Implementação da estrutura de dados Halfedge visando aplicações em mecânica de fluidos computacional

Archibald de Araujo Silva

DISSERTAÇÃO APRESENTADA
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA
OBTENÇÃO DO TÍTULO DE MESTRE
EM
CIÊNCIAS

Área de concentração: Matemática Aplicada
Orientador: Prof Dr. Alexandre Megiorin Roma

Durante a elaboração deste trabalho o autor recebeu auxílio financeiro do CNPQ.

São Paulo, 01 de setembro de 2006 .

Implementação da estrutura de dados
Halfedge visando aplicações em
mecânica de fluidos computacional

Este exemplar corresponde à redação
final da dissertação devidamente corrigida
e defendida por Archibald de Araujo Silva
e aprovada pela comissão julgadora.

São Paulo, 1 de setembro de 2006.

Banca examinadora:

- Prof. Dr. Alexandre Megiorin Roma (Orientador) - IME - USP
- Prof. Dr. Luis Carlos de Castro Santos - IME - USP
- Prof. Dr. Antônio Castelo Filho - ICMC - USP - São Carlos

Dedico esta dissertação
aos meus pais, a minha esposa e filhas

"Não desistiremos de explorar
E o fim de todas nossas explorações
Será chegarmos ao lugar de onde partimos
E conhecer o lugar pela primeira vez."
Thomas Stearns Eliot (1888-1965).

Agradecimentos

Agradeço aos meus pais pelo carinho e amor que sempre me dedicaram.

Agradeço a minha amada esposa, por seu amor, paciência e dedicação durante toda esta jornada; sem ela eu não teria conseguido.

Agradeço as minhas filhas, pela compreensão e carinho.

Agradeço ao meus avós, aos meus irmãos e a toda minha família pelo apoio que sempre me deram.

Agradeço o meu orientador $Prof.\ Dr.$ Alexandre Megiorin Roma, por todos seus conselhos e orientações, por toda sua paciência em nossas longas reuniões, por toda sua generosidade como ser humano e por sua genuína dedicação na difícil arte de ensinar e orientar.

Agradeço ao $Prof.\ Dr$ Paulo Domingos Cordaro, a $Prof^a\ Dr^a$. Heloisa Daruiz Borsari e a $Prof^a\ Dr^a$. Ana Flora Pereira de Castro Humes, por terem me dado uma carta de recomendação para ingresso no mestrado.

Agradeço ao meu orientador de programa, o $Prof.\ Dr.$ André Salles de Carvalho, por sua ajuda e incentivo.

Agradeço ao corpo de professores do IME pela dedicação e empenho no ensino da matemática.

Agradeço ao grupo de Dinâmica de Fluidos Computacional do Laboratório de Computação de Alto Desempenho da Universidade de São Paulo em São Carlos pela cessão de direito de uso do software FreeFlow e, de forma especial, ao $Prof.\ Dr.$ Antonio Castelo Filho por todas suas explicações sobre a utilização deste software.

Agradeço ao grupo do Laboratório de Transferência de Calor e Massa e Dinâmica dos Fluidos da Faculdade de Engenharia Mecânica da Universidade Federal de Uberlândia pelo envio do tutorial do GMSH em português.

Agradeço aos colegas do IME, Rudimar Nós, Olga Saito, Alessandro Santana, Daniel Mendes Azeredo, Millena Villar, Eder Annibal e a todos os outros companheiros, que muito auxiliaram neste trabalho.

Agradeço a todos os funcionários do IME por toda atenção que sempre me dispensaram.

Agradeço ao CAPES, que através da aprovação do processo n.04/13781-1, permitiu a

compra de computadores nos quais os principais resultados computacionais deste trabalho foram obtidos.

Agradeço ao CNPQ pela Bolsa de Estudos concedida durante o período de mestrado.

Resumo

Utiliza-se o programa livre GMSH para se representar um sólido por intermédio de sua superfície externa (fronteira imersa) e também para se gerar uma malha triangular não estruturada para discretizar esta superfície. Os dados desta malha são armazenados em uma estrutura de dados topológica chamada Halfedge, a qual é largamente utilizada em Computação Gráfica para se representar superfícies fechadas e orientadas.

