1. p. 115–120

Fuhrmann, A. L., Gröller, E., 1998. *Real-time techniques for 3D flow visualization*. Proceedings of IEEE Visualization '98, p. 305–312.

Garcke, H., Preußer, T., Rumpf, M., Telea, A., Weikard, U., van Wijk, J. J., 2000. *A continuous clustering method for vector fields.* Proceedings of IEEE Visualization 2000, p. 351–358.

Garcke, H., Prußer, T., Rumpf, M., Telea, A. C., Weikard, U., van Wijk, J. J., 2001. *A phase field model for continuous clustering on vector fields*. Proceedings of IEEE Transactions on Visualization and Computer Graphics, vol 7(3), p. 230– 241.

Gerris, 2010. http://gfs.sourceforge.net/wiki/index.php/Main_Page. Disponível em 22-02-2010.

Giraldi, G., Feijóo, R., 2008. *Técnicas para Visualização de Vórtices.* http://virtual01.lncc.br/monografia/monografia3/node3.html . Disponível em 28/05/08.

Glau, T., 1999. *Exploring instationary fluid flows by interactive volume movies*. Proccedings of Data Visualization '99, Eurographics, p.277–283.

Globus, A., Levit, C., Lasinski, T., 1991. *A tool for visualizing the topology of three-dimensional vector fields.* Proceedings of 2nd conference in Visualization '91, p. 33–40.

Grant, J., Erlebacher, G., O'Brien, J., 2002. *Case study: Visualization of thermoclines in the ocean using Lagrangian-Eulerian time surfaces*. Proceedings of IEEE Visualization '02, p. 529–532.

Greenberg, M. D., 1998. *Advanced Engineering Mathematics.* Prentice-Hall, New Jersey, USA.

Guthe, S., Gumhold, S., Straßer, W., 2002. *Interactive visualization of volumetric vector fields using texture based particles.* WSCG 2002 Conference Proceedings, p. 33–41.

Hauser, H., Laramee, R. S., Doloisch, H., 2002. *State-of-the-Art Report 2002 in Flow Visualization.* Technical Report TR-VRVis-2002-003, VRVis Research Center, Vienna, Austria.

Heckel, B., Weber, G. H., Hamann, B., Joy, K. I., 1999. *Construction of vector field hierarchies.* Proceedings of IEEE Visualization '99, p. 19–26, San Francisco.

Hege, H., Stalling, D., 1998. *Fast LIC with piecewise polynomial filter kernels*. Mathematical Visualization, p. 295–314. Springer Verlag, Heidelberg.

Heidrich, W., Westermann, R., Seidel, H.-P., Ertl, T., 1999. *Applications of pixel textures in visualization and realistic image synthesis*. ACM Symposium on Interactive 3D Graphics, p. 127–134.

Helman, J. L., Hesselink, L., 1991. *Visualizing vector field topology in fluid flows*. Proceedings of IEEE Computer Graphics and Applications, vol 11(3), p.36–46.

Henze, Chris., 1998. *Feature detection in linked derived spaces*. IEEE Visualization ’98, p. 87–94.

Hesselink, L., Frits, H., Post, Van Wijk, J. J., 1994. *Research issues in vector and tensor field visualization.* IEEE Computer Graphics and Applications, 14(2):76–79.

Hultquist, J. P. M., 1990. *Interactive numerical flow visualization using stream surfaces.* Computing Systems in Engineering, vol. 1(2-4), p. 349–353.

Interrante.V. L., 1997. *Illustrating surface shape in volume data via principal direction-driven 3D line integral convolution*. Proceedings of SIGGRAPH‘97 Conference, p. 109–116.

Interrante, V., Grosch, C., 1997. *Strategies for effectively visualizing 3D flow with volume LIC.* Proceedings of IEEE Visualization ’97, p. 421–424.

Interrante, V., Grosch, C., 1998. *Visualizing 3D flow*. IEEE Computer Graphics & Applications, vol. 18(4) p.49–53.

Jeff P. M. Hultquist, J. P. M., 1990. *Interactive numerical flow visualization using Stream Surfaces*. Computing Systems in Engineering, 1(2-4):349–353, 1990.

Iescheck, A.L., 2006. *Representação e Visualização Volumétrica de dados Espaciais para avaliação de Solos.* Tese de Doutorado. Curso de Pós-Graduação em Ciências Geodésicas, Universidade Federal do Paraná.

Jobard, B., Lefer, W., 1997. *The motion map: Efficient computation of steady flow animations.* Proceedings of the 8th Annual IEEE Conference on Visualization (VISU-97), p. 323–328.

Jobard, B., Lefer, W., 1997. *Creating evenly–spaced streamlines of arbitrary density*. Proceedings of the Eurographics Workshop, volume 7 of Visualization in Scientific Computing ’97, Boulogne-sur-Mer, France.

Jobard, B., Erlebacher, G., Hussaini, M., Y., 2000. *Hardware-accelerated texture advection*. Proceedings of IEEE Visualization 2000, p. 155–162.

Jobard, B., Erlebacher, G., Hussaini, M. Y., 2001. *Lagrangian-eulerian advection for unsteady flow visualization.* Proceedings of IEEE Visualization’01.

Jobard, B., Lefer, W., 2000. *Unsteady flow visualization by animating evenly-spaced streamlines.* Eurographics Computer Graphics Proceedings, volume 19(3).

Jobard, B., Lefer, W., 2001. *Multiresolution flow visualization*. WSCG ’01, 5-9 Fevereiro. Plzen, República Tcheca.

Kao, D., Zhang, B., Kim, K., Pang, A., 2001. *3d flow visualization using texture advection*. Proceedings of International Conference on Computer Graphics and Imaging ’01, Honolulu, Hawai.

Kenwright, D., Lane, D. A., 1996. *Interactive Time- Dependent Particle Tracing Using Tetrahedral Decomposition*. Proceedings of IEEE Transactions on Visualization and Computer Graphics, p.120–129.

Kenwright, D., Haimes, R., 1997. *Vortex identification - applications in aerodynamics*. Proceedings of IEEE Visualization ´ 97, p. 413–416.

Kenwright, D. N., 1998. *Automatic detection of open and closed separation and attachment lines*. IEEE Visualization ’98, p. 151–158.

Kirby, R M., Marmanis, H., Laidlaw, D. H., 1999. *Visualizing multivalued data from 2D incompressible flows using concepts from painting*. In David S. Ebert, Markus Gross, and Bernd Hamann, editors, Proceedings of the 1999 IEEE Conference on Visualization (VIS-99), pages 333–340, N.Y.

Kiu, M., Banks, D. C., 1996. *Multi-frequency noise for LIC*. Proceedings of the IEEE Conference on Visualization, p. 121–126, Los Alamitos.

Khouas, L., Odet, C., Friboulet, D., 1999. 2D vector field visualization using fur-like texture. Proceedings of Eurographics Data Visualization ’99, p. 35–44.

Klassen, R. V., Harrington, S. J., 1991. *Shadowed hedgehogs: A technique for visualizing 2D slices of 3D vector fields*. Proceedings of IEEE Visualization ’91, p. 148–153.

Krüger, J., Kipfer, P., Konclratieva, P., Westermann., R., 2005. *A particle system for interactive visualization of 3D flows,* Transactions on Visualization and Computer Graphics, volume 11, issue 6, pp. 744 – 756.

Laramee, R. S., Bergeron, R. D., 2002. *An Isosurface Continuity Algorithm for Super Adaptive Resolution Data.* CGI 2002, Computer Graphics International, Bradford, UK, Julho 1-5.

Laramee, R. S., 2002. *Interactive 3D Flow Visualization Using a Streamrunner*. In *CHI 2002, ACM Conference on Human Factors in Computing Systems, Extended Abstracts*, Minneapolis, Minnesota, April 20-25 2002. ACM SIGCHI, ACM Press. accepted for publication.

Lavin, Y., Batra, R. K., Hesselink, L.,1998. *Feature comparisons of vector fields using earth mover’s distance.* Proceedings of IEEE Visualization, p. 103–110,

de Leeuw, W. C., vanWijk, J. J., 1993. *A probe for local flow field visualization*. Proceedings of the Visualization ’93 Conference, p. 39–45.

de Leeuw, W. C., Pagendarm, H-C., Post, F. H., Waltzer, B., 1995. *Visual simulation of experimental oil-flow visualization by spot noise from numerical flow simulation*. Visualization in Scientific Computing ’95, p. 135–148.

de Leeuw, W. C., van Wijk, J. J., 1995. *Enhanced spot noise for vector field visualization*. Proceedings of IEEE Visualization ’95, p 233–239.

de Leeuw, W. C., Post, F. H., Vaatstra, R. W., 1996. *Visualization of turbulent flow by spot noise*. Virtual Environments and Scientific Visualization ’96, pages 286–295. Springer-Verlag Wien, April 1996.

de Leeuw, W. C., van Liere, R., 1997. *Divide and conquer spot noise*. Proceedings of Supercomputing’97 (CD-ROM), Amsterdam.

de Leeuw, W. C., van Liere, R, 1998. Comparing LIC and spot noise. Proceedings of IEEE Visualization ’98, p. 359–366.

de Leeuw, W. C., van Liere, R., 1999. *Visualization of global flow structures using multiple levels of topology*. Eurographics Data Visualization ’99, p. 45–52.

de Leeuw, W. C., van Liere, R., 1999. *Collapsing flow topology using area metrics*. Proceedings of IEEE Visualization ’99, p. 349–354.

Li, L., 2007. *Image-Based Streamline Generation and Rendering,* Proc. IEEE Conf. Visualization, pp. 630-640.

Liu, Z. P., Moorhead, R. J., 2002. *AUFLIC: An accelerated algorithm for unsteady flow line integral convolution.* EG / IEEE TCVG Symposium on Visualization ’02, p. 43–52.

Lodha, S. K., Renteria, J. C., Roskin, K. M., 2000. *Topology preserving compression of 2D vector fields.* Proceedings of IEEE Visualization 2000, p. 343–350.

Löffelmann, H., König, A., Gröller, E., 1997. *Fast visualization of 2D dynamical systems by the use of virtual ink droplets*. 13th Spring Conference on Computer Graphics, p. 111–118. Comenius University, Bratislava, Slovakia.

Löffelmann, H., Mroz, H., Gröller, E., 1997. *Hierarchical streamarrows for the visualization of dynamical systems*. Visualization in Scientific Computing ’97, Eurographics, p. 155–164. Wien New York.

Löffelmann, H., Mroz, L., Gröller, E., Purgathofer, W., 1997. *Stream arrows: Enhancing the use of stream surfaces for the visualization of dynamical systems.* The Visual Computer, vol. 13(8), p.359–369.

Löffelmann, H., Gröller, E., 1998. *Enhancing the visualization of characteristic structures in dynamical systems*. Visualization in Scientific Computing ’98, Eurographics, p. 59–68. Springer-Verlag Wien, New York.

Löffelmann, H., Kucera, T., Gröller, E., 1998. *Visualizing poincaré maps together with the underlying flow.* Mathematical Visualization, p. 315–328. Springer Verlag, Heidelberg.

Maliska, C. R., 1995. *Transferência de calor e mecânica dos fluidos computacional*. 1ª ed. LTC, Rio de Janeiro.

Mao, X., Kikukawa, M., Fujita, N., Imamiya, A., 1997. *Line integral convolution for 3D surfaces*. Visualization in Scientific Computing ’97. Proceedings of the Eurographics Workshop in Boulogne-sur- Mer, France, p. 57–70.

Mao, X., Hatanaka, Y., Higashida, H., Imamiya, A., 1998. *Image-guided streamline placement on curvilinear grid surfaces*. Proceedings of IEEE Visualization ’98, p. 135–142.

Mao, X., Hong, L., Kaufman, A., Fujita, N., Kikukawa, M., 1998. *Multi-granularity noise for curvilinear grid LIC.* Graphics Interface, p. 193–200, junho 1998.

Martinez, M.L., 1995. *Uso de linhas de corrente, linhas de trajetória e linhas de emissão na visualização de fluxos,* Anais do VIII Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens (SIBGRAPI'95), pp. 303-304. São Carlos-SP.

Max, N., Crawfis, R., Williams, D., 1992. Visualizing wind velocities by advecting cloud textures. IEEE Visualization '92, p. 179–184.

Max, N., Becker, B., Crawfis, R., 1993. *Flow volumes for interactive vector field visualization*. Proceedings of the IEEE Visualization '93 Conference, p 19–24, San Jose, CA.

Max, N., Crawfis, R., Grant, C., 1994. *Visualizing 3D velocity fields near contour surfaces*. Proceedings of IEEE Visualization '94, p. 248–256.

Max, N., Becker, B., 1995. *Flow visualization using moving textures*. Proceedings of the ICASW/LaRC Symposium on Visualizing Time-Varying Data, p. 77–87.

Moran, P., Henze, C., Ellsworth, D., Bryson, S., Kenwrigth, D.. *Field Encapsulation Library.* http://www.nas.nasa.gov/Groups/VisTech/projects/fel/. Disponível em 22/10/2008.

Nielson, G. M., Hagen, H., Müller, H, 1997. *Scientific Visualization: Overviews, Methodologies, and Techniques*. IEEE Computer Society Press, Washington, DC, USA

Nielson, G. M., Jung, I-H., 1999. *Tools for computing tangent curves for linearly varying vector fields over tetrahedral domains*. Proceedings of IEEE Transactions on Visualization and Computer Graphics, vol 05, nº 04, p.360–372.

nVidia, 2010. http://www.nvidia.com/page/home.html. Disponível em 22/03/2010.

Okada, A., Kao, D. L., 1997. *Enhanced line integral convolution with flow feature detection*. Em SPIE Vol. 3017 Visual Data Exploration and Analysis IV, p. 206–217.

Ono, K., Matsumoto, H., Himeno, R., 2001. *Visualization of thermal flows in an automotive cabin with volume rendering method*. Proceedings of the Joint Eurographics - IEEE

TCVG Symposium on Visualizatation (VisSym-01). Wien, Austria, May 28–30, p 301–308, 2001.

Pagendarm, H. G., Henne, B., Rutten, M., 1999. *Detecting vortical phenomena in vector data by medium-scale correlation.* Proceedings of Conference on IEEE Visualization ’99, p. 409–412.

Paiva, A.C., Seixas, R.B., Gattass, M., 1999. *Introdução à Visualiação Volumétrica,* PUC-RioInf.MCC03/99. Rio de Janeiro, 106 p.

Peikert, R., Roth, M., 1998. *A higher-order method for finding vortex core lines*. IEEE Visualization ’98, p. 143– 150.

Peikert, R., Roth, M., 1999. *The parallel vectors operator - A vector field visualization primitive*. Procceedings of IEEE Visualization, p. 263–270.

Reinders, F., Post, F. H., Spoelder, H. J. W., 1997. *Feature extraction from pioneer venus OCPP data.* Proceedings of Visualization in Scientific Computing ’97, Eurographics, p. 85–94, New York.

Reinders, F., Spoelder, H. J. W., Post, F. H., 1998. *Experiments on the accuracy of feature extraction*. Visualization in Sientific Computing ’98, Eurographics, p. 49–58.

Reinders, F., Post, F. H., Spoelder, H. J. W., 1999. *Attribute-based feature tracking*. Data Visualization ’99, Eurographics, p. 63–72.

Reinders, F., Jacobson, M. E. D., Post, F. H., 2000. *Skeleton graph generation for feature shape description*. Eurographics Data Visualization 2000, p. 73–82.

Rezk-Salama, C., Hastreiter, P., Teitzel, C., Ertl, T., 1999. *Interactive exploration of volume line integral convolution based on 3D-texture mapping*. Proceedings of IEEE Visualization ’99, p. 233–240.

Risquet, C. P., 1998. *Visualizing 2D flows: Integrate and draw*. EG Workshop on Visualization in Scientific Computing, p. 132–142.

ROBERTSON, P. K., 1991. *A methodology for choosing data representations.* IEEE Computer Graphics & Applications, vol. 11, n. 3, p. 56-67, 1991.

Roth, M., Peikert, R., 1996. *Flow visualization for turbomachinery design*. Proceedings of IEEE Visualization, p. 381–384.

Röttger, S., Kraus, M., Ertl, T., 2000. *Hardware-accelerated volume and isosurface rendering based on cell-projection.*, Proceedings of IEEE Visualization 2000, p. 109– 116.

Sadarjoen, I. A., Boer, A. J., Post, F. H., Mynett, A. E., 1998. *Particle tracing in σ-transformed grids using tetrahedral 6-decomposition*. Proceedings of Visualization in Scientific Computing ’98, Eurographics, p. 71–80.

Sadarjoen, I. A., Post, F. H., Ma, B., Banks, D. C., Pagendarm, H-G., 1998. *Selective visualization of vortices in hydrodynamic flows*. Proceedings of IEEE Visualization ’98, p. 419– 422.

Sadarjoen, I. A., Post, F. H., 1999. *Geometric methods for vortex extraction*. Eurographics Data Visualization ’99, p 53–62.