Uma vez que o sólido já está armazenado nesta estrutura de dados, passa-se a estudar uma hipotética interação dinâmica entre a fronteira imersa e o escoamento do fluido. Esta interação é estudada apenas em um sentido, considera-se apenas a condição de não deslizamento, isto é, a fronteira imersa acompanhará passivamente um campo de velocidades pré-estabelecido (imposto), sem exercer qualquer forca ou influência sobre ele.

Neste estudo, utiliza-se o método da fronteira imersa, que faz uso de dois tipos básicos de malha computacional: a primeira, é chamada de malha euleriana, e é utilizada para representar o fluído e a segunda, chamada de malha lagrangeana, e é utilizada para representar o contorno do sólido em contato com o fluido.

Finalmente, para acompanhar e visualizar de forma mais precisa o comportamento dos pontos do fluido nas vizinhanças da fronteira do sólido, foi utilizado uma técnica chamada de malha adaptativa. Esta técnica consiste em fazer o refinamento automático de uma malha inicial, em algumas regiões do domínio próximas à superficie onde, em geral, ocorrem fenômenos importantes no escoamento do fluido.

Índice

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1

Introdução

Na Seção 1.1, são apresentados a motivação e os objetivos deste trabalho; na Seção 1.2, apresenta-se uma breve revisão da literatura sobre o Método da Fronteira Imersa, sobre o refinamento automático de malhas e sobre os programas gratuitos que são utilizados para desenhar um sólido e gerar malhas sobre sua superfície e na Seção 1.3, mostra-se como estão divididos os capítulos deste trabalho.

1.1 Motivação e Objetivos

Na área de mecânica dos fluidos computacional, tem-se vivenciado uma fase de grandes avanços, tanto no desenvolvimento de novas ferramentas computacionais, como na aplicação a novos problemas em engenharia, física, medicina, etc. Muito esforço tem sido empregado para se obter metodologias que permitam uma boa relação custo-benefício, isto é, alta precisão a um custo computacional baixo.

Um dos grandes desafios que se apresentam é o de se obter ferramentas que permitam simular escoamentos ao redor de corpos em geometrias complexas nos quais podem estar imersas estruturas móveis e/ou deformáveis. Em busca destas ferramentas, diversas metodologias têm sido propostas as quais empregam discretizações via diferenças finitas, elementos finitos ou volumes finitos. As duas últimas são empregadas usualmente para o estudo de domínios mais complexos estando, entretanto, sujeitas a processos de remalhagem que, em geral, são muito custosos computacionalmente. Dentre os métodos comumente empregados encontram-se: o Modelo de Gás em Rede (Lattice Boltzmann Method) [26], [27] e [28]; Métodos de

Malha Móvel (Moving Mesh Methods) [25], de acompanhamento e de captura da interface (Front-Tracking e Front-Capturing Methods) [29].

Um outro grande desafio é a simulação de escoamentos multifásicos, tais como, emulsões e espumas, onde interfaces de separações estão presentes (e.g. bolhas e gotas). Nos escoamentos estratificados, a interface que separa as fases está sujeita à tensão superficial e o escoamento a tensões de cisalhamento. Alguns trabalhos nesta área, foram feitos por [32], que apresenta um estudo da estrutura de um escoamento bifásico água-ar em tubos de Venturini com diferentes relações de área; ou por [33], que aborda a questão de independência do sistema de referência das forças interfacias em relação às forças que aparecem sobre a fase dispersa diferentes das forças de arrasto (non-drag-forces); ou por [34], que propõem um novo modelo que é obtido a partir da integração das forças de pressão que atuam sobre uma bolha que se movimenta num escoamento irrotacional; ou ainda por [35], que estuda diferentes algoritmos utilizados para o tratamento numérico das forças interfacias.

As dificuldades presentes nas simulações de escoamentos envolvendo geometrias complexas e escoamentos multifásicos contendo superfícies de separação podem ser contornadas pelo uso do Método da Fronteira Imersa [5]. Este método permite que a solução das equações de Navier-Stokes num malha cartesiana estruturada fixa (malha euleriana) e a descrição da presença da interface por intermédio de uma malha lagrangeana que acompanha a fronteira imersa. Entretanto, o Método da Fronteira Imersa exige uma alta densidade de pontos na malha euleriana para fornecer bons resultados numéricos, em escoamentos com altos números de Reynolds. Para diminuir o custo computacional que malhas densas acarretam e de, ao mesmo tempo, capturar detalhes geométricos e fenômenos locais de forma mais precisa, emprega-se algum tipo adaptativo de malha. No presente trabalho, optou-se por adotar a técnica de malha adaptativa introduzida por Berger [9] e utilizadas nos trabalhos de [5], [13] e [24], a qual consiste basicamente em aproximar soluções numéricas de equações diferenciais parciais, numa sequência hierárquica de malhas apropriadamente aninhadas. O refinamento adaptativo da malha euleriana, permite que o esforco computacional seja dispendido, onde ele é mais necessário, como por exemplo, nas regiões físicas com geometrias complexas.

Os objetivos principais deste trabalho são:

- Explorar uma ferramenta que permita gerar a malha lagrangeana que representa a fronteira imersa.

- Implementar uma estrutura de dados que consuma pouco memória computacional e que permita um rápido acesso às informações da fronteira imersa.

Para gerar a fronteira imersa, utiliza-se o GMSH, um software livre e para a estrutura de dados, adota-se a Halfedge, explicadas nos Capítulos 2 e 3, respectivamente. Este trabalho fornecerá base para estudos mais complexos envolvendo aplicações com fronteira imersa e sua interação com o escoamento de fluidos.

1.2 Revisão da Literatura

Em 1972, Peskin [5] e [6], introduziu um modelo matemático e um método computacional para estudar o escoamento sangüíneo ao redor da vávula mitral. Em contraste com outros autores, ele não fez qualquer hipótese particular acerca da geometria deste escoamento para observar o movimento da válvula. Partindo apenas das leis de Newton e de características físicas e fisiólogicas do fluido, da musculatura e da válvula cardíaca, deduziu as equações dinâmicas do movimento, as quais descrevem a forte interação existente entre a válvula e o escoamento sangüíneo.

Para simplificar a implementação computacional do método, todas as estruturas cardíacas foram assumidas com tendo a mesma densidade e completamente imersa em sangue (como afirma Peskin: "... um coração pulsando num aquário de fluido, ao invés de em seu próprio lugar."). Por este motivo, posteriormente, o método de Peskin tornou-se conhecido com *Método da Fronteira Imersa*.

O Método da Fronteira Imersa considera que a fronteira está acoplada ao movimento do fluido exercendo sobre ele uma força singular cuja intensidade é grande nas vizinhanças da fronteira e zero em todos os outros lugares, além de estar se movendo com a velocidade local do fluido. A integração das equações do fluido é feita empregando-se a malha computacional euleriana que discretiza o domínio espacial e para os pontos da fronteira imersa, em geral, adota-se uma representação lagrangeana. Em cada passo no tempo, o método usa o posicionamento da fronteira no início desse passo no tempo para calcular as forcas definidas pela fronteira. Estas forças são, então, aplicadas nas vizinhanças dos pontos da malha do fluido. Desde de que o fluido seja incompressível, o espalhamento dos efeitos desta força por intermédio do fluido por ação do campo de pressão é imediato. A nova velocidade do fluido é

usada para mover a fronteira, completando assim um passo no tempo.

No trabalho aqui desenvolvido, a interação dinâmica entre a fronteira imersa e o escoamento do fluido é estudada apenas em um sentido: considera-se apenas a condição de não deslizamento, a fronteira imersa acompanhará passivamente um campo de velocidades pré-estabelecido (imposto), sem exercer qualquer forca ou influenciar este campo de qualquer maneira. O campo de velocidades que atua sobre a fronteira imersa é interpolado a partir do campo de velocidades do fluido na vizinhança desta fronteira. Este processo de interpolação será melhor explicado no Capítulo 4.

O Método da Fronteira Imersa estabelece indiretamente uma condição de contorno na superfície imersa de modo a que ela se faça presente no escoamento. Isto é feito por intermédio de um campo de força inserido como termo forçante nas equações de Navier-Stokes. Uma proposta de classificação dos Métodos da Fronteira Imersa é feita por [36], e, segundo a qual, este métodos podem ser divididos em:

- **A) Métodos de Força Contínua**

  Neste modelo, o termo forçante é incorporado às equações de Navier-Stokes na forma contínua, isto é, antes que estas equações sejam discretizadas. Este modelo, por sua vez, é também subdividido em:

  A.1) Fronteiras Elásticas

  O Fluido é representado pelas equações de Navier-Stokes, que são resolvidas sobre todo o domínio computacional e os pontos da malha que representam a fronteira imersa, são supostos estarem conectados entre si por uma força elástica. Trabalhos nesta área, foram realizados por [5], [13] e [37].