Sadarjoen, I. A., Post. F. H., 2000. *Detection, Quantification, and Tracking of Vortices using Streamline Geometry*. Computers and Graphics, vol 24(3), p.333–341, Junho de 2000.

Sanna, A., Montrucchio, B., Montuschi, P. *A survey on visualization of vector fields by texturebased methods.* Research Developments in Pattern Recognition, 1(1), 2000. publisher: www.transworldresearch.com.

Sanna, A., Montrucchio, B., Arinaz, R., 2000. *Visualizing unsteady flows by adaptive streaklines.* Proceedings of WSCG 2000 Conference.

Scheuermann, G., Hagen, H., Krüger, H., Menzel, M., Rockwood, A., 1997. *Visualization of higher order singularities in vector fields*. Proceedings of IEEE Visualization ’97.

Scheuermann, G., Burbach, H., Hagen, H., 1999. *Visualizing planar vector fields with normal component using line integral convolution.* Proceedings of IEEE Visualization ’99, p. 255– 262, San Francisco.

Scheuermann, G., Tricoche, X., Hagen, H. *C1- interpolation for vector field topology visualization.* Proceedings of IEEE Visualization ’99, p. 271–278.

Schroeder, W., Volpe, C. R., Lorensen, W. E., 1991. *The stream polygon: A technique for 3D vector field visualization*. Proceedings of IEEE Visualization ’91, p. 126–132.

Schroeder, W., Martin, K., Lorensen, B., 2004. *The Visualization Toolkit,* 3th Edition, Prentice-Hall, 2004. *The VTK User’s Guide,* Kitware, Inc.

Schussman, G., Ma, K. L., Schissel, D., Evans, T., 2000. *Visualizing DIII-D Tokamak Magnetic Field Lines*. Proceedings of IEEE Visualization 2000 (Case Studies), p. 501–504.

Schulz, M., Reck, F., Bartelheimer, W., Ertl, T., 1999. *Interactive visualization of fluid dynamics simulations in locally refined cartesian grids*. IEEE Visualization ’99. San Francisco, p. 413–416.

Shen, H-W., Johnson, C. R., Ma, K-L., 1996. *Visualizing vector fields using line integral convolution and dye advection.* IEEE Volume Visualization Symposium, p. 63–70.

Shen, H-W., Kao, D. L., 1997. UFLIC: *A line integral convolution algorithm for visualizing unsteady flows*. Proceedings of IEEE Visualization ’97, p. 317–323.

Shen, H.-W, Kao, D. L., 1998. *A new line integral convolution algorithm for visualizing timevarying flow fields.* IEEE Transactions on Visualization and Computer Graphics, vol 4(2), p. 98–108.

Silver, D., Wang, X., 1997. *Tracking and Visualizing Turbulent 3D Features*. IEEE Transactions on Visualization and Computer Graphics, vol 3(2), p.129–141.

Silver, D., Wang, X., 1998. *Tracking features in unstructured datasets*. IEEE Visualization ’98, p. 79–86.

Shirley, P., Tuchman, A., 1990. *A polygonal approximation to direct scalar volume rendering*. Workshop on Volume Visualization, vol. 24, p. 63–70, San Diego.

Sperandio, D., Mendes, J.T., Silva, L.H.M., 2003. *Cálculo Numérico – Características Matemáticas e Computacionais dos Métodos Numéricos,* Prentice Hall, São Paulo.

Solid, 2010. http://www.directindustry.com/prod/solidworks-europe/cfd-software-14975-51333.html. Disponível em 16-03-2010.

Stalling, D., Hege, H., 1995. *Fast and resolution independent line integral convolution*. SIGGRAPH’95 Conference Proceedings. p. 249–256.

Stalling, D., 1997. *LIC on surfaces*. Em Texture Synthesis with Line Integral Convolution, p. 51–64. Siggraph ’97, Int. Conf. Computer Graphics and Interactive Techniques.

Sundquist, A., 2003. *Dynamic line integral convolution for visualizing streamline evolution*. IEEE Transactions on Visualization and Computer Graphics.

Swan, J. E., Lanzagorta, M., Maxwell, D., Kou, E., Uhlmann, J., Anderson, W., Shyu, H., Smith, W., 2000. *A computational steering system for studying microwave interactions with missile bodies*. Proceedings of IEEE Visualization 2000, p 441–444.

Tang, C-K., Medioni, G. G., 1998. *Extremal feature extraction from 3D vector and noisy scalar fields*. Proccedings in IEEE Visualization’98, p 95–102.

Teitzel, C., Grosso, R., Ertl, T., 1997. *Efficient and reliable integration methods for particle tracing in unsteady flows on discrete meshes*. Proceedings of Visualization in Scientific Computing ’97, Eurographics, p. 31–42.

Teitzel, C., Grosso, R., Ertl, T., 1997. *Line integral convolution on triangulated surfaces*. Proceedings of the Fifth International Conference in Central Europe on Computer Graphics and Visualization ’97, n. 8, p. 572– 581.

Teitzel, C., Ertl, T, 1999. *New approaches for particle tracing on sparse grids*. Data Visualization ’99, Eurographics, p. 73–84.

Telea, A., van Wijk, J. J., 1999. *Simplified representation of vector fields*. Proceedings of IEEE Visualization ’99, p. 35–42.

Teitzel, C., Grosso, R., Ertl, T., 1998. *Particle tracing on sparse grids*. Proceedings of Visualization in Sientific Computing ’98, Eurographics, p 81–90.

Treinish, L. A., 2000. *Multi-resolution visualization techniques for nested weather models*. Proceedings of IEEE Visualization 2000, p. 513–516.

Tricoche, X., Scheuermann, G., Hagen, H., 2000. *A topology simplification method for 2D vector fields.* Proceedings of the 11th Ann. IEEE Visualization Conference (Vis) 2000.

Tricoche, X., Scheuermann, G., Hagen, H., Clauss, S., 2001. Vector and tensor field topology simplification on irregular grids. Proceedings of the Joint Eurographics – IEEE TCVG Symposium on Visualizatation (VisSym-01), p. 107– 116.

Turk, G., Banks, D., 1996. *Image-guided streamline placement.* Proceedings of SIGGRAPH 96 Conference, p. 453–460.

Ueng, S-K., Sikorski, C., Ma, K-L., 1996. *Efficient Streamline, Streamribbon, and Streamtube Constructions on Unstructured Grids*. *IEEE Transactions on Visualization and Computer Graphics*, vol 2, nº 02, p.100–110.

Verma, V., Kao, D., Pang, A., 1999. *PLIC: Bridging the gap between streamlines and LIC*. Proceedings of the 1999 IEEE Conference on Visualization (VIS-99), p. 341–348.

Verma, V., Kao, D., Pang, A., 2000. *A flow-guided streamline seeding strategy*. Proceedings of IEEE Visualization 2000, p. 163–170.

Visual C++, 2009. http://msdn.microsoft.com/pt-br/visualc/ee340952.aspx. Disponível em 14/09/2009.

Visual C#, 2009. http://msdn.microsoft.com/en-us/vcsharp/default.aspx. Disponível em 14/09/2009.

VTK, 2010. http://www.vtk.org/. Disponível em 02/02/2010.

VTK.NET, 2009. http://vtkdotnet.sourceforge.net/. Disponível em 12/10/2009.

vtkEdge, 2010. http://www.vtkedge.org/. Disponível em 14/03/2010.

Weber, G., Kreylos, O., Ligocki, T., Shalf, J. M., Hagen, H., Hamann, B, Joy, K., I., 2001. *Extraction of crack-free isosurfaces from adaptive mesh refinement data*. Proceedings of the Joint Eurographics – IEEE TCVG Symposium on Visualizatation (VisSym-01), p. 25– 34, Austria.

van Walsum, T., Post, F. H., Silver, D., Post, F. J., 1996. *Feature Extraction and Iconic Visualization.* IEEE Transactions on Visualization and Computer Graphics, vol 2(2), p.111–119.

Wegenkittl, R., Gröller, E., 1997. *Fast oriented line integral convolution for vector field visualization via the Internet.* Proceedings of the 8th Annual IEEE Conference on Visualization (VISU-97), p. 309–316.

Wegenkittl, R. Gröller, E., Purgathofer, W., 1997. *Animating flow fields: Rendering of oriented line integral convolution.* IEEE Computer Animation ’97 Proceedings, p. 15–21.

Wegenkittl, R., Gröller, E., Purgathofer, W., 1997. *Visualizing the dynamical behavior of Wonderland*. IEEE Computer Graphics and Applications, vol. 17, nº. 6, p. 71-79, Novembro.

Weiskopf, D., Hopf, M., Ertl, T, 2001. *Hardware-accelerated visualization of time-varying 2D and 3D vector fields by texture advection via programmable per-pixel operations*. Vision, Modeling, and Visualization VMV ’01 Conference, p. 439–446.

Weiskopf, D., Erlebacher, D., Hopf, M., Ertl, t., 2002. *Hardware-accelerated Lagrangian-Eulerian texture advection for 2D flow visualization*. Vision, Modeling, and Visualization VMV ’02 Conference, p. 77–84.

Weiskopf, D., Erlebacher, G., 2004. *Flow Visualization Overview.* people.scs.fsu.edu/~erlebach/home/publications/overview_flow_weiskopf_erlebach_2004.pdf . (disponível em 20/04/08).

Westermann, R., 2001. *The rendering of unstructured grids revisited*. Proceedings of the Joint Eurographics – IEEE TCVG Symposium on Visualizatation (VisSym-01), p. 65– 74, Austria, Springer-Verlag.

Westermann, R., Johnson, C., Ertl, T., 2001. *Topology-preserving smoothing of vector fields.* IEEE Transactions on Visualization and Computer Graphics, vol. 7, nº 3, p. 222–229.

Westover, L., 1990. *Footprint evaluation for volume rendering*. Computer Graphics Proceedings of ACM SIGGRAPH 90, vol. 24, p.367–376.

Whitrow, G. J., 1993. O tempo na história – concepções do tempo da pré-história aos nossos dias. Jorge Zahar Editora, Rio de Janeiro.

wiseGEEK, 2010. http://www.wisegeek.com/what-is-a-divide-and-conquer-strategy.htm. (Disponível em 29/09/2010).

van Wijk, J. J., 1991. *Spot noise-texture synthesis for data visualization*. Computer Graphics (SIGGRAPH ’91 Proceedings), vol 25, p 309–318.

van Wijk., J. J., 1993. *Flow visualization with surface particles*. IEEE Computer Graphics and Applications, 13(4):18–24, July 1993

van Wijk, J. J., 1993. *Implicit stream surfaces.* Proceedings of IEEE Visualization ’93, p. 245–252.

van Wijk. J. J., 2002. *Image based flow visualization*. ACM Transactions on Graphics, vol. 21(3), p.745–754.

Windows, 2010. http://www.microsoft.com/windows/products/. (Disponível em 05/10/2010).

Wittenbrink, C. M., Pang, A. T., Lodha, S. K., 1996. *Glyphs for visualizing uncertainty in vectorfields*. IEEE Transactions on Visualization and Computer Graphics, vol 2(3), p.266–279.

Wong, P. C., Foote H., Leung, R., Jurrus, E., Adams, D., Thomas, J., 2000. *Vector fields simplification - a case study of visualizing climate modeling and simulation data sets*. Proceedings of the 11th Ann. IEEE Visualization Conference (Vis).

Zöckler, M., Stalling, D., Hege, H-C., 1996. *Interactive visualization of 3D-vector fields using illuminated streamlines.* Proceedings of IEEE Visualization ’96, p. 107–113, San Francisco, USA.

Zöckler, M., Stalling, D., Hege, H., 1997. *Parallel line integral convolution*. Parallel Computing, vol 23(7), p.975–989, July 1997.

ANEXOS

C++

A linguagem C foi desenvolvida tendo por base duas outras linguagens: BCPL criada por Martin Richards e B escrita por Ken Thompson e com influência da primeira. Dennis Richie a implementou num computador DEC PDP-11, rodando o sistema operacional UNIX (Alves, 2002). É uma linguagem de programação de uso geral, empregada no desenvolvimento dos mais diversos tipos de software, de sistema operacional a aplicativos de escritório ou jogos.

A linguagem C++ é uma evolução natural da linguagem C, desenvolvida por Bjarne Stroustrup nos laboratórios Bell (AT&T) entre 1983 e 1985. Inicialmente foi chamada de "C com classes", pois ele havia combinado as classes e a orientação a objetos da linguagem. É caracterizada por três propriedades:

- *Encapsulamento:* compreende a combinação de estrutura de dados com funções que a manipulam, também chamados de métodos.
- *Herança***:** capacidade de criar classes novas que herdam funções e estruturas de dados definidas em outra classe, sendo possível redefinir ou mesmo adicionar novos elementos.
- *Polimorfismo***:** recurso que permite nomear um determinado elemento da classe (método ou função) que é compartilhado por toda a hierarquia de objetos, mas realiza ações diferentes de acordo com a classe que a chamou.

É sem dúvida a linguagem mais utilizada no desenvolvimento de sistemas básicos e softwares sofisticados.

C++ Builder 6

A Borland ingressou no mercado de compiladores C em meados da década de 80, com o lançamento do Turbo C, semelhante ao Turbo Pascal, que na época se encontrava na versão 4.0. Em 1990 a Borland lança o Turbo C++, com um ambiente aperfeiçoado e totalmente compatível com a especificação 2.0 do C++, criada pela AT&T.

O C++ Builder é uma ferramenta para o desenvolvimento de aplicações que se baseia no conceito denominado RAD (Rapid Application Development – Desenvolvimento Rápido de Aplicação).

Microsoft Visual C++

O Visual C++ é o produto de IDE da Microsoft para as linguagens de programação C, C++ e C++/CLI. O Visual C++ 2008 fornece um ambiente de desenvolvimento para a criação de aplicativos com base no Microsoft Windows e no Microsoft .NET (Visual C++, 2009). É possível experimentar a programação com Visual C++ através da versão gratuita do Visual C++ Express.

Microsoft Visual C#

C# é uma linguagem orientada a objeto, simples, que permite aos programadores criar uma variedade de aplicativos. Combinado com o .NET Framework, o Visual C# 2008 possibilita a criação de aplicativos para o Windows, Web services, ferramentas de bancos de dados, componentes e controles. O Visual Studio é o IDE (Ambiente de Desenvolvimento Integrado) no qual os desenvolvedores trabalham ao criar programas em uma dentre várias linguagens, incluindo Visual C#, para o .NET Framework. Da mesma maneira que o Visual C++, existe uma versão gratuita do Visual C# para ser usada, disponível na internet (Visual C#, 2009). Estão disponíveis, também, diversos tutoriais e exemplos para desenvolvimentos de aplicativos em C# e C++.

Visualization Toolkit – VTK

O *Visualization ToolKit* (VTK) é um software 3D aberto, disponível gratuitamente para o desenvolvimento de sistemas de computação gráfica, processamento de imagem e visualização. O VTK consiste em uma biblioteca de classes C++ e várias camadas de interface interpretadas incluindo Tcl/Tk, Java e Python. O apoio profissional e produtos para VTK são fornecidos pela Kitware, Inc.. VTK suporta uma grande variedade de algoritmos de visualização incluindo métodos escalares, vetoriais, tensoriais, texturas e volumétricos, e avançadas técnicas de modelagem, como modelagem implícita, redução poligonal, suavização

de malhas, cortes, contorno e triangulação de Delaunay. Além disso, dezenas de algoritmos de imagem foram diretamente integrados para permitir ao utilizador misturar imagens 2D/3D, gráficos, algoritmos e dados. A concepção e implementação da biblioteca foram fortemente influenciadas por princípios objetos-orientados. O VTK foi instalado e testado em quase todas as plataformas baseadas em Unix, PC (Windows 98/ME/NT/2000/XP), e Mac OSX, Jaguar ou mais recente (VTK, 2010).

O modelo gráfico em VTK está em um nível de abstração superior ao de bibliotecas de renderização como OpenGL ou PEX. Isto significa que é muito mais fácil criar aplicativos úteis e visualização gráfica. Em VTK aplicações podem ser escritas diretamente em C++, Tcl, Java, ou Python.