  A.2) Fronteiras Rígidas

  Neste modelo, utilizam-se constantes ad-hoc na formulação da força e, desta forma, procura-se encontrar uma força que ajuste a velocidade do fluido sobre a fronteira imersa à velocidade da própria fronteira. Trabalhos nesta área, foram desenvolvidos por [38] e [39].

- **B) Métodos de Força Discreta**

  Neste modelo, o termo forçante é introduzido após as equações de Navier-Stokes terem sido discretizadas. Este modelo, por sua vez, é também subdividido em:

B.1) Imposição Indireta da Condição de Contorno

Neste modelo, a imposição da condição de contorno é feita de forma indireta, isto é, deve-se obter uma força que, ao ser inserida nas equações de Navier-Stokes, leve a obtenção da condição de contorno especificada para a fronteira imersa. Trabalhos nesta área, foram realizados por [40] e [41].

B.2) Imposição Direta da Condição de Contorno

Neste modelo, a imposição da condição de contorno é feita de forma direta, ou seja, impõe-se à fronteira imersa uma condição de contorno desejada, utilizando-se uma função extrapoladora. Trabalhos nesta área, foram desenvolvidos por [42] e [43].

O próximo assunto a ser tratado é sobre o algoritmo de refinamento de malha adaptativa.

Todas as técnicas e algoritmos empregados para se fazer o refinamento local de malhas adaptativas, serão chamados de AMR, as quais podem ser usadas para gerar vários blocos uniforme de malha acomodando-os uns sobre os outros.

Esta técnica for introduzida por Berger [9] para a solução numérica de equações diferenciais hiperbólicas e desde então esta técnica tem sido desenvolvida e extendida para ajudar na solução de diferentes problemas, dentre os quais podemos citar: escoamentos multifásicos e mais recentemente em escoamento de fluidos incompressíveis [8] e [10].

Em termos gerais, o algoritmo AMR aproxima soluções numéricas de equações diferenciais parciais numa sequência hierárquica de malhas aninhadas em forma de paralelepípedo. O processo de refinamento permite que regiões físicas complicadas (por exemplo, com muitos contornos, recortes e pontas) que estão inseridas no domínio computacional , possam ser tratadas por intermédio do uso de um sistema de coordenadas que vai se ajustando, tanto quanto se queira (limitado à memória computacional), a estas partes problemáticas de forma a captar os detalhes de interesse.

O algoritmo começa com a estimação do erro da solução numérica em todos os pontos da malha, depois disto, os pontos, onde a estimativa de erro exceder a um certo valor crítico, são marcados e agrupados juntamente com alguns outros pontos que não foram marcados, formando outras malhas menores (sub-malhas ou retalhos de mallas) contidas na malha global. O resultado deste algoritmo é o refinamento de uma região, cuja forma final, é dada pela união de pequenas malhas em forma de paralelepípedo.

A princípio, os paralelepípedos formados, a partir deste refinamento podem assumir posições arbitrárias, isto é, eles podem estar alinhados ou rotacionados uns com relação aos outros e também com respeito a malha global, podendo inclusive estarem superpostos. Em qualquer caso, cada retalho de malha que foi criado tem sua própria solução numérica armazenada, utilizando-se o mínimo de memória necessária para descrever a exata localização e tamanho de cada um destes retalhos.

Usualmente o método de refinamento é aplicado recursivamente, ou seja, se uma malha, que acabou de ser refinada, ainda apresentar pontos com estimativa de erro maior do que um certo valor crítico; um outro nível de malha mais refinada é criado e assim sucessivamente até que se obtenha a precisão requerida ou interrompermos este processo por qualquer outro critério de parada, caso a precisão não seja atingida.