APÊNDICE 1 Listagem de programas de computador

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop
#include <time.h>
#include <windows.h>
#include <math.h>

#include "vtkActor.h"
#include "vtkActor2D.h"
#include "vtkAxesActor.h"
#include "vtkCamera.h"
#include "vtkCaptionActor2D.h"
#include "vtkCellArray.h"
#include "vtkCleanPolyData.h"
#include "vtkClipPolyData.h"
#include "vtkConeSource.h"
#include "vtkContourFilter.h"
#include "vtkCubeSource.h"
#include "vtkCutter.h"
#include "vtkDataSetMapper.h"
#include "vtkDoubleArray.h"
#include "vtkFeatureEdges.h"
#include "vtkFloatArray.h"
#include "vtkGeometryFilter.h"
#include "vtkGlyph3D.h"
#include "vtkInteractorObserver.h"
#include "vtkLinearExtrusionFilter.h"
#include "vtkLineWidget.h"
#include "vtkLineSource.h"
#include "vtkLookupTable.h"
#include "vtkObject.h"
#include "vtkPlane.h"
#include "vtkPlaneSource.h"
#include "vtkPointData.h"
#include "vtkPoints.h"
#include "vtkPolyData.h"
#include "vtkPolyDataMapper.h"
#include "vtkPolyDataNormals.h"
#include "vtkProperty.h"
#include "vtkProperty2D.h"
#include "vtkPolyDataMapper.h"
#include "vtkRectilinearGridGeometryFilter.h"
#include "vtkRectilinearGrid.h"
#include "vtkRectilinearGridOutlineFilter.h"
#include "vtkRenderer.h"
#include "vtkRenderWindow.h"
```

```
#include "vtkRenderWindowInteractor.h"
#include "vtkRibbonFilter.h"
#include "vtkRuledSurfaceFilter.h"
#include "vtkRungeKutta4.h"
#include "vtkScalarBarActor.h"
#include "vtkScalarBarWidget.h"
#include "vtkSphereSource.h"
#include "vtkStreamLine.h"
#include "vtkStreamPoints.h"
#include "vtkStreamTracer.h"
#include "vtkStripper.h"
#include "vtkStructuredGrid.h"
#include "vtkStructuredGridOutlineFilter.h"
#include "vtkTextProperty.h"
#include "vtkTransform.h"
#include "vtkTransformPolyDataFilter.h"
#include "vtkTriangleFilter.h"
#include "vtkTubeFilter.h"
#include "vtkUnstructuredGrid.h"

#include "Tunel3D.h"
#include "sobre.h"
#include "Cortes.h"
#include "VisPrincipal.h"
#include "DadosSaida.h"
#include "Opcoes.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "vtkBorlandRenderWindow"
#pragma resource "*.dfm"
TPrincipal *Principal;

static float xv[57], yv[41], zv[41];

time_t inicio, fim;

//---------------------------------------------------------------------------
__fastcall TPrincipal::TPrincipal(TComponent* Owner) : TForm(Owner)
{

}
//---------------------------------------------------------------------------


void __fastcall TPrincipal::btnIniciarClick(TObject *Sender)
{
array = NULL;
vetores = NULL;
cubo = NULL;
```

```
Ux = 0;
Uy = 0;
Uz = 0;
intValorU = 0;
dblValorU = 0;
a = 50;
b = 40;
c = 40;
Parar = 0;
ContadorFrame = 0;
time (&inicio);
ContadorFPS = 0;

Vx = 400; //StrToInt(Trim(edtVx->Text));

for (i=0; i<=a; i++)
      for (j=0; j<=b; j++)
            for (k=0; k<=c; k++) {
            u[i][j][k] = 0.005;
            vx[i][j][k] = 0;
            vy[i][j][k] = 0;
            vz[i][j][k] = 0;
           }
for (i=0; i<=a; i++)
 xv[i] = i;
for (j=0; j<=b; j++)
 yv[j] = j;
for (k=0; k<=c; k++)
 zv[k] = k;


for (j=0; j<=b; j++)
      for (k=0; k<=c; k++)  {
      u[0][j][k] = Vx;
}


// **** Início da visualização  VTK

//VTK1->GetRenderer()->ResetCamera();
//formSobre->VTK2->GetRenderer()->ResetCamera();

// *** Criando os forms

delete formOpcoes;
delete formCortes;
delete formDadosSaida;
delete formVisPrincipal;
```

```
Application->CreateForm(__classid(TformOpcoes),&formOpcoes);
Application->CreateForm(__classid(TformCortes),&formCortes);
Application->CreateForm(__classid(TformDadosSaida),&formDadosSaida);
Application->CreateForm(__classid(TformVisPrincipal),&formVisPrincipal);

// *** Início da visualização dos pontos

vetores = vtkFloatArray::New();
  vetores->SetNumberOfComponents(3);

//vtkPoints *pontos = vtkPoints::New();

vtkFloatArray *xCoords = vtkFloatArray::New();
for (i=0; i<=a; i++) xCoords->InsertNextValue(xv[i]);

vtkFloatArray *yCoords = vtkFloatArray::New();
for (i=0; i<=b; i++) yCoords->InsertNextValue(yv[i]);

vtkFloatArray *zCoords = vtkFloatArray::New();
for (i=0; i<=c; i++) zCoords->InsertNextValue(zv[i]);

array = vtkDoubleArray::New();
array->SetNumberOfValues((a+1)*(b+1)*(c+1));
  for (k=0; k<=c; k++)
  for (j=0; j<=b; j++)
  for (i=0; i<=a; i++) {
    if (formOpcoes->mnuVtotal->Checked)
      array->SetValue(k*(a+1)*(b+1)+j*(a+1)+i, Vmaxgrad[i][j][k]);

    if (formOpcoes->mnuVx->Checked)
      array->SetValue(k*(a+1)*(b+1)+j*(a+1)+i, vx[i][j][k]);

    if (formOpcoes->mnuVy->Checked)
      array->SetValue(k*(a+1)*(b+1)+j*(a+1)+i, vy[i][j][k]);

    if (formOpcoes->mnuVz->Checked)
      array->SetValue(k*(a+1)*(b+1)+j*(a+1)+i, vz[i][j][k]);

      vetores->InsertTuple3(k*(a+1)*(b+1)+j*(a+1)+i, vx[i][j][k], vy[i][j][k], vz[i][j][k]);

  }

rgrid = vtkRectilinearGrid::New();
  rgrid->SetDimensions(a+1,b+1,c+1);
  rgrid->SetXCoordinates(xCoords);
  rgrid->SetYCoordinates(yCoords);
  rgrid->SetZCoordinates(zCoords);
  rgrid->GetPointData()->SetVectors(vetores);
  rgrid->GetPointData()->SetScalars(array);
```

```
if (btnIniciar->Down)
      tmrTempo->Enabled = true;

else
      tmrTempo->Enabled = false;

xCoords->Delete();
yCoords->Delete();
zCoords->Delete();
//rgrid->Delete();
//array->Delete();
//Table->Delete();
//plane->Delete();
//vetores->Delete();

// *** Eixos

vtkTextProperty *propriedadesTexto1 = vtkTextProperty::New();
   propriedadesTexto1->SetFontFamilyToTimes();
   propriedadesTexto1->SetColor(1, 0, 0);
   propriedadesTexto1->SetFontSize(0.8);
   propriedadesTexto1->SetShadow(2);

vtkAxesActor *eixos = vtkAxesActor::New();
  eixos->SetShaftTypeToCylinder();
  eixos->SetXAxisLabelText("x");
  eixos->SetYAxisLabelText("y");
  eixos->SetZAxisLabelText("z");
  eixos->SetTotalLength (13, 13, 13);
  eixos->SetConeRadius (0.13);
  eixos->SetCylinderRadius (0.025);
  eixos->SetPosition(-10, -10, -10);
  eixos->GetXAxisCaptionActor2D()->SetCaptionTextProperty(propriedadesTexto1);
  eixos->GetYAxisCaptionActor2D()->SetCaptionTextProperty(propriedadesTexto1);
  eixos->GetZAxisCaptionActor2D()->SetCaptionTextProperty(propriedadesTexto1);

  formVisPrincipal->VTK1->GetRenderer()->AddActor(eixos);
  formCortes->VTK2->GetRenderer()->AddActor(eixos);

eixos->Delete();
propriedadesTexto1->Delete();

// *** Fim da visualização dos eixos

formVisPrincipal->VTK1->Invalidate();
formCortes->VTK2->Invalidate();

}
//---------------------------------------------------------------------------
```

```
void __fastcall TPrincipal::tmrTempoTimer(TObject *Sender)
{
if (Parar != 1) {


// ***

dblVx = 0;
dblVy = 0;
dblVz = 0;
intVxi = 0;
intVyi = 0;
intVzi = 0;
intVxj = 0;
intVyj = 0;
intVzj = 0;
intVxk = 0;
intVyk = 0;
intVzk = 0;
Ux = 0;
Uy = 0;
Vmax = 0;
Vmin = Vx;
Vmax1 = 0;
Corte = formCortes->tkbHorizontal->Position;

if ( (formOpcoes->edtX1line->Text == "") )
X1line = 1;
else
X1line = StrToInt(Trim(formOpcoes->edtX1line->Text));

if ( (formOpcoes->edtX2line->Text == "") )
X2line = 1;
else
X2line = StrToInt(Trim(formOpcoes->edtX2line->Text));

if ( (formOpcoes->edtY1line->Text == "") )
Y1line = 1;
else
Y1line = StrToInt(Trim(formOpcoes->edtY1line->Text));

if ( (formOpcoes->edtY2line->Text == "") )
Y2line = 1;
else
Y2line = StrToInt(Trim(formOpcoes->edtY2line->Text));

if ( (formOpcoes->edtZ1line->Text == "") )
Z1line = 1;
else
```

```
Z1line = StrToInt(Trim(formOpcoes->edtZ1line->Text));

if ( (formOpcoes->edtZ2line->Text == "") )
Z2line = 1;
else
Z2line = StrToInt(Trim(formOpcoes->edtZ2line->Text));

if ( (formOpcoes->edtPlanoOrigemX->Text == "") )
StreamPlanoOrigemX = 10;
else
StreamPlanoOrigemX = StrToInt(Trim(formOpcoes->edtPlanoOrigemX->Text));

if ( (formOpcoes->edtPlanoOrigemY->Text == "") )
StreamPlanoOrigemY = 10;
else
StreamPlanoOrigemY = StrToInt(Trim(formOpcoes->edtPlanoOrigemY->Text));

if ( (formOpcoes->edtPlanoOrigemZ->Text == "") )
StreamPlanoOrigemZ = 10;
else
StreamPlanoOrigemZ = StrToInt(Trim(formOpcoes->edtPlanoOrigemZ->Text));


// *** Posição e dimensões do paralelepípedo ***

if ( (formOpcoes->edtCuboX->Text == "") )
CuboX = 14;
else
CuboX = StrToFloat(Trim(formOpcoes->edtCuboX->Text));

if ( (formOpcoes->edtCuboY->Text == "") )
CuboY = 14;
else
CuboY = StrToInt(Trim(formOpcoes->edtCuboY->Text));

if ( (formOpcoes->edtCuboZ->Text == "") )
CuboZ = 14;
else
CuboZ = StrToInt(Trim(formOpcoes->edtCuboZ->Text));


if ( (formOpcoes->edtCuboCentroX->Text == "") )
CuboCentroX = a/2;
else
CuboCentroX = StrToInt(Trim(formOpcoes->edtCuboCentroX->Text));

if ( (formOpcoes->edtCuboCentroY->Text == "") )
CuboCentroY = b/2;
else
CuboCentroY = StrToInt(Trim(formOpcoes->edtCuboCentroY->Text));
```

```
if ( (formOpcoes->edtCuboCentroZ->Text == "") )
CuboCentroZ = c/2;
else
CuboCentroZ = StrToInt(Trim(formOpcoes->edtCuboCentroZ->Text));

//*** Fim das condições do paralelepípedos ***


//*** Posição e dimensões da esfera e semi-esfera ***

if ( (formOpcoes->edtEsferaRaio->Text == "") )              // Raio da esfera
raio = 11;
else
raio = StrToInt(Trim(formOpcoes->edtEsferaRaio->Text));

if ( (formOpcoes->edtSemi1Raio->Text == "") )               // Raio da semi-esfera 1
raio1 = 10;
else
raio1 = StrToInt(Trim(formOpcoes->edtSemi1Raio->Text));

if ( (formOpcoes->edtSemi2Raio->Text == "") )               // Raio da semi-esfera  2
raio2 = 10;
else
raio2 = StrToInt(Trim(formOpcoes->edtSemi2Raio->Text));

if ( (formOpcoes->edtEsferaCentroX->Text == "") )
EsferaCentroX = a/2;
else
EsferaCentroX = StrToInt(Trim(formOpcoes->edtEsferaCentroX->Text));

if ( (formOpcoes->edtEsferaCentroY->Text == "") )
EsferaCentroY = b/2;
else
EsferaCentroY = StrToInt(Trim(formOpcoes->edtEsferaCentroY->Text));

if ( (formOpcoes->edtEsferaCentroZ->Text == "") )
EsferaCentroZ = c/2;
else
EsferaCentroZ = StrToInt(Trim(formOpcoes->edtEsferaCentroZ->Text));


if ( (formOpcoes->edtSemi1CentroX->Text == "") )
Semi1CentroX = a/2;
else
Semi1CentroX = StrToInt(Trim(formOpcoes->edtSemi1CentroX->Text));

if ( (formOpcoes->edtSemi1CentroY->Text == "") )
Semi1CentroY = b/2;
else
```

```
Semi1CentroY = StrToInt(Trim(formOpcoes->edtSemi1CentroY->Text));

if ( (formOpcoes->edtSemi1CentroZ->Text == "") )
Semi1CentroZ = c/2;
else
Semi1CentroZ = StrToInt(Trim(formOpcoes->edtSemi1CentroZ->Text));


if ( (formOpcoes->edtSemi2CentroX->Text == "") )
Semi2CentroX = a/2;
else
Semi2CentroX = StrToInt(Trim(formOpcoes->edtSemi2CentroX->Text));

if ( (formOpcoes->edtSemi2CentroY->Text == "") )
Semi2CentroY = b/2;
else
Semi2CentroY = StrToInt(Trim(formOpcoes->edtSemi2CentroY->Text));

if ( (formOpcoes->edtSemi2CentroZ->Text == "") )
Semi2CentroZ = c/2;
else
Semi2CentroZ = StrToInt(Trim(formOpcoes->edtSemi2CentroZ->Text));


//*** Fim das condições das esferas ***

// *** Cálculo da equação do perfil da asa ***

DX0 = 15;
FX0 = 15;
DX1 = (formOpcoes->trkX1->Position/10.);
FX1 = (formOpcoes->trkY1->Position/10.);
DX2 = (formOpcoes->trkX2->Position/10.);
FX2 = (formOpcoes->trkY2->Position/10.);
DX3 = (formOpcoes->trkX3->Position/10.);
FX3 = (formOpcoes->trkY3->Position/10.);
DX4 = (formOpcoes->trkX4->Position/10.);
FX4 = (formOpcoes->trkY4->Position/10.);
DX5 = 32;
FX5 = 15;

FX0X1 = (FX1-FX0)/(DX1-DX0);
FX1X2 = (FX2-FX1)/(DX2-DX1);
FX2X3 = (FX3-FX2)/(DX3-DX2);
FX3X4 = (FX4-FX3)/(DX4-DX3);
FX4X5 = (FX5-FX4)/(DX5-DX4);

FX0X1X2 = (FX1X2-FX0X1)/(DX2-DX0);
FX1X2X3 = (FX2X3-FX1X2)/(DX3-DX1);
FX2X3X4 = (FX3X4-FX2X3)/(DX4-DX2);
```

```
FX3X4X5 = (FX4X5-FX3X4)/(DX5-DX3);

FX0X1X2X3 = (FX1X2X3-FX0X1X2)/(DX3-DX0);
FX1X2X3X4 = (FX2X3X4-FX1X2X3)/(DX4-DX1);
FX2X3X4X5 = (FX3X4X5-FX2X3X4)/(DX5-DX2);

FX0X1X2X3X4 = (FX1X2X3X4-FX0X1X2X3)/(DX4-DX0);
FX1X2X3X4X5 = (FX2X3X4X5-FX1X2X3X4)/(DX5-DX1);

FX0X1X2X3X4X5 = (FX1X2X3X4X5-FX0X1X2X3X4)/(DX5-DX0);

if (X1line <=0 )
      X1line = 1;
if (Y1line <=0 )
      Y1line = 1;
if (Z1line <=0 )
      Z1line = 1;
if (X1line >=a)
      X1line = a-1;
if (Y1line >=b)
      Y1line = b-1;
if (Z1line >=c)
      Z1line = c-1;
if (X2line <=0 )
      X2line = 1;
if (Y2line <=0 )
      Y2line = 1;
if (Z2line <=0 )
      Z2line = 1;
if (X2line >=a)
      X2line = a-1;
if (Y2line >=b)
      Y2line = b-1;
if (Z2line >=c)
      Z2line = c-1;


//*** Resolução e dimensão das linhas de corrente ***

if ( (formOpcoes->edtResolucao->Text == "") )
StreamResolucao = 5;
else
StreamResolucao = (StrToInt(Trim(formOpcoes->edtResolucao->Text))-1);

if ( (formOpcoes->edtStreamRaio->Text == "") )
StreamRaio = 0.3;
else
StreamRaio = StrToFloat(Trim(formOpcoes->edtStreamRaio->Text));

//*** Fim da Resolução e dimensão das linhas de corrente ***
```

```
// ******** Início do cálculo dos pontos  *****

//for (h=1; h<=4; h++)
for (i=1; i<a; i++)
for (j=0; j<=b; j++)
for (k=0; k<=c; k++)
{
	vx[i][j][k] = -(u[i+1][j][k]-u[i][j][k]);
	vy[i][j][k] = -(u[i][j+1][k]-u[i][j][k]);
	vz[i][j][k] = -(u[i][j][k+1]-u[i][j][k]);
	ua = 1;
	ub = 1;
	uc = 1;
	ud = 1;
	ue = 1;
	uf = 1;

 // *** Condições de contorno ***

	if ( u[i+1][j][k]==0 ) {
		ub = 2;
		vx[i][j][k] = 0;
	}
	if ( u[i-1][j][k]==0 ) {
		ua = 2;
		vx[i][j][k] = 0;
	}
	if ( u[i][j+1][k]==0 ) {
		ud = 2;
		vy[i][j][k] = 0;
	}
	if ( u[i][j-1][k]==0) {
		uc = 2;
		vy[i][j][k] = 0;
	}
	if ( u[i][j][k+1]==0 ) {
		uf = 2;
		vz[i][j][k] = 0;
	}
	if ( u[i][j][k-1]==0 ) {
		ue = 2;
		vz[i][j][k] = 0;
	}


//*** Condição geral ***

	u[i][j][k] = (ua*u[i+1][j][k]+ ub*u[i-1][j][k]+ uc*u[i][j+1][k]+ ud*u[i][j-1][k]+
ue*u[i][j][k+1]+ uf*u[i][j][k-1])/6.;
```

```
//** Condições de Barreiras ***

if (tbnCubo->Down){
        if ( (i>=(CuboCentroX-
CuboX/2))&&(i<=(CuboCentroX+CuboX/2))&&(j>=(CuboCentroY-
CuboY/2))&&(j<=(CuboCentroY+CuboY/2))&&(k>=(CuboCentroZ-
CuboZ/2))&&(k<=(CuboCentroZ+CuboZ/2)) ){    // condição da barreira
                u[i][j][k] = 0;
                vx[i][j][k] = 0;
                vy[i][j][k] = 0;
                vz[i][j][k] = 0;
        }
}
if (tbnEsfera->Down){
        Xesfera = (i-EsferaCentroX)*(i-EsferaCentroX)+(j-EsferaCentroY)*(j-
EsferaCentroY)+(k-EsferaCentroZ)*(k-EsferaCentroZ) ;
        Yesfera = raio*raio;
        if ( (Xesfera<=Yesfera) )    // condição da barreira
        {
                u[i][j][k] = 0;
                vx[i][j][k] = 0;
                vy[i][j][k] = 0;
                vz[i][j][k] = 0;
        }
}
if (tbnSemi1->Down){
        XSemi1 = (i-Semi1CentroX)*(i-Semi1CentroX)+(j-Semi1CentroY)*(j-
Semi1CentroY)+(k-Semi1CentroZ)*(k-Semi1CentroZ) ;
        YSemi1 = raio1*raio1;
        if ( (XSemi1<=YSemi1)&&(i>=Semi1CentroX) )    // condição da barreira
        {
                u[i][j][k] = 0;
                vx[i][j][k] = 0;
                vy[i][j][k] = 0;
                vz[i][j][k] = 0;
        }
}
if (tbnSemi2->Down){
        XSemi2 = (i-Semi2CentroX)*(i-Semi2CentroX)+(j-Semi2CentroY)*(j-
Semi2CentroY)+(k-Semi2CentroZ)*(k-Semi2CentroZ);
        YSemi2 = raio2*raio2;
        if ( (XSemi2<=YSemi2)&&(i<=Semi2CentroX) )  {   // condição da barreira

                u[i][j][k] = 0;
                vx[i][j][k] = 0;
                vy[i][j][k] = 0;
                vz[i][j][k] = 0;
        }
}
```

```
if (tbnAsa->Down){
            if ((i>=DX0)&&(i<=DX5)) {
            PX = FX0 +
                FX0X1*(i-DX0) +
                FX0X1X2*(i-DX0)*(i-DX1) +
                FX0X1X2X3*(i-DX0)*(i-DX1)*(i-DX2)+
                FX0X1X2X3X4*(i-DX0)*(i-DX1)*(i-DX2)*(i-DX3) +
                FX0X1X2X3X4X5*(i-DX0)*(i-DX1)*(i-DX2)*(i-DX3)*(i-DX4); //+
               // FX0X1X2X3X4X5X6*(i-DX0)*(i-DX1)*(i-DX2)*(i-DX3)*(i-DX4)*(i-DX5)
+
              //  FX0X1X2X3X4X5X6X7*(i-DX0)*(i-DX1)*(i-DX2)*(i-DX3)*(i-DX4)*(i-
DX5)*(i-DX6);

                    if ((j>=15)&&(j<=PX))
                    if ((k>=13)&&(k<=27)){
                          u[i][j][k] = 0;
                          vx[i][j][k] = 0;
                          vy[i][j][k] = 0;
                          vz[i][j][k] = 0;
                    }
            }
}

// *** Fim das condições de barreiras

Vmaxgrad[i][j][k] = sqrt(vx[i][j][k]*vx[i][j][k]+vy[i][j][k]*vy[i][j][k]+vz[i][j][k]*vz[i][j][k]);
Vmax1= Vmaxgrad[i][j][k];

if (Vmax1>Vmax){
      Vmax = Vmax1;
      posXVmax = i;
      posYVmax = j;
      posZVmax = k;
}
if ((Vmax1<Vmin)&&(Vmax1!=0)) {
      Vmin = Vmax1;
      posXVmin = i;
      posYVmin = j;
      posZVmin = k;
}

   if (abs(u[i][j][k]) > abs(dblValorU)){
      dblValorU = u[i][j][k];
      Ux = i;
      Uy = j;
      Uz = k;
   }
   if (abs(vx[i][j][k]) > abs(dblVx)) {
      dblVx = vx[i][j][k];
      intVxi = i;
```

```
        intVxj = j;
        intVxk = k;
    }
    if (abs(vy[i][j][k]) > abs(dblVy)) {
        dblVy = vy[i][j][k];
        intVyi = i;
        intVyj = j;
        intVyk = k;
    }
    if (abs(vz[i][j][k]) > abs(dblVz)) {
        dblVz = vz[i][j][k];
        intVzi = i;
        intVzj = j;
        intVzk = k;
    }
}

// ----- fim do cálculo --------

// *** Início das Visualizações VTK

if ( (mnuPrincipal->Checked)||(mnuCortes->Checked) ) {

// *** Visualização da Velocidade Máxima
if (mnuVmax->Checked){
  sphereVmax->SetCenter(posXVmax, posYVmax, posZVmax);
}

// *** Visualização da Velocidade Mínima
if (mnuVmin->Checked){
    sphereVmin->SetCenter(posXVmin, posYVmin, posZVmin);
}

//*** Condições de Barreiras

// *** Paralelepípedo
if (tbnCubo->Down){
    cubo->SetXLength(CuboX);
    cubo->SetYLength(CuboY);
    cubo->SetZLength(CuboZ);
    cubo->SetCenter(CuboCentroX,CuboCentroY,CuboCentroZ);
}
// *** Fim do paralelepípedo

// *** Esfera
if (tbnEsfera->Down){
    sphere->SetRadius(raio-0.35);
    sphere->SetCenter(EsferaCentroX, EsferaCentroY, EsferaCentroZ);
}
// *** Fim da esfera
```

```
// *** Semi-esfera 1
if (tbnSemi1->Down){
   sphere1->SetRadius(raio1-0.35);
   sphere1->SetCenter(Semi1CentroX, Semi1CentroY, Semi1CentroZ);
   plane1->SetOrigin(Semi1CentroX, Semi1CentroY, Semi1CentroZ);
}
// *** Fim da semi-esfera 1

// *** Semi-esfera 2
if (tbnSemi2->Down){
   sphere2->SetRadius(raio2-0.35);
   sphere2->SetCenter(Semi2CentroX, Semi2CentroY, Semi2CentroZ);
   plane2->SetOrigin(Semi2CentroX, Semi2CentroY, Semi2CentroZ);
}
// *** Fim da semi-esfera 2

//*** Início da Asa ***
if (tbnAsa->Down) {

for (i=1; i<=15; i++) {

PX = FX0 +
    FX0X1*(i+15-DX0) +
    FX0X1X2*(i+15-DX0)*(i+15-DX1) +
    FX0X1X2X3*(i+15-DX0)*(i+15-DX1)*(i+15-DX2)+
    FX0X1X2X3X4*(i+15-DX0)*(i+15-DX1)*(i+15-DX2)*(i+15-DX3) +
    FX0X1X2X3X4X5*(i+15-DX0)*(i+15-DX1)*(i+15-DX2)*(i+15-DX3)*(i+15-DX4);

if (PX<15) PX = 15;

pontos->InsertPoint (i, i+15, PX, 13.0);
}
pontos->Modified();
}
//*** Fim da Asa ***

//*** Fim das condições de barreira


//*** Início da visualização do domínio

for (k=0; k<=c; k++)
for (j=0; j<=b; j++)
for (i=0; i<=a; i++) {
   if (formOpcoes->mnuVtotal->Checked)
     array->SetValue(k*(a+1)*(b+1)+j*(a+1)+i, Vmaxgrad[i][j][k]);

   if (formOpcoes->mnuVx->Checked)
     array->SetValue(k*(a+1)*(b+1)+j*(a+1)+i, vx[i][j][k]);
```

```
    if (formOpcoes->mnuVy->Checked)
      array->SetValue(k*(a+1)*(b+1)+j*(a+1)+i, vy[i][j][k]);

    if (formOpcoes->mnuVz->Checked)
      array->SetValue(k*(a+1)*(b+1)+j*(a+1)+i, vz[i][j][k]);

      vetores->InsertTuple3(k*(a+1)*(b+1)+j*(a+1)+i, vx[i][j][k], vy[i][j][k], vz[i][j][k]);

  }

array->Modified();
vetores->Modified();
rgrid->GetPointData()->SetVectors(vetores);
rgrid->GetPointData()->SetScalars(array);
rgrid->Modified();

// ***** Escala de cores

if (mnuEscalaCores->Checked) {

Table->SetRange(rgrid->GetScalarRange());

}

 // ***** Glifos

/*vtkPolyData *glifodados = vtkPolyData::New();
  glifodados->SetPoints(pontos);

vtkConeSource *balls =  vtkConeSource::New();
  balls->SetRadius(0.08);
  balls->SetResolution(5);

vtkTransform *transform = vtkTransform::New();
    transform->Translate(0.5, 0.0, 0.0);

vtkTransformPolyDataFilter *transformF = vtkTransformPolyDataFilter::New();
    transformF->SetInputConnection(balls->GetOutputPort());
    transformF->SetTransform(transform);

vtkGlyph3D *glyphPoints = vtkGlyph3D::New();
  glyphPoints->SetInput(rgrid);
  glyphPoints->SetSource(transformF->GetOutput());
  glyphPoints->SetVectorModeToUseVector();
  glyphPoints->SetScaleFactor(0.03);

vtkPolyDataMapper *glyphMapper = vtkPolyDataMapper::New();
  glyphMapper->SetInputConnection(glyphPoints->GetOutputPort());
```

```
vtkActor *glyph = vtkActor::New();
  glyph->SetMapper(glyphMapper);

if ( formVisPrincipal->cbxGlifos->Checked )
formVisPrincipal->VTK1->GetRenderer()->AddActor(glyph);

glifodados->Delete();
balls->Delete();
glyphPoints->Delete();
glyphMapper->Delete();
glyph->Delete();
transform->Delete();
transformF->Delete();

*/
//eval   [glyph GetProperty] SetDiffuseColor $tomato
//  glyph-> GetProperty] SetSpecular .3
//  [glyph GetProperty] SetSpecularPower 30

// **** Fim dos glifos

// *** Linhas de fluxo

if (formVisPrincipal->cbxStreamTubos->Checked) {
      formVisPrincipal->streamTube->SetRadius(Principal->StreamRaio);
      formVisPrincipal->streamTube->SetNumberOfSides(6);
      formVisPrincipal->streamTube->SetVaryRadiusToVaryRadiusOff();
}

if ((formVisPrincipal->cbxStreamTubos->Checked)||(formVisPrincipal->cbxStreamFitas-
>Checked)) {

      if (formOpcoes->cbxRaio->Checked){
            formVisPrincipal->streamTube->SetRadius(0.2);
            formVisPrincipal->streamTube->SetVaryRadiusToVaryRadiusByScalar();
      }

      if (formOpcoes->rdbStreamLinhaOrigem->Checked) {
            formVisPrincipal->rakeLine->SetPoint1(X1line, Y1line, Z1line);
            formVisPrincipal->rakeLine->SetPoint2(X2line, Y2line, Z2line);
            formVisPrincipal->rakeLine->SetResolution(StreamResolucao);
            //formVisPrincipal->rakeLine->Modified();

            formVisPrincipal->sl->SetSource(formVisPrincipal->rakeLine->GetOutput());
      }

      if ( formOpcoes->rdbStreamPlanoOrigem->Checked ) {
            formVisPrincipal->rakePlane->SetXResolution(StreamResolucao);
            formVisPrincipal->rakePlane->SetYResolution(StreamResolucao);
```

```
            formVisPrincipal->rakePlane->SetOrigin(StreamPlanoOrigemX,
StreamPlanoOrigemY, StreamPlanoOrigemZ);
            formVisPrincipal->rakePlane->SetPoint1(X1line, Y1line, Z1line);
            formVisPrincipal->rakePlane->SetPoint2(X2line, StreamPlanoOrigemY, Z2line);
            //formVisPrincipal->rakePlane->Modified();

            formVisPrincipal->sl->SetSource(formVisPrincipal->rakePlane->GetOutput());

      }
      formVisPrincipal->sl->Modified();

}

 /*

// *** Stream Pontos

if (formVisPrincipal->cbxPontos->Checked ){

vtkStreamPoints *strPoints = vtkStreamPoints::New();
   //strPoints->SetInput(rgrid);
   //strPoints->SetIntegrator(rk4);
   strPoints->SetStartPosition( 4, 20, 20);
   strPoints->SetMaximumPropagationTime(200);
   strPoints->SetTimeIncrement(0.5);
   strPoints->SetIntegrationDirectionToIntegrateBothDirections();
   strPoints->VorticityOn();
   //strPoints->Update();

vtkConeSource *strCone = vtkConeSource::New();
   strCone->SetResolution(6);
   strCone->SetRadius(0.5);

vtkGlyph3D *strCones = vtkGlyph3D::New();
   strCones->SetInputConnection(strPoints->GetOutputPort());
   strCones->SetSource(strCone->GetOutput());
   //strCones->SetScaleFactor(0.5);
   //strCones->SetScaleModeToScaleByVector();
   strCones->SetScaleModeToDataScalingOff();

vtkPolyDataMapper *strmapCones = vtkPolyDataMapper::New();
   strmapCones->SetInputConnection(strCones->GetOutputPort());
   //strmapCones->SetScalarRange(rgrid->GetScalarRange());
   strmapCones->GlobalImmediateModeRenderingOn();

vtkActor *strConesActor = vtkActor::New();
   strConesActor->SetMapper(strmapCones);

if (mnuPrincipal->Checked)
   formVisPrincipal->VTK1->GetRenderer()->AddActor(strConesActor);
```

```
strPoints->Delete();
strCone->Delete();
strCones->Delete();
strmapCones->Delete();
strConesActor->Delete();

}

 */

// *** Fim da stream line


//*** Planos de corte

if (formCortes->rdbGraXY->Checked){
      formCortes->planoXY->SetExtent( 0,a, 0, b, Corte, Corte);
}
if (formCortes->rdbIsoXY->Checked){
      formCortes->planoCorteXY->SetOrigin (0, 0, Corte);
}
if (formCortes->rdbGraXZ->Checked){
      formCortes->planoXZ->SetExtent ( 0,a, Corte, Corte, 0,c);
}
if (formCortes->rdbIsoXZ->Checked){
      formCortes->planoCorteXZ->SetOrigin (0, Corte,0);
}
if (formCortes->rdbGraYZ->Checked){
      formCortes->planoYZ->SetExtent (Corte, Corte, 0,b, 0,c);
}
if (formCortes->rdbIsoYZ->Checked){
      formCortes->planoCorteYZ->SetOrigin (Corte, 0, 0);
}

//*** Fim dos planos de corte

}

formVisPrincipal->VTK1->Invalidate();
formCortes->VTK2->Invalidate();

// *** Fim da visulalização VTK

// *** Contador de Frames por segundo ***

ContadorFrame += 1;
time (&fim);

difTempo = difftime(fim,inicio);
```

```
if (difTempo >= 1) {
      time (&inicio);
      ContadorFPS = ContadorFrame/difTempo;
      lblFPS->Caption = Format("%2.2f", ARRAYOFCONST((ContadorFPS)));
      ContadorFrame = 0;
}