A Figura 1.1 mostra o resultado, em duas dimensões, de um corte paralelo a uma das faces do domínio, de uma malha em forma de paralelepípedo. Pode-se notar que a malha que foi gerada pelo algorítimo AMR, é composta por três níveis e que as malhas não estão rotacionadas umas com relacão às outras.

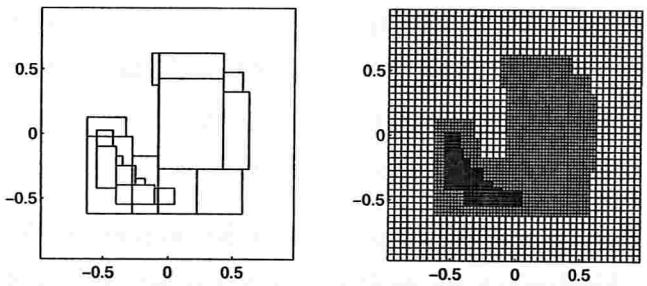


Figura 1.1: Corte de uma malha gerada pelo algoritmo AMR: Só o contorno da malha (à esquerda) e incluindo todas as suas linhas (à direita) [13].

Basicamente a construção de uma malha composta pode ser dividida em três passos independentes:

- *Estimador de Erro*: O estimador de erro determina, de acordo com algum critério, onde a acuracidade da solução não é suficiente.
- *Gerador da Malha*: O gerador de malha deve ser capaz de criar malhas finas que cubram todas as regiões que necessitem de refinamento.

- *Estrutura de Dados*: A estrutura de dados deve permitir, de uma forma rápida e com baixo consumo de memória computacional, atualizações para incluir (ou remover) as novas (ou as velhas) malhas.

É importante ressaltar que o algoritmo AMR, depende, também, do tipo de problema que está sendo resolvido (por exemplo, se o fluido é compressível ou incompressível) e dos lugares onde as variáveis de estado são colocadas (por exemplo, no centro da célula, ou na face da célula).

Finalmente, pode se dizer que o algoritmo AMR tem sido bastante utilizado, ao longo dos anos, com bons resultados, numa grande variedade de problemas de escoamento tanto com fluidos compressíveis quanto com incompressíveis, como pode ser visto no trabalho de [24], que trata do refinamento automático de malha adaptativa para as equações de Euler visando aplicações em engenharia aeroespacial; ou no trabalho de [30], que trata da geração de malhas cartesianas robustas e eficientes baseadas em componentes da geometria de um avião; ou ainda no trabalho de [31], que trata de um algoritmo de refinamento de malha adaptativa tridimensional para a solução de sistemas hiperbólicos das leis de conservação.

Faremos agora uma breve apanhado sobre dois programas gratuitos que podem ser utilizados para a geração da geometria de um sólido e de malhas bidimensionais e/ou tridimensionais. Alguns destes programas permitem simular escoamento de fluidos. A intenção não é fazer uma comparação entre estes programas com o objetivo de se escolher o melhor, apenas serão apresentados as principais características de cada programa, para que o leitor interessado possa escolher, dentro de suas necessidades, o que melhor lhe convier.

O primeiro programa é o FreeFlow, que foi desenvolvido pelo grupo de Dinâmica de Fluidos Computacional do Laboratório de Computação de Alto Desempenho da Universidade de São Paulo em São Carlos. Este programa, na verdade, é um ambiente que permite a simulação de escoamento de fluidos Newtonianos, Não-Newtonianos, turbulentos e escoamento não isotérmicos. Ele é composto de quatro módulos: um modelador de moldes e escoamentos; um simulador de escoamento; um visualizador de escoamento e um reinicializador de escoamento. Suas principais caracteristicas são:

- trabalha em plataforma Unix;
- foi escrito em linguagem C;

- a criação e/ou modificação da geometria é feita por linhas de comando escrita em linguagem C, entretanto, já existe um banco de geometrias pronto, que pode ser acessado facilmente pelo programa;
- tem um ótimo e rápido programa de visualização do escoamento;
- os elementos da malha bidimensionais podem ser escolhidos entre triângulos ou quadriláteros;
- o programa gera, também, malhas tridimensionais.

Este programa não pode ser copiado diretamente da rede, entretanto, uma cópia do programa foi fornecida, gratuitamente, ao Departamento de Matemática Aplicada do Instituto de Matemática da Universidade de São Paulo.