// *** Fim da contagem de FPS ***

formDadosSaida->lblVx->Caption = Format("%2.6f", ARRAYOFCONST((dblVx)));
formDadosSaida->lblVy->Caption = Format("%2.6f", ARRAYOFCONST((dblVy)));
formDadosSaida->lblVz->Caption = Format("%2.6f", ARRAYOFCONST((dblVz)));
formDadosSaida->lblVmax->Caption = Format("%2.5f", ARRAYOFCONST((Vmax)));
formDadosSaida->lblVmin->Caption = Format("%2.5f", ARRAYOFCONST((Vmin)));
formDadosSaida->lblVxi->Caption = intVxi;
formDadosSaida->lblVxj->Caption = intVxj;
formDadosSaida->lblVxk->Caption = intVxk;
formDadosSaida->lblVyi->Caption = intVyi;
formDadosSaida->lblVyj->Caption = intVyj;
formDadosSaida->lblVyk->Caption = intVyk;
formDadosSaida->lblVzi->Caption = intVzi;
formDadosSaida->lblVzj->Caption = intVzj;
formDadosSaida->lblVzk->Caption = intVzk;
formDadosSaida->lblVmaxX->Caption = posXVmax;
formDadosSaida->lblVmaxY->Caption = posYVmax;
formDadosSaida->lblVmaxZ->Caption = posZVmax;
formDadosSaida->lblVminX->Caption = posXVmin;
formDadosSaida->lblVminY->Caption = posYVmin;
formDadosSaida->lblVminZ->Caption = posZVmin;

formDadosSaida->lblX1->Caption = Format("%2.4f",
ARRAYOFCONST((vx[(a/2)][(b/2)][(c/2)])));
formDadosSaida->lblY1->Caption = Format("%2.4f",
ARRAYOFCONST((vy[(a/2)][(b/2)][(c/2)])));
formDadosSaida->lblZ1->Caption = Format("%2.4f",
ARRAYOFCONST((vz[(a/2)][(b/2)][(c/2)])));

formDadosSaida->lblPotInicio->Caption = Format("%2.2f",
ARRAYOFCONST((u[0][(b/2)][(c/2)])));
formDadosSaida->lblPotMeio->Caption = Format("%2.2f",
ARRAYOFCONST((u[(a/2)][(b/2)][(c/2)])));
formDadosSaida->lblPotFim->Caption = Format("%2.2f", ARRAYOFCONST((u[(a-
1)][(b/2)][(c/2)])));

formCortes->lblCorte->Caption = Corte*2.5;

}

}
```

```
//---------------------------------------------------------------------------

void __fastcall TPrincipal::opcSobreClick(TObject *Sender)
{
Application->CreateForm(__classid(TformSobre),&formSobre);
formSobre->ShowModal();
formSobre->Free();
}
//---------------------------------------------------------------------------

 void __fastcall TPrincipal::FormClose(TObject *Sender,  TCloseAction &Action)
{
  formVisPrincipal->VTK1->GetRenderer()->Clear();
  formVisPrincipal->VTK1->GetRenderer()->RemoveAllViewProps();
  delete formVisPrincipal->VTK1;
  formVisPrincipal->Hide();

  formCortes->VTK2->GetRenderer()->Clear();
  formCortes->VTK2->GetRenderer()->RemoveAllViewProps();
  delete formCortes->VTK2;
  formCortes->Hide();

  formDadosSaida->Hide();
  formOpcoes->Hide();

}
//---------------------------------------------------------------------------

void __fastcall TPrincipal::CordeFundo1Click(TObject *Sender)
{
if (!CordeFundo->Execute())
    {
    return;
    }
  DWORD  L = ColorToRGB(CordeFundo->Color);
  float r = GetRValue(L)/255.0;
  float g = GetGValue(L)/255.0;
  float b = GetBValue(L)/255.0;
  if (mnuPrincipal->Checked) {
    formVisPrincipal->VTK1->GetRenderer()->SetBackground(r, g, b);
    formVisPrincipal->VTK1->Invalidate();
  }

}
//---------------------------------------------------------------------------

void __fastcall TPrincipal::mnuCortesClick(TObject *Sender)
{
if  (mnuCortes->Checked) {
    formCortes->Show();
```

```
   tbnCortes->Down = true;
}

else {
  formCortes->VTK2->GetRenderer()->Clear();
  formCortes->VTK2->GetRenderer()->RemoveAllViewProps();
  formCortes->Hide();
  tbnCortes->Down = false;
}

}
//---------------------------------------------------------------------------

void __fastcall TPrincipal::opcSairClick(TObject *Sender)
{

  formVisPrincipal->VTK1->GetRenderer()->Clear();
  formVisPrincipal->VTK1->GetRenderer()->RemoveAllViewProps();
  delete formVisPrincipal->VTK1;
  formVisPrincipal->Hide();

  formCortes->VTK2->GetRenderer()->Clear();
  formCortes->VTK2->GetRenderer()->RemoveAllViewProps();
  delete formCortes->VTK2;
  formCortes->Hide();

  formDadosSaida->Hide();
  formOpcoes->Hide();

Application->Terminate();
}
//---------------------------------------------------------------------------

void __fastcall TPrincipal::opcAbrirClick(TObject *Sender)
{
if (dlgAbrir->Execute())
    {
    Show();
    //   Arquivo->LoadFromFile(dlgAbrir->FileName);
   }
}
//---------------------------------------------------------------------------

void __fastcall TPrincipal::mnuPrincipalClick(TObject *Sender)
{

if  (mnuPrincipal->Checked) {
    formVisPrincipal->Show();
    tbnPrincipal->Down = true;
}
```

```
else {
  //formVisPrincipal->VTK1->GetRenderer()->Clear();
  //formVisPrincipal->VTK1->GetRenderer()->RemoveAllViewProps();
  formVisPrincipal->Hide();
  tbnPrincipal->Down = false;
}
}
//---------------------------------------------------------------------------

void __fastcall TPrincipal::mnuDadosSaidaClick(TObject *Sender)
{
if  (mnuDadosSaida->Checked) {
    formDadosSaida->Show();
    tbnDadosSaida->Down = true;
}

else {
    formDadosSaida->Hide();
    tbnDadosSaida->Down = false;
}
}
//---------------------------------------------------------------------------

void __fastcall TPrincipal::mnuOpcoes1Click(TObject *Sender)
{
if  (mnuOpcoes1->Checked) {
    formOpcoes->Show();
    tbnOpcoes->Down = true;
}

else {
    formOpcoes->Hide();
    tbnOpcoes->Down = false;
}

}
//---------------------------------------------------------------------------

void __fastcall TPrincipal::btnPararClick(TObject *Sender)
{
Parar = 1;
}
//---------------------------------------------------------------------------

 void __fastcall TPrincipal::btnContinuarClick(TObject *Sender)

{
Parar = 0;
}
```

```
//---------------------------------------------------------------------------

void __fastcall TPrincipal::tbnCuboClick(TObject *Sender)
{
cubo = vtkCubeSource::New();
   cubo->SetXLength(CuboX);
   cubo->SetYLength(CuboY);
   cubo->SetZLength(CuboZ);
   cubo->SetCenter(CuboCentroX,CuboCentroY,CuboCentroZ);

  vtkPolyDataMapper *cuboMapper = vtkPolyDataMapper::New();
   cuboMapper->SetInput(cubo->GetOutput());
   cuboMapper->GlobalImmediateModeRenderingOn();

static  vtkActor *cubo1 = vtkActor::New();
   cubo1->SetMapper(cuboMapper);
   cubo1->GetProperty()->SetColor(0.52, 0.27, 0);
   cubo1->GetProperty()->SetAmbient(0.2);
   cubo1->GetProperty()->SetDiffuse(0.3);
   cubo1->GetProperty()->SetSpecular(0.8);

if  (tbnCubo->Down) {
   formVisPrincipal->VTK1->GetRenderer()->AddActor(cubo1);
   formCortes->VTK2->GetRenderer()->AddActor(cubo1);
}

else {
   formVisPrincipal->VTK1->GetRenderer()->RemoveViewProp(cubo1);
   formCortes->VTK2->GetRenderer()->RemoveViewProp(cubo1);
}

  cubo->Delete();
  cuboMapper->Delete();
  //cubo1->Delete();
  //formVisPrincipal->VTK1->GetRenderer()->ResetCamera();
  //formCortes->VTK2->GetRenderer()->ResetCamera();
  //formVisPrincipal->VTK1->Invalidate();
  //formCortes->VTK2->Invalidate();

}
//---------------------------------------------------------------------------

void __fastcall TPrincipal::tbnEsferaClick(TObject *Sender)
{
// *** Esfera

sphere = vtkSphereSource::New();
   sphere->SetThetaResolution(30);
   sphere->SetPhiResolution(15);
```

```
   sphere->SetRadius(raio-0.2);
   sphere->SetCenter(EsferaCentroX, EsferaCentroY, EsferaCentroZ);

vtkPolyDataMapper *sphereMapper = vtkPolyDataMapper::New();
   sphereMapper->SetInput(sphere->GetOutput());
   sphereMapper->GlobalImmediateModeRenderingOn();

static vtkActor *sphereActor = vtkActor::New();
   sphereActor->SetMapper(sphereMapper);
   sphereActor->GetProperty()->SetColor(0.12, 0.27, 0.57);
   sphereActor->GetProperty()->SetAmbient(0.2);
   sphereActor->GetProperty()->SetDiffuse(0.3);
   sphereActor->GetProperty()->SetSpecular(0.8);

  sphere->Delete();
  sphereMapper->Delete();
  //sphereActor->Delete();

// *** Fim da esfera


if  (tbnEsfera->Down) {
   formVisPrincipal->VTK1->GetRenderer()->AddActor(sphereActor);
   formCortes->VTK2->GetRenderer()->AddActor(sphereActor);
}

else {
   formVisPrincipal->VTK1->GetRenderer()->RemoveViewProp(sphereActor);
   formCortes->VTK2->GetRenderer()->RemoveViewProp(sphereActor);
}

//formVisPrincipal->VTK1->GetRenderer()->ResetCamera();
//formCortes->VTK2->GetRenderer()->ResetCamera();

}
//---------------------------------------------------------------------------

void __fastcall TPrincipal::tbnSemi1Click(TObject *Sender)
{

// *** Semi-esfera 1

sphere1 = vtkSphereSource::New();
   sphere1->SetRadius(raio1-0.2);
   sphere1->SetPhiResolution(15);
   sphere1->SetThetaResolution(30);
   sphere1->SetCenter(Semi1CentroX, Semi1CentroY, Semi1CentroZ);

plane1 = vtkPlane::New();
   plane1->SetOrigin(Semi1CentroX, Semi1CentroY, Semi1CentroZ);
```

```
    plane1->SetNormal( 1, 0, 0);

vtkClipPolyData  *clipper1 = vtkClipPolyData::New();
    clipper1->SetInput(sphere1->GetOutput());
    clipper1->SetClipFunction(plane1);
    clipper1->GenerateClipScalarsOn();
    clipper1->GenerateClippedOutputOn();
    clipper1->SetValue(0);

vtkPolyDataMapper *clipMapper1 =vtkPolyDataMapper::New();
    clipMapper1->SetInput(clipper1->GetOutput());
    clipMapper1->ScalarVisibilityOff();
    clipMapper1->GlobalImmediateModeRenderingOn();

vtkProperty *backProp1 = vtkProperty::New();
    backProp1->SetDiffuseColor( 1, 0, 0.5 );

static vtkActor *semiActor1 = vtkActor::New();
    semiActor1->SetMapper(clipMapper1);
    semiActor1->GetProperty()->SetColor( 0.2, 0.8, 0.1);
    semiActor1->SetBackfaceProperty(backProp1);

// now extract feature edges
vtkFeatureEdges *boundaryEdges1 =  vtkFeatureEdges::New();
  boundaryEdges1->SetInput(clipper1->GetOutput());
  boundaryEdges1->BoundaryEdgesOn();
  boundaryEdges1->FeatureEdgesOff();
  boundaryEdges1->NonManifoldEdgesOff();

vtkCleanPolyData *boundaryClean1 = vtkCleanPolyData::New();
  boundaryClean1->SetInput(boundaryEdges1->GetOutput());

vtkStripper *boundaryStrips1 = vtkStripper::New();
  boundaryStrips1->SetInput(boundaryClean1->GetOutput());
  boundaryStrips1->Update();

vtkPolyData *boundaryPoly1 = vtkPolyData::New();
  boundaryPoly1->SetPoints( (boundaryStrips1->GetOutput())->GetPoints());
  boundaryPoly1->SetPolys( (boundaryStrips1->GetOutput())->GetLines());

vtkTriangleFilter *boundaryTriangles1 = vtkTriangleFilter::New();
  boundaryTriangles1->SetInput(boundaryPoly1);

vtkPolyDataMapper *boundaryMapper1 = vtkPolyDataMapper::New();
  boundaryMapper1->SetInput(boundaryPoly1);
  boundaryMapper1->GlobalImmediateModeRenderingOn();

static vtkActor *boundaryActor1 = vtkActor::New();
  boundaryActor1->SetMapper(boundaryMapper1);
  boundaryActor1->GetProperty()->SetColor(0.7, 0.8, 0.2);
```

```
//semiActor1->Delete();
//boundaryActor1->Delete();
boundaryMapper1->Delete();
boundaryTriangles1->Delete();
boundaryPoly1->Delete();
boundaryStrips1->Delete();
boundaryClean1->Delete();
boundaryEdges1->Delete();
backProp1->Delete();
clipper1->Delete();
plane1->Delete();
sphere1->Delete();
clipMapper1->Delete();

// *** Fim da semi-esfera 1

if (tbnSemi1->Down) {
   formVisPrincipal->VTK1->GetRenderer()->AddActor(semiActor1);
   formVisPrincipal->VTK1->GetRenderer()->AddActor(boundaryActor1);
   formCortes->VTK2->GetRenderer()->AddActor(semiActor1);
   formCortes->VTK2->GetRenderer()->AddActor(boundaryActor1);
}

else {
   formVisPrincipal->VTK1->GetRenderer()->RemoveViewProp(semiActor1);
   formVisPrincipal->VTK1->GetRenderer()->RemoveViewProp(boundaryActor1);
   formCortes->VTK2->GetRenderer()->RemoveViewProp(semiActor1);
   formCortes->VTK2->GetRenderer()->RemoveViewProp(boundaryActor1);
}

//formVisPrincipal->VTK1->GetRenderer()->ResetCamera();
//formCortes->VTK2->GetRenderer()->ResetCamera();

}
//---------------------------------------------------------------------------

void __fastcall TPrincipal::tbnSemi2Click(TObject *Sender)
{
//*** Semi-esfera 2

sphere2 = vtkSphereSource::New();
   sphere2->SetRadius(raio2-0.2);
   sphere2->SetPhiResolution(15);
   sphere2->SetThetaResolution(30);
   sphere2->SetCenter(Semi2CentroX, Semi2CentroY, Semi2CentroZ);

plane2 = vtkPlane::New();
   plane2->SetOrigin(Semi2CentroX, Semi2CentroY, Semi2CentroZ);
```

```
    plane2->SetNormal( -1, 0, 0);

vtkClipPolyData  *clipper2 = vtkClipPolyData::New();
    clipper2->SetInput(sphere2->GetOutput());
    clipper2->SetClipFunction(plane2);
    clipper2->GenerateClipScalarsOn();
    clipper2->GenerateClippedOutputOn();
    clipper2->SetValue(0);
vtkPolyDataMapper *clipMapper2 =vtkPolyDataMapper::New();
    clipMapper2->SetInput(clipper2->GetOutput());
    clipMapper2->ScalarVisibilityOff();
    clipMapper2->GlobalImmediateModeRenderingOn();

vtkProperty *backProp2 = vtkProperty::New();
    backProp2->SetDiffuseColor( 1, 0, 0.5 );

static vtkActor *semiActor2 = vtkActor::New();
    semiActor2->SetMapper(clipMapper2);
    semiActor2->GetProperty()->SetColor( 0.2, 0.8, 0.1);
    semiActor2->SetBackfaceProperty(backProp2);

// now extract feature edges
vtkFeatureEdges *boundaryEdges2 =  vtkFeatureEdges::New();
  boundaryEdges2->SetInput(clipper2->GetOutput());
  boundaryEdges2->BoundaryEdgesOn();
  boundaryEdges2->FeatureEdgesOff();
  boundaryEdges2->NonManifoldEdgesOff();

vtkCleanPolyData *boundaryClean2 = vtkCleanPolyData::New();
  boundaryClean2->SetInput(boundaryEdges2->GetOutput());

vtkStripper *boundaryStrips2 = vtkStripper::New();
  boundaryStrips2->SetInput(boundaryClean2->GetOutput());
  boundaryStrips2->Update();

vtkPolyData *boundaryPoly2 = vtkPolyData::New();
  boundaryPoly2->SetPoints( (boundaryStrips2->GetOutput())->GetPoints());
  boundaryPoly2->SetPolys( (boundaryStrips2->GetOutput())->GetLines());

vtkTriangleFilter *boundaryTriangles2 = vtkTriangleFilter::New();
  boundaryTriangles2->SetInput(boundaryPoly2);

vtkPolyDataMapper *boundaryMapper2 = vtkPolyDataMapper::New();
  boundaryMapper2->SetInput(boundaryPoly2);
  boundaryMapper2->GlobalImmediateModeRenderingOn();

static vtkActor *boundaryActor2 = vtkActor::New();
  boundaryActor2->SetMapper(boundaryMapper2);
  boundaryActor2->GetProperty()->SetColor(0.7, 0.8, 0.2);
```

```
if (tbnSemi2->Down){
        formVisPrincipal->VTK1->GetRenderer()->AddActor(semiActor2);
        formVisPrincipal->VTK1->GetRenderer()->AddActor(boundaryActor2);
        formCortes->VTK2->GetRenderer()->AddActor(semiActor2);
        formCortes->VTK2->GetRenderer()->AddActor(boundaryActor2);
}

else {
        formVisPrincipal->VTK1->GetRenderer()->RemoveViewProp(semiActor2);
        formVisPrincipal->VTK1->GetRenderer()->RemoveViewProp(boundaryActor2);
        formCortes->VTK2->GetRenderer()->RemoveViewProp(semiActor2);
        formCortes->VTK2->GetRenderer()->RemoveViewProp(boundaryActor2);
}

//semiActor2->Delete();
//boundaryActor2->Delete();
boundaryMapper2->Delete();
boundaryTriangles2->Delete();
boundaryPoly2->Delete();
boundaryStrips2->Delete();
boundaryClean2->Delete();
boundaryEdges2->Delete();
backProp2->Delete();
clipper2->Delete();
plane2->Delete();
sphere2->Delete();
clipMapper2->Delete();

// *** Fim da semi-esfera 2

//formVisPrincipal->VTK1->GetRenderer()->ResetCamera();
//formCortes->VTK2->GetRenderer()->ResetCamera();

}
//---------------------------------------------------------------------------

void __fastcall TPrincipal::tbnPrincipalClick(TObject *Sender)
{
if  (tbnPrincipal->Down) {
   mnuPrincipal->Checked = true;
   formVisPrincipal->Show();
   //formVisPrincipal->VTK1->Invalidate();
}

else {
   mnuPrincipal->Checked = false;
  formVisPrincipal->Hide();
}

}
```

```
//---------------------------------------------------------------------------

void __fastcall TPrincipal::tbnCortesClick(TObject *Sender)
{

if  (tbnCortes->Down) {
   mnuCortes->Checked = true;
   formCortes->Show();
   formCortes->VTK2->Invalidate();
}

else {
   mnuCortes->Checked = false;
   formCortes->Hide();
}

}
//---------------------------------------------------------------------------

void __fastcall TPrincipal::tbnOpcoesClick(TObject *Sender)
{

if  (tbnOpcoes->Down) {
   mnuOpcoes1->Checked = true;
   formOpcoes->Show();
}

else {
   mnuOpcoes1->Checked = false;
   formOpcoes->Hide();
}

}
//---------------------------------------------------------------------------

void __fastcall TPrincipal::tbnDadosSaidaClick(TObject *Sender)
{

if  (tbnDadosSaida->Down) {
   mnuDadosSaida->Checked = true;
   formDadosSaida->Show();
}

else {
   mnuDadosSaida->Checked = false;
   formDadosSaida->Hide();
}

}
//---------------------------------------------------------------------------
```

```
void __fastcall TPrincipal::mnuEscalaCoresClick(TObject *Sender)
{
// *** Escala de cores

vtkProperty2D *propriedades2D = vtkProperty2D::New();
vtkTextProperty *propriedadesTexto = vtkTextProperty::New();
static vtkScalarBarActor *barra = vtkScalarBarActor::New();

  propriedades2D->SetColor(1, 0, 0);
  propriedades2D->SetLineWidth(1);

  propriedadesTexto->SetFontFamilyToTimes();
  propriedadesTexto->SetColor(0.1, 0.25, 0.25);
  propriedadesTexto->SetFontSize(1.2);
  propriedadesTexto->SetShadow(2);

Table = vtkLookupTable::New();
  //Table->SetNumberOfColors(150);
  //Table->SetHueRange(0, 0.667);
  Table->SetRange(rgrid->GetScalarRange());
  //Table->SetRampToLinear();
  Table->Build();

  barra->SetLookupTable(Table);
  barra->SetProperty(propriedades2D);
  barra->SetTitle("Velocidade");
  barra->SetTitleTextProperty(propriedadesTexto);
  barra->GetPositionCoordinate()->SetCoordinateSystemToNormalizedViewport();
  barra->GetPositionCoordinate()->SetValue(0.1, 0.01);
  barra->SetWidth(0.85);
  barra->SetHeight(0.1);
  barra->SetLabelTextProperty(propriedadesTexto);
  barra->SetOrientationToHorizontal();

if (mnuEscalaCores->Checked) {
  formVisPrincipal->VTK1->GetRenderer()->AddActor(barra);
  formCortes->VTK2->GetRenderer()->AddActor(barra);
}
else {
  formVisPrincipal->VTK1->GetRenderer()->RemoveViewProp(barra);
  formCortes->VTK2->GetRenderer()->RemoveViewProp(barra);
}

barra->Delete();
propriedades2D->Delete();
propriedadesTexto->Delete();
Table->Delete();

formVisPrincipal->VTK1->Invalidate();
```

```
formCortes->VTK2->Invalidate();

// *** Fim da escala de cores

}
//---------------------------------------------------------------------------

void __fastcall TPrincipal::mnuVmaxClick(TObject *Sender)
{
sphereVmax = vtkSphereSource::New();
   sphereVmax->SetThetaResolution(15);
   sphereVmax->SetPhiResolution(15);
   sphereVmax->SetRadius(0.5);
   sphereVmax->SetCenter(posXVmax, posYVmax, posZVmax);

  vtkPolyDataMapper *sphereMapperVmax = vtkPolyDataMapper::New();
   sphereMapperVmax->SetInput(sphereVmax->GetOutput());
   sphereMapperVmax->ImmediateModeRenderingOn();

static  vtkActor *sphereActorVmax = vtkActor::New();
   sphereActorVmax->SetMapper(sphereMapperVmax);
   sphereActorVmax->GetProperty()->SetColor(0.9, 0.6, 0.1);
   sphereActorVmax->GetProperty()->SetAmbient(0.2);
   sphereActorVmax->GetProperty()->SetDiffuse(0.3);
   sphereActorVmax->GetProperty()->SetSpecular(0.8);

if (mnuVmax->Checked){
  formVisPrincipal->VTK1->GetRenderer()->AddActor(sphereActorVmax);
  formCortes->VTK2->GetRenderer()->AddActor(sphereActorVmax);
}
else {
  formVisPrincipal->VTK1->GetRenderer()->RemoveViewProp(sphereActorVmax);
  formCortes->VTK2->GetRenderer()->RemoveViewProp(sphereActorVmax);
}

  sphereVmax->Delete();
  sphereMapperVmax->Delete();
  //sphereActorVmax->Delete();
}
//---------------------------------------------------------------------------

void __fastcall TPrincipal::mnuVminClick(TObject *Sender)
{
// *** Visualização da Velocidade Mínima

 sphereVmin = vtkSphereSource::New();
   sphereVmin->SetThetaResolution(15);
   sphereVmin->SetPhiResolution(15);
   sphereVmin->SetRadius(0.5);
   sphereVmin->SetCenter(posXVmin, posYVmin, posZVmin);
```

```
  vtkPolyDataMapper *sphereMapperVmin = vtkPolyDataMapper::New();
    sphereMapperVmin->SetInput(sphereVmin->GetOutput());
    sphereMapperVmin->GlobalImmediateModeRenderingOn();

static vtkActor *sphereActorVmin = vtkActor::New();
    sphereActorVmin->SetMapper(sphereMapperVmin);
    sphereActorVmin->GetProperty()->SetColor(0.1, 0.6, 0.9);
    sphereActorVmin->GetProperty()->SetAmbient(0.2);
    sphereActorVmin->GetProperty()->SetDiffuse(0.3);
    sphereActorVmin->GetProperty()->SetSpecular(0.8);

if (mnuVmin->Checked){
    formVisPrincipal->VTK1->GetRenderer()->AddActor(sphereActorVmin);
    formCortes->VTK2->GetRenderer()->AddActor(sphereActorVmin);
}
else {
    formVisPrincipal->VTK1->GetRenderer()->AddActor(sphereActorVmin);
    formCortes->VTK2->GetRenderer()->AddActor(sphereActorVmin);
}

  sphereVmin->Delete();
  sphereMapperVmin->Delete();
//  sphereActorVmin->Delete();

}
//---------------------------------------------------------------------------

void __fastcall TPrincipal::tbnAsaClick(TObject *Sender)
{
pontos = vtkPoints::New();

poly = vtkCellArray::New();
poly->InsertNextCell (17);//number of points

for (i=0; i<=16; i++) {

PX = FX0 +
     FX0X1*(i+15-DX0) +
     FX0X1X2*(i+15-DX0)*(i+15-DX1) +
     FX0X1X2X3*(i+15-DX0)*(i+15-DX1)*(i+15-DX2)+
     FX0X1X2X3X4*(i+15-DX0)*(i+15-DX1)*(i+15-DX2)*(i+15-DX3) +
     FX0X1X2X3X4X5*(i+15-DX0)*(i+15-DX1)*(i+15-DX2)*(i+15-DX3)*(i+15-DX4);

if (PX<15) PX = 15;

pontos->InsertPoint (i, i+15, PX, 13.0);
poly->InsertCellPoint (i);

}
```

```
profile = vtkPolyData::New();
profile->SetPoints(pontos);
profile->SetPolys(poly);

vtkLinearExtrusionFilter *extrude = vtkLinearExtrusionFilter::New();
extrude->SetInput(profile);
extrude->SetScaleFactor(14);
extrude->SetExtrusionTypeToVectorExtrusion();
extrude->SetVector(0, 0, 1);

vtkPolyDataNormals *normals = vtkPolyDataNormals::New();
normals->SetInput(extrude->GetOutput());
normals->SetFeatureAngle(6);

vtkPolyDataMapper *cylinderMapper = vtkPolyDataMapper::New();
cylinderMapper->SetInput(normals->GetOutput());

static vtkActor *AsaActor = vtkActor::New();
AsaActor->SetMapper(cylinderMapper);
AsaActor->GetProperty()->SetColor(0.9, 0.6, 0.3);
AsaActor->GetProperty()->SetAmbient(0.2);
AsaActor->GetProperty()->SetDiffuse(0.3);
AsaActor->GetProperty()->SetSpecular(0.8);

if (tbnAsa->Down){
        formVisPrincipal->VTK1->GetRenderer()->AddActor(AsaActor);
        formCortes->VTK2->GetRenderer()->AddActor(AsaActor);
}

else {
        formVisPrincipal->VTK1->GetRenderer()->RemoveViewProp(AsaActor);
        formCortes->VTK2->GetRenderer()->RemoveViewProp(AsaActor);
}

pontos->Delete();
profile->Delete();
extrude->Delete();
normals->Delete();

//formVisPrincipal->VTK1->GetRenderer()->ResetCamera();
//formCortes->VTK2->GetRenderer()->ResetCamera();

}
//---------------------------------------------------------------------------
```

```
//---------------------------------------------------------------------------

#ifndef Tunel3DH
#define Tunel3DH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Buttons.hpp>
#include <ExtCtrls.hpp>
#include <ComCtrls.hpp>
#include <jpeg.hpp>
#include <Menus.hpp>
#include "vtkBorlandRenderWindow.h"
#include <Dialogs.hpp>
#include <ToolWin.hpp>
#include <ImgList.hpp>
#include <AppEvnts.hpp>

class vtkCellArray;
class vtkCubeSource;
class vtkDoubleArray;
class vtkFloatArray;
class vtkLookupTable;
class vtkPlane;
class vtkPolyData;
class vtkPoints;
class vtkRectilinearGrid;
class vtkSphereSource;



//---------------------------------------------------------------------------
class TPrincipal : public TForm
{
__published:  // IDE-managed Components
        TTimer *tmrTempo;
        TMainMenu *mnuMenu;
        TMenuItem *mnuOpcoes;
        TMenuItem *mnuHelp;
        TMenuItem *opcSobre;
        TColorDialog *CordeFundo;
        TMenuItem *Opes1;
        TMenuItem *CordeFundo1;
        TMenuItem *mnuEscalaCores;
        TMenuItem *Visualizao1;
        TMenuItem *mnuVmax;
        TMenuItem *mnuVmin;
        TMenuItem *mnuCortes;
```

```
TMenuItem *mnuArquivo;
TMenuItem *opcAbrir;
TMenuItem *N1;
TMenuItem *opcSair;
TOpenDialog *dlgAbrir;
TMenuItem *mnuPrincipal;
TMenuItem *mnuDadosSaida;
TMenuItem *Posiodasvelocidades1;
TMenuItem *N2;
TMenuItem *mnuOpcoes1;
TImageList *ImageList1;
TControlBar *ctbBotoes;
TToolBar *ToolBar1;
TToolButton *btnContinuar;
TToolButton *btnParar;
TToolButton *btnSeparador;
TToolButton *btnIniciar;
TToolBar *ToolBar2;
TImageList *ImageList2;
TToolButton *tbnCubo;
TToolButton *tbnEsfera;
TToolButton *tbnSemi1;
TToolButton *tbnSemi2;
TToolBar *ToolBar3;
TImageList *ImageList3;
TToolButton *tbnPrincipal;
TToolButton *tbnCortes;
TToolButton *tbnOpcoes;
TToolButton *tbnDadosSaida;
TGroupBox *GroupBox1;
TLabel *Label1;
TLabel *lblFPS;
TToolButton *tbnAsa;
void __fastcall tmrTempoTimer(TObject *Sender);
void __fastcall opcSobreClick(TObject *Sender);
void __fastcall FormClose(TObject *Sender, TCloseAction &Action);
void __fastcall CordeFundo1Click(TObject *Sender);
void __fastcall mnuCortesClick(TObject *Sender);
void __fastcall opcSairClick(TObject *Sender);
void __fastcall opcAbrirClick(TObject *Sender);
void __fastcall mnuPrincipalClick(TObject *Sender);
void __fastcall mnuDadosSaidaClick(TObject *Sender);
void __fastcall btnContinuarClick(TObject *Sender);
void __fastcall mnuOpcoes1Click(TObject *Sender);
void __fastcall btnPararClick(TObject *Sender);
void __fastcall btnIniciarClick(TObject *Sender);
void __fastcall tbnCuboClick(TObject *Sender);
void __fastcall tbnEsferaClick(TObject *Sender);
void __fastcall tbnSemi1Click(TObject *Sender);
void __fastcall tbnSemi2Click(TObject *Sender);
```

```
        void __fastcall tbnPrincipalClick(TObject *Sender);
        void __fastcall tbnCortesClick(TObject *Sender);
        void __fastcall tbnOpcoesClick(TObject *Sender);
        void __fastcall tbnDadosSaidaClick(TObject *Sender);
        void __fastcall mnuEscalaCoresClick(TObject *Sender);
        void __fastcall mnuVmaxClick(TObject *Sender);
        void __fastcall mnuVminClick(TObject *Sender);
        void __fastcall tbnAsaClick(TObject *Sender);
private:        // User declarations
public:         // User declarations
        __fastcall TPrincipal(TComponent* Owner);


vtkCellArray *poly;
vtkCubeSource *cubo;
vtkDoubleArray *array;
vtkFloatArray *vetores;
vtkLookupTable *Table;
vtkPlane *plane1;
vtkPlane *plane2;
vtkPoints *pontos;
vtkPolyData *profile;
vtkRectilinearGrid *rgrid;
vtkSphereSource *sphere;
vtkSphereSource *sphere1;
vtkSphereSource *sphere2;
vtkSphereSource *sphereVmax;
vtkSphereSource *sphereVmin;

int     h, i, j, k, l, m, n, o, a, b, c, intValorU,
        Ux, Uy, Uz, intVxi, intVyi, intVzi, intVxj, intVyj, intVzj, intVxk, intVyk, intVzk,
        XX, XX1, YY, YY1, Corte, posXVmax, posYVmax, posZVmax, posXVmin,
posYVmin, posZVmin,
        raio, raio1, raio2,
        x_stream, y_stream, z_stream,
        Parar, ua, ub, uc, ud, ue, uf;

float   eixoX, eixoY, Zoom, BZoom, dblValorU, dblVx, dblVy, dblVz,
        Vx, Vy, Vz, vx[60][60][60], vy[60][60][60], vz[60][60][60],
        x[18], y[18], z[18], Vmax, Vmax1, Vmin,
        Vmaxgrad[60][60][60], Vmax_stream[60][60][60],
        X[60][60][60], Y[60][60][60], Z[60][60][60],
        ContadorFrame, ContadorFPS, difTempo, X1line, X2line, Y1line, Y2line, Z1line,
Z2line,
        Xesfera, Yesfera, XSemi1, YSemi1, XSemi2, YSemi2,
        StreamPlanoOrigemX, StreamPlanoOrigemY, StreamPlanoOrigemZ,
        StreamResolucao, StreamRaio, CuboX, CuboY, CuboZ, CuboCentroX, CuboCentroY,
CuboCentroZ,
        EsferaCentroX, EsferaCentroY, EsferaCentroZ,
```

```
      Semi1CentroX, Semi1CentroY, Semi1CentroZ, Semi2CentroX, Semi2CentroY,
Semi2CentroZ;

float  u[60][45][45];

float   PX, DX, DX0, DX1, DX2, DX3, DX4, DX5, DX6, DX7, FX0, FX1, FX2, FX3, FX4,
FX5, FX6, FX7,
      FX0X1, FX1X2, FX2X3, FX3X4, FX4X5, FX5X6, FX6X7,
      FX0X1X2, FX1X2X3, FX2X3X4, FX3X4X5, FX4X5X6, FX5X6X7,
      FX0X1X2X3, FX1X2X3X4, FX2X3X4X5, FX3X4X5X6, FX4X5X6X7,
      FX0X1X2X3X4, FX1X2X3X4X5, FX2X3X4X5X6, FX3X4X5X6X7,
      FX0X1X2X3X4X5, FX1X2X3X4X5X6, FX2X3X4X5X6X7,
      FX0X1X2X3X4X5X6, FX1X2X3X4X5X6X7,
      FX0X1X2X3X4X5X6X7;

};
//---------------------------------------------------------------------------
extern PACKAGE TPrincipal *Principal;
//---------------------------------------------------------------------------
#endif
```

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "vtkActor.h"
#include "vtkActor2D.h"
#include "vtkAxesActor.h"
#include "vtkCamera.h"
#include "vtkCaptionActor2D.h"
#include "vtkCellArray.h"
#include "vtkCleanPolyData.h"
#include "vtkClipPolyData.h"
#include "vtkConeSource.h"
#include "vtkContourFilter.h"
#include "vtkCubeSource.h"
#include "vtkCutter.h"
#include "vtkDataSetMapper.h"
#include "vtkDoubleArray.h"
#include "vtkFeatureEdges.h"
#include "vtkFloatArray.h"
#include "vtkGeometryFilter.h"
#include "vtkGlyph3D.h"
#include "vtkInteractorObserver.h"
#include "vtkLineWidget.h"
#include "vtkLineSource.h"
#include "vtkLookupTable.h"
#include "vtkObject.h"
#include "vtkPlane.h"
#include "vtkPlaneSource.h"
#include "vtkPointData.h"
#include "vtkPoints.h"
#include "vtkPolyData.h"
#include "vtkPolyDataMapper.h"
#include "vtkProperty.h"
#include "vtkProperty2D.h"
#include "vtkPolyDataMapper.h"
#include "vtkRectilinearGridGeometryFilter.h"
#include "vtkRectilinearGrid.h"
#include "vtkRectilinearGridOutlineFilter.h"
#include "vtkRenderer.h"
#include "vtkRenderWindow.h"
#include "vtkRenderWindowInteractor.h"
#include "vtkRibbonFilter.h"
#include "vtkRuledSurfaceFilter.h"
#include "vtkRungeKutta4.h"
#include "vtkScalarBarActor.h"
#include "vtkScalarBarWidget.h"
#include "vtkSphereSource.h"
#include "vtkStreamLine.h"
```

```
#include "vtkStreamPoints.h"
#include "vtkStreamTracer.h"
#include "vtkStripper.h"
#include "vtkStructuredGrid.h"
#include "vtkStructuredGridOutlineFilter.h"
#include "vtkTextProperty.h"
#include "vtkTransform.h"
#include "vtkTransformPolyDataFilter.h"
#include "vtkTriangleFilter.h"
#include "vtkTubeFilter.h"
#include "vtkUnstructuredGrid.h"

#include "VisPrincipal.h"
#include "Tunel3D.h"
#include "Cortes.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "vtkBorlandRenderWindow"
#pragma resource "*.dfm"
TformVisPrincipal *formVisPrincipal;

//---------------------------------------------------------------------------
__fastcall TformVisPrincipal::TformVisPrincipal(TComponent* Owner)
        : TForm(Owner)
{

}
//---------------------------------------------------------------------------
void __fastcall TformVisPrincipal::FormClose(TObject *Sender,
      TCloseAction &Action)
{
Principal->mnuPrincipal->Checked = false;
Principal->tbnPrincipal->Down = false;
}
//---------------------------------------------------------------------------

void __fastcall TformVisPrincipal::FormDestroy(TObject *Sender)
{
Principal->mnuPrincipal->Checked = false;
Principal->tbnPrincipal->Down = false;
}
//---------------------------------------------------------------------------


void __fastcall TformVisPrincipal::cbxGradienteClick(TObject *Sender)
{

vtkRectilinearGridGeometryFilter *plane = vtkRectilinearGridGeometryFilter::New();
  plane->SetInput(Principal->rgrid);
```

```
vtkPolyDataMapper *rgridMapper = vtkPolyDataMapper::New();
  rgridMapper->SetInput(plane->GetOutput());
  rgridMapper->SetScalarRange(Principal->rgrid->GetScalarRange());
  rgridMapper->GlobalImmediateModeRenderingOn();

static vtkActor *wireActor = vtkActor::New();
  wireActor->SetMapper(rgridMapper);

if ( cbxGradiente->Checked )
  VTK1->GetRenderer()->AddActor(wireActor);

else
  VTK1->GetRenderer()->RemoveViewProp(wireActor);

plane->Delete();
rgridMapper->Delete();


VTK1->Invalidate();
VTK1->GetRenderer()->ResetCamera();

}

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

void __fastcall TformVisPrincipal::cbxDominioClick(TObject *Sender)
{
// *** Linhas de contorno do domínio computacional

vtkRectilinearGridOutlineFilter *outline = vtkRectilinearGridOutlineFilter::New();
  outline->SetInput(Principal->rgrid);

vtkPolyDataMapper *outlineMapper = vtkPolyDataMapper::New();
  outlineMapper->SetInputConnection(outline->GetOutputPort());
  outlineMapper->GlobalImmediateModeRenderingOn();

static vtkActor *outlineActor = vtkActor::New();
  outlineActor->SetMapper(outlineMapper);
  outlineActor->GetProperty()->SetColor( 0, 0, 0);

if ( cbxDominio->Checked ){
  VTK1->GetRenderer()->AddActor(outlineActor);
  formCortes->VTK2->GetRenderer()->AddActor(outlineActor);
}
else {
  VTK1->GetRenderer()->RemoveViewProp(outlineActor);
  formCortes->VTK2->GetRenderer()->RemoveViewProp(outlineActor);
}

//VTK1->Invalidate();
```

```
VTK1->GetRenderer()->ResetCamera();
//formCortes->VTK2->Invalidate();
formCortes->VTK2->GetRenderer()->ResetCamera();

//*** Fim das linhas de contorno do domínio computacional
}
//---------------------------------------------------------------------------

void __fastcall TformVisPrincipal::cbxStreamTubosClick(TObject *Sender)
{

rakeLine = vtkLineSource::New();
  rakeLine->SetPoint1(Principal->X1line, Principal->Y1line, Principal->Z1line);
  rakeLine->SetPoint2(Principal->X2line, Principal->Y2line, Principal->Z2line);
  rakeLine->SetResolution(Principal->StreamResolucao);

rakePlane = vtkPlaneSource::New();
  rakePlane->SetXResolution(Principal->StreamResolucao);
  rakePlane->SetYResolution(Principal->StreamResolucao);
  rakePlane->SetOrigin(Principal->StreamPlanoOrigemX, Principal->StreamPlanoOrigemY,
Principal->StreamPlanoOrigemZ);
  rakePlane->SetPoint1(Principal->X1line, Principal->Y1line, Principal->Z1line);
  rakePlane->SetPoint2(Principal->X2line, Principal->StreamPlanoOrigemY, Principal-
>Z2line);

sl = vtkStreamTracer::New();
  sl->SetInput(Principal->rgrid);
  sl->SetSource(rakeLine->GetOutput());
  sl->SetIntegratorTypeToRungeKutta4();
  sl->SetMaximumPropagation(Principal->a);
  sl->SetMaximumPropagationUnitToLengthUnit();
  sl->SetInitialIntegrationStep(0.17);
  sl->SetIntegrationDirectionToBoth();
  sl->ComputeVorticityOn();

//rakeLine->Delete();
//rakePlane->Delete();
//sl->Delete();

streamTube  = vtkTubeFilter::New();
  streamTube->SetInputConnection(sl->GetOutputPort());
  streamTube->SetRadius(Principal->StreamRaio);
  streamTube->SetNumberOfSides(5);

vtkPolyDataMapper *StreamTubeMapper = vtkPolyDataMapper::New();
    StreamTubeMapper->SetInputConnection(streamTube->GetOutputPort());
    StreamTubeMapper->SetScalarRange(Principal->rgrid->GetScalarRange());
    StreamTubeMapper->GlobalImmediateModeRenderingOn();

static vtkActor *streamTubeActor = vtkActor::New();
```

```
    streamTubeActor->SetMapper(StreamTubeMapper);


if ( cbxStreamTubos->Checked )  {
   VTK1->GetRenderer()->AddActor(streamTubeActor);
}
else {
   VTK1->GetRenderer()->RemoveViewProp(streamTubeActor);
}

streamTube->Delete();
StreamTubeMapper->Delete();

VTK1->GetRenderer()->ResetCamera();
VTK1->Invalidate();

}
//---------------------------------------------------------------------------

void __fastcall TformVisPrincipal::cbxStreamFitasClick(TObject *Sender)
{

rakeLine = vtkLineSource::New();
  rakeLine->SetPoint1(Principal->X1line, Principal->Y1line, Principal->Z1line);
  rakeLine->SetPoint2(Principal->X2line, Principal->Y2line, Principal->Z2line);
  rakeLine->SetResolution(Principal->StreamResolucao);

rakePlane = vtkPlaneSource::New();
  rakePlane->SetXResolution(Principal->StreamResolucao);
  rakePlane->SetYResolution(Principal->StreamResolucao);
  rakePlane->SetOrigin(Principal->StreamPlanoOrigemX, Principal->StreamPlanoOrigemY,
Principal->StreamPlanoOrigemZ);
  rakePlane->SetPoint1(Principal->X1line, Principal->Y1line, Principal->Z1line);
  rakePlane->SetPoint2(Principal->X2line, Principal->StreamPlanoOrigemY, Principal-
>Z2line);

sl = vtkStreamTracer::New();
  sl->SetInput(Principal->rgrid);
  sl->SetSource(rakeLine->GetOutput());
  sl->SetIntegratorTypeToRungeKutta4();
  sl->SetMaximumPropagation(Principal->a);
  sl->SetMaximumPropagationUnitToLengthUnit();
  sl->SetInitialIntegrationStep(0.17);
  sl->SetIntegrationDirectionToBoth();
  sl->ComputeVorticityOn();

//rakeLine->Delete();
//rakePlane->Delete();
//sl->Delete();
```

```
vtkRuledSurfaceFilter *scalarSurface = vtkRuledSurfaceFilter::New();
  scalarSurface->SetInputConnection(sl->GetOutputPort());
  scalarSurface->SetOffset(2);
  scalarSurface->SetOnRatio(2);
  scalarSurface->PassLinesOn();
  scalarSurface->SetRuledModeToPointWalk();
  scalarSurface->SetRuledModeToResample();
  scalarSurface->SetResolution (22, 22);
  scalarSurface->SetDistanceFactor(25);

vtkPolyDataMapper *streamMapper = vtkPolyDataMapper::New();
  streamMapper->SetInputConnection(scalarSurface->GetOutputPort());
  streamMapper->SetScalarRange(Principal->rgrid->GetScalarRange());
  streamMapper->GlobalImmediateModeRenderingOn();

static vtkActor *streamFitaActor = vtkActor::New();
  streamFitaActor->SetMapper(streamMapper);

scalarSurface->Delete();
streamMapper->Delete();

if ( cbxStreamFitas->Checked )  {
   VTK1->GetRenderer()->AddActor(streamFitaActor);
   }
else {
   VTK1->GetRenderer()->RemoveViewProp(streamFitaActor);
   }

VTK1->GetRenderer()->ResetCamera();
VTK1->Invalidate();

}
//---------------------------------------------------------------------------

void __fastcall TformVisPrincipal::FormShow(TObject *Sender)
{

//VTK1->Invalidate();
}
//---------------------------------------------------------------------------

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

#ifndef VisPrincipalH
#define VisPrincipalH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
```

```
#include "vtkBorlandRenderWindow.h"
#include <ComCtrls.hpp>
#include <ExtCtrls.hpp>
#include <Graphics.hpp>
#include <ToolWin.hpp>
#include <Menus.hpp>

class vtkLineSource;
class vtkPlaneSource;
class vtkStreamTracer;
class vtkTubeFilter;

//---------------------------------------------------------------------------
class TformVisPrincipal : public TForm
{
__published:  // IDE-managed Components
        TPanel *Panel1;
        TvtkBorlandRenderWindow *VTK1;
        TMainMenu *mnuVisPrincipal;
        TMenuItem *Gradiente1;
        TMenuItem *cbxGradiente;
        TMenuItem *cbxStreamTubos;
        TMenuItem *cbxPontos;
        TMenuItem *cbxGlifos;
        TMenuItem *cbxStreamFitas;
        TMenuItem *cbxDominio;
        void __fastcall FormClose(TObject *Sender, TCloseAction &Action);
        void __fastcall FormDestroy(TObject *Sender);
        void __fastcall cbxGradienteClick(TObject *Sender);
        void __fastcall cbxDominioClick(TObject *Sender);
        void __fastcall cbxStreamTubosClick(TObject *Sender);
        void __fastcall cbxStreamFitasClick(TObject *Sender);
        void __fastcall FormShow(TObject *Sender);
private:        // User declarations
public:         // User declarations
        __fastcall TformVisPrincipal(TComponent* Owner);

vtkLineSource *rakeLine;
vtkPlaneSource *rakePlane;
vtkStreamTracer *sl;
vtkTubeFilter *streamTube;

};
//---------------------------------------------------------------------------
extern PACKAGE TformVisPrincipal *formVisPrincipal;
//---------------------------------------------------------------------------
#endif
```

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "vtkRenderWindow.h"
#include "vtkRenderWindowInteractor.h"

#include "VisPrincipal.h"
#include "Opcoes.h"
#include "Tunel3D.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TformOpcoes *formOpcoes;
//---------------------------------------------------------------------------
__fastcall TformOpcoes::TformOpcoes(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TformOpcoes::FormClose(TObject *Sender,
      TCloseAction &Action)
{
Principal->mnuOpcoes1->Checked = false;
}
//---------------------------------------------------------------------------

void __fastcall TformOpcoes::FormDestroy(TObject *Sender)
{
Principal->mnuOpcoes1->Checked = false;
}
//---------------------------------------------------------------------------



void __fastcall TformOpcoes::mnuCMClick(TObject *Sender)
{
if (mnuCM->Checked) {
        formVisPrincipal->VTK1->GetRenderWindow()->SetStereoTypeToRedBlue();
        formVisPrincipal->VTK1->GetRenderWindow()->StereoRenderOn();
}

else
       formVisPrincipal->VTK1->GetRenderWindow()->StereoRenderOff();

}
//---------------------------------------------------------------------------

void __fastcall TformOpcoes::mnuDresdenClick(TObject *Sender)
```

```
{
if (mnuDresden->Checked) {
      formVisPrincipal->VTK1->GetRenderWindow()->SetStereoTypeToDresden();
      formVisPrincipal->VTK1->GetRenderWindow()->StereoRenderOn();
  }
else
formVisPrincipal->VTK1->GetRenderWindow()->StereoRenderOff();

}
//---------------------------------------------------------------------------

void __fastcall TformOpcoes::mnuInterClick(TObject *Sender)
{
if (mnuInter->Checked) {
      formVisPrincipal->VTK1->GetRenderWindow()->SetStereoTypeToInterlaced();
      formVisPrincipal->VTK1->GetRenderWindow()->StereoRenderOn();
  }
else
formVisPrincipal->VTK1->GetRenderWindow()->StereoRenderOff();

}
//---------------------------------------------------------------------------

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

#ifndef OpcoesH
#define OpcoesH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ComCtrls.hpp>
#include <ExtCtrls.hpp>
//---------------------------------------------------------------------------
class TformOpcoes : public TForm
{
__published:  // IDE-managed Components
      TPageControl *pgOpcoes;
      TTabSheet *TabSheet1;
      TTabSheet *TabSheet2;
      TPanel *Panel1;
      TTabSheet *TabSheet3;
      TCheckBox *cbxRaio;
      TRadioButton *mnuVtotal;
      TRadioButton *mnuVz;
      TRadioButton *mnuVy;
      TRadioButton *mnuVx;
      TCheckBox *mnuCM;
      TCheckBox *mnuDresden;
```

```
TCheckBox *mnuInter;
TEdit *edtX1line;
TLabel *Label3;
TEdit *edtY1line;
TEdit *edtZ1line;
TLabel *Label2;
TEdit *edtZ2line;
TEdit *edtY2line;
TLabel *Label6;
TLabel *Label7;
TEdit *edtX2line;
TEdit *edtResolucao;
TLabel *Label8;
TRadioButton *rdbStreamLinhaOrigem;
TRadioButton *rdbStreamPlanoOrigem;
TLabel *Label9;
TEdit *edtPlanoOrigemX;
TEdit *edtPlanoOrigemY;
TEdit *edtPlanoOrigemZ;
TTabSheet *TabSheet4;
TLabel *Label13;
TEdit *edtCuboX;
TLabel *Label14;
TEdit *edtCuboY;
TLabel *Label15;
TEdit *edtCuboZ;
TEdit *edtCuboCentroZ;
TEdit *edtCuboCentroY;
TEdit *edtCuboCentroX;
TLabel *Label19;
TLabel *Label20;
TLabel *Label21;
TEdit *edtEsferaRaio;
TLabel *Label22;
TEdit *edtSemi1Raio;
TGroupBox *GroupBox2;
TGroupBox *GroupBox1;
TGroupBox *GroupBox3;
TEdit *edtSemi1CentroX;
TLabel *Label16;
TEdit *edtSemi1CentroY;
TEdit *edtSemi1CentroZ;
TLabel *Label17;
TLabel *Label18;
TLabel *Label23;
TGroupBox *GroupBox4;
TLabel *Label24;
TLabel *Label25;
TLabel *Label26;
TLabel *Label27;
```

```
        TLabel *Label28;
        TEdit *edtSemi2Raio;
        TEdit *edtSemi2CentroX;
        TEdit *edtSemi2CentroY;
        TEdit *edtSemi2CentroZ;
        TLabel *Label29;
        TEdit *edtEsferaCentroX;
        TLabel *Label30;
        TEdit *edtEsferaCentroY;
        TLabel *Label31;
        TEdit *edtEsferaCentroZ;
        TLabel *Label32;
        TLabel *Label4;
        TEdit *edtStreamRaio;
        TLabel *Label5;
        TGroupBox *GroupBox5;
        TGroupBox *GroupBox6;
        TTrackBar *trkY1;
        TTrackBar *trkY2;
        TTrackBar *trkY3;
        TTrackBar *trkY4;
        TTrackBar *trkX4;
        TTrackBar *trkX3;
        TTrackBar *trkX2;
        TTrackBar *trkX1;
        void __fastcall FormClose(TObject *Sender, TCloseAction &Action);
        void __fastcall FormDestroy(TObject *Sender);
        void __fastcall mnuCMClick(TObject *Sender);
        void __fastcall mnuDresdenClick(TObject *Sender);
        void __fastcall mnuInterClick(TObject *Sender);
private:        // User declarations
public:         // User declarations
        __fastcall TformOpcoes(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TformOpcoes *formOpcoes;
//---------------------------------------------------------------------------
#endif
```

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "DadosSaida.h"
#include "Tunel3D.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TformDadosSaida *formDadosSaida;
//---------------------------------------------------------------------------
__fastcall TformDadosSaida::TformDadosSaida(TComponent* Owner)
        : TForm(Owner)
{

}
//---------------------------------------------------------------------------
void __fastcall TformDadosSaida::FormClose(TObject *Sender,
      TCloseAction &Action)
{
Principal->mnuDadosSaida->Checked = false;
}
//---------------------------------------------------------------------------

void __fastcall TformDadosSaida::FormDestroy(TObject *Sender)
{
Principal->mnuDadosSaida->Checked = false;
}
//---------------------------------------------------------------------------

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

#ifndef DadosSaidaH
#define DadosSaidaH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//---------------------------------------------------------------------------
class TformDadosSaida : public TForm
{
__published:  // IDE-managed Components
        TGroupBox *GroupBox1;
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *lblX1;
        TLabel *lblY1;
```

```
        TLabel *lblZ1;
        TGroupBox *GroupBox2;
        TLabel *lblVmax;
        TLabel *Label4;
        TLabel *lblVmaxX;
        TLabel *lblVmaxY;
        TLabel *lblVmaxZ;
        TGroupBox *GroupBox3;
        TLabel *lblVmin;
        TLabel *Label6;
        TLabel *lblVminX;
        TLabel *lblVminY;
        TLabel *lblVminZ;
        TGroupBox *GroupBox4;
        TLabel *lblVx;
        TLabel *Label7;
        TLabel *lblVxi;
        TLabel *lblVxj;
        TLabel *lblVxk;
        TGroupBox *GroupBox5;
        TLabel *lblVy;
        TLabel *Label12;
        TLabel *lblVyi;
        TLabel *lblVyj;
        TLabel *lblVyk;
        TGroupBox *GroupBox6;
        TLabel *lblVz;
        TLabel *Label17;
        TLabel *lblVzi;
        TLabel *lblVzj;
        TLabel *lblVzk;
        TGroupBox *GroupBox7;
        TLabel *lblPotInicio;
        TLabel *Label5;
        TLabel *Label8;
        TLabel *Label9;
        TLabel *lblPotMeio;
        TLabel *lblPotFim;
        void __fastcall FormClose(TObject *Sender, TCloseAction &Action);
        void __fastcall FormDestroy(TObject *Sender);
private:        // User declarations
public:         // User declarations
        __fastcall TformDadosSaida(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TformDadosSaida *formDadosSaida;
//---------------------------------------------------------------------------
#endif
```

```
//---------------------------------------------------------------------------
#include "vtkActor.h"
#include "vtkPolyDataMapper.h"
#include "vtkRectilinearGrid.h"
#include "vtkRectilinearGridGeometryFilter.h"
#include "vtkRenderWindow.h"
#include "vtkRenderWindowInteractor.h"
#include "vtkLookupTable.h"


#include <vcl.h>
#pragma hdrstop

#include "Cortes.h"
#include "Tunel3D.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "vtkBorlandRenderWindow"
#pragma resource "*.dfm"
TformCortes *formCortes;
//---------------------------------------------------------------------------
__fastcall TformCortes::TformCortes(TComponent* Owner)
        : TForm(Owner)
{

}
//---------------------------------------------------------------------------
void __fastcall TformCortes::FormClose(TObject *Sender,
      TCloseAction &Action)
{
Principal->mnuCortes->Checked = false;
Principal->tbnCortes->Down = false;

}
//---------------------------------------------------------------------------

void __fastcall TformCortes::FormDestroy(TObject *Sender)
{
Principal->mnuCortes->Checked = false;
Principal->tbnCortes->Down = false;
}
//---------------------------------------------------------------------------

void __fastcall TformCortes::rdbGraXYClick(TObject *Sender)
{

planoXY = vtkRectilinearGridGeometryFilter::New();
planoXY->SetInput(Principal->rgrid);
planoXY->SetExtent ( 0,Principal->a, 0, Principal->b, Principal->Corte, Principal->Corte);
```

```
vtkPolyDataMapper *MapperXY = vtkPolyDataMapper::New();
  MapperXY->SetInput(planoXY->GetOutput());
  MapperXY->SetScalarRange(Principal->rgrid->GetScalarRange());
  //MapperXY->SetLookupTable(Principal->Table);
  MapperXY->GlobalImmediateModeRenderingOn();

static vtkActor * PlanodeCorteXY = vtkActor::New();
  PlanodeCorteXY->SetMapper(MapperXY);

if (rdbGraXY->Checked){
    VTK2->GetRenderer()->AddActor(PlanodeCorteXY);
}
else
    VTK2->GetRenderer()->RemoveViewProp(PlanodeCorteXY);

  planoXY->Delete();
  MapperXY->Delete();
}
//---------------------------------------------------------------------------

void __fastcall TformCortes::rdbIsoXYClick(TObject *Sender)
{
vtkContourFilter *ContourXY = vtkContourFilter::New();
ContourXY->SetInput(Principal->rgrid);
ContourXY->GenerateValues(23, Principal->rgrid->GetScalarRange());
//ContourXY->Update();

planoCorteXY = vtkPlane::New();
planoCorteXY->SetNormal(0, 0, 1);
planoCorteXY->SetOrigin (0, 0, Principal->Corte);

vtkCutter *CutterXY = vtkCutter::New();
   CutterXY->SetInput(ContourXY->GetOutput());
   CutterXY->SetCutFunction(planoCorteXY);

vtkPolyDataMapper *isoMapperXY = vtkPolyDataMapper::New();
  isoMapperXY->SetInput(CutterXY->GetOutput());
  isoMapperXY->SetScalarRange(Principal->rgrid->GetScalarRange());
  //isoMapperXY->SetLookupTable(Table);
  isoMapperXY->GlobalImmediateModeRenderingOn();

static vtkActor * isoPlanodeCorteXY = vtkActor::New();
  isoPlanodeCorteXY->SetMapper(isoMapperXY);

if (rdbIsoXY->Checked)
       VTK2->GetRenderer()->AddActor(isoPlanodeCorteXY);
else
       VTK2->GetRenderer()->RemoveViewProp(isoPlanodeCorteXY);
```

```
  ContourXY->Delete();
  CutterXY->Delete();
  planoCorteXY->Delete();
  isoMapperXY->Delete();
}
//---------------------------------------------------------------------------

void __fastcall TformCortes::rdbGraXZClick(TObject *Sender)
{
planoXZ = vtkRectilinearGridGeometryFilter::New();
planoXZ->SetInput(Principal->rgrid);
planoXZ->SetExtent ( 0,Principal->a, Principal->Corte, Principal->Corte, 0,Principal->c);

  vtkPolyDataMapper *MapperXZ = vtkPolyDataMapper::New();
  MapperXZ->SetInput(planoXZ->GetOutput());
  MapperXZ->SetScalarRange(Principal->rgrid->GetScalarRange());
  MapperXZ->GlobalImmediateModeRenderingOn();

static  vtkActor * PlanodeCorteXZ = vtkActor::New();
  PlanodeCorteXZ->SetMapper(MapperXZ);

if (rdbGraXZ->Checked){
    VTK2->GetRenderer()->AddActor(PlanodeCorteXZ);
}
else
    VTK2->GetRenderer()->RemoveViewProp(PlanodeCorteXZ);

  planoXZ->Delete();
  MapperXZ->Delete();
}
//---------------------------------------------------------------------------

void __fastcall TformCortes::rdbIsoXZClick(TObject *Sender)
{
vtkContourFilter *ContourXZ = vtkContourFilter::New();
ContourXZ->SetInput(Principal->rgrid);
ContourXZ->GenerateValues(23, Principal->rgrid->GetScalarRange());
//ContourXY->Update();

planoCorteXZ = vtkPlane::New();
planoCorteXZ->SetNormal(0, 1, 0);
planoCorteXZ->SetOrigin (0, Principal->Corte, 0);

vtkCutter *CutterXZ = vtkCutter::New();
   CutterXZ->SetInput(ContourXZ->GetOutput());
   CutterXZ->SetCutFunction(planoCorteXZ);

vtkPolyDataMapper *isoMapperXZ = vtkPolyDataMapper::New();
  isoMapperXZ->SetInput(CutterXZ->GetOutput());
  isoMapperXZ->SetScalarRange(Principal->rgrid->GetScalarRange());
```

```
  //isoMapperXY->SetLookupTable(Table);
  isoMapperXZ->GlobalImmediateModeRenderingOn();

static vtkActor * isoPlanodeCorteXZ = vtkActor::New();
  isoPlanodeCorteXZ->SetMapper(isoMapperXZ);

if (rdbIsoXZ->Checked)
      VTK2->GetRenderer()->AddActor(isoPlanodeCorteXZ);
else
      VTK2->GetRenderer()->RemoveViewProp(isoPlanodeCorteXZ);

  ContourXZ->Delete();
  CutterXZ->Delete();
  planoCorteXZ->Delete();
  isoMapperXZ->Delete();
}
//---------------------------------------------------------------------------

void __fastcall TformCortes::rdbIsoYZClick(TObject *Sender)
{
vtkContourFilter *ContourYZ = vtkContourFilter::New();
ContourYZ->SetInput(Principal->rgrid);
ContourYZ->GenerateValues(23, Principal->rgrid->GetScalarRange());

planoCorteYZ = vtkPlane::New();
planoCorteYZ->SetNormal(1, 0, 0);
planoCorteYZ->SetOrigin (Principal->Corte, 0, 0);

vtkCutter *CutterYZ = vtkCutter::New();
   CutterYZ->SetInput(ContourYZ->GetOutput());
   CutterYZ->SetCutFunction(planoCorteYZ);

vtkPolyDataMapper *isoMapperYZ = vtkPolyDataMapper::New();
  isoMapperYZ->SetInput(CutterYZ->GetOutput());
  isoMapperYZ->SetScalarRange(Principal->rgrid->GetScalarRange());
  isoMapperYZ->GlobalImmediateModeRenderingOn();

static vtkActor * isoPlanodeCorteYZ = vtkActor::New();
  isoPlanodeCorteYZ->SetMapper(isoMapperYZ);

if (rdbIsoYZ->Checked)
      VTK2->GetRenderer()->AddActor(isoPlanodeCorteYZ);
else
      VTK2->GetRenderer()->RemoveViewProp(isoPlanodeCorteYZ);

  ContourYZ->Delete();
  CutterYZ->Delete();
  planoCorteYZ->Delete();
  isoMapperYZ->Delete();
}
```

```
//---------------------------------------------------------------------------

void __fastcall TformCortes::rdbGraYZClick(TObject *Sender)
{
planoYZ = vtkRectilinearGridGeometryFilter::New();
planoYZ->SetInput(Principal->rgrid);
planoYZ->SetExtent (Principal->Corte, Principal->Corte, 0,Principal->b, 0,Principal->c);

  vtkPolyDataMapper *MapperYZ = vtkPolyDataMapper::New();
  MapperYZ->SetInput(planoYZ->GetOutput());
  MapperYZ->SetScalarRange(Principal->rgrid->GetScalarRange());
  MapperYZ->GlobalImmediateModeRenderingOn();

static  vtkActor * PlanodeCorteYZ = vtkActor::New();
  PlanodeCorteYZ->SetMapper(MapperYZ);

if (rdbGraYZ->Checked){
    VTK2->GetRenderer()->AddActor(PlanodeCorteYZ);
}
else
    VTK2->GetRenderer()->RemoveViewProp(PlanodeCorteYZ);

  planoYZ->Delete();
  MapperYZ->Delete();
}
//---------------------------------------------------------------------------

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

#ifndef CortesH
#define CortesH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include "vtkBorlandRenderWindow.h"
#include <ExtCtrls.hpp>
#include <Menus.hpp>
#include <ComCtrls.hpp>

#include "vtkContourFilter.h"
#include "vtkCutter.h"
#include "vtkPlane.h"
#include "vtkRectilinearGridGeometryFilter.h"

//---------------------------------------------------------------------------
class vtkRectilinearGridGeometryFilter;
class vtkPlane;
```

```
class TformCortes : public TForm
{
__published:  // IDE-managed Components
        TvtkBorlandRenderWindow *VTK2;
        TPanel *Panel1;
        TMainMenu *MainMenu1;
        TMenuItem *mnuPlanoXY;
        TMenuItem *mnuPlanoXZ;
        TMenuItem *mnuPlanoYZ;
        TMenuItem *rdbGraXY;
        TMenuItem *rdbIsoXY;
        TMenuItem *rdbGraXZ;
        TMenuItem *rdbIsoXZ;
        TMenuItem *rdbGraYZ;
        TMenuItem *rdbIsoYZ;
        TTrackBar *tkbHorizontal;
        TLabel *lblCorte;
        TLabel *Label1;
        void __fastcall FormClose(TObject *Sender, TCloseAction &Action);
        void __fastcall FormDestroy(TObject *Sender);
        void __fastcall rdbGraXYClick(TObject *Sender);
        void __fastcall rdbIsoXYClick(TObject *Sender);
        void __fastcall rdbGraXZClick(TObject *Sender);
        void __fastcall rdbIsoXZClick(TObject *Sender);
        void __fastcall rdbIsoYZClick(TObject *Sender);
        void __fastcall rdbGraYZClick(TObject *Sender);
private:        // User declarations
public:         // User declarations
        __fastcall TformCortes(TComponent* Owner);

vtkPlane *planoCorteXY;
vtkPlane *planoCorteXZ;
vtkPlane *planoCorteYZ;
vtkRectilinearGridGeometryFilter *planoXY;
vtkRectilinearGridGeometryFilter *planoXZ;
vtkRectilinearGridGeometryFilter *planoYZ;

};
//---------------------------------------------------------------------------
extern PACKAGE TformCortes *formCortes;
//---------------------------------------------------------------------------
#endif
```