O segundo programa analisado é o GMSH. Este programa é composto por quatro módulos básicos: Geometry, Mesh, Solver e Post-Processing. No próximo capítulo, é feita uma descrição mais detalha deste programa, que foi o escolhido para desenvolver este trabalho. Suas principais caracteristicas são:

- trabalha em plataforma Unix (i386, glibc 2.3 e acima), Mac OS X (10.3.9 e acima) e Windows (95/98/NT/2000/XP);
- a criação e/ou modificação da geometria é feita facilmente na tela, utilizando-se ferramentas CAD ou por intermédio de linhas de comandos.
- gera uma grande variedade de tipos de arquivos de saida que podem ser utilizado por outros programas.
- os elementos da malha bidimensional podem ser escolhidos entre triângulos ou quadriláteros;
- o programa gera, também, malhas tridimensionais em forma de tetraedro, hexaedro, prisma ou pirâmide.
- possui um tutorial online.
- possui um ótimo manual de referência.

Este programa pode ser copiado a partir do endereço:

http://geuz.org/gmsh/#Download.

O GMSH foi utilizado nos trabalhos de [20], que estuda um método adaptativo para encontrar aproximações para a viscosidade nas equações de Hamilton-Jacobi; no trabalho de [21], que modela um escoamento turbulento usando uma abordagem probabilistica; ou no trabalho de [22], que aplica técnicas computacionais de alto desempenho (processamento paralelo) para modelar o escoamento de àgua através do solo; ou ainda no trabalho de [23], que simula o comportamento de tecidos finos deformáveis para sistemas de treinamento cirúrgico.

1.3 Organização do Trabalho

Este trabalho foi dividido em capítulos, que estão assim estruturados:

No Capítulo 2, é apresentado o programa GMSH, que foi utilizado para construir a geometria do sólido e também para gerar uma malha triangular não estruturada sobre sua superfície. Este software é de acesso livre, gratuito, e é composto por quatro módulos, dois dos quais, foram utilizados neste trabalho: Geometry e Mesh. Este Capítulo não tem a pretensão de ser um manual de referência ou mesmo um tutorial sobre o software, entretanto, são apresentados três exemplos de construção de sólidos simples: um cubo vazado, uma esfera e um octaedro regular, que poderão ajudar, um principiante no estudo deste software, a construir geometrias mais complexas, como também gerar malhas sobre elas.

No Capítulo 3, se faz a descrição detalhada de todos os campos da estrutura de dados Halfedge, que é utilizada para armazenar as informações da geometria e da malha gerada sobre a superfície do sólido e também são mostrados os passos necessários ao seu preenchimento a partir dos dados gerados pelo GMSH.

No Capítulo 4, são apresentadas as fórmulas matemáticas e o método numérico utilizado na movimentação da fronteira imersa, como também, o algoritmo de refinamento de malhas adaptativas desenvolvido por Berger [9].

No Capítulo 5, são apresentadas os resultados numéricos deste trabalho, onde pode ser visto algumas cenas da movimentação da fronteira imersa e das malhas compostas utilizadas durante esta movimentação.

Finalmente, no Capítulo 6, são apresentadas as conclusões deste trabalho.

CAPÍTULO 2

O Gerador de Malhas GMSH

Neste Capítulo, apresenta-se o software GMSH, descrevendo os módulos que o compõe, além de mostrar, por intermédio de um exemplo, como é feita a construção da geometria de um sólido e de como é gerada uma malha triangular sobre sua superfície.

Na Seção 2.1, apresenta-se informações gerais sobre o software, sobre seus criadores e um exemplo de geometria mais complexa, que podem ser geradas pelo GMSH.

Na Seção 2.2, apresenta-se uma descrição sobre os quatro módulos que compõe o ambiente GMSH, incluindo seus pontos fortes e fracos.

Na Seção 2.3, mostra-se a construção, passo a passo, da geometria de um cubo vazado e da malha triangular sobre sua superfície, utilizando a meta-linguagem própria do GMSH.

2.1 Introdução

O GMSH é um software que permite: construir a geometria de um sólido; gerar uma malha não estruturada bidimensional ou tridimensional para este sólido; resolver equações diferenciais parciais que envolvam este sólido e fazer a apresentação gráfica dos resultados das etapas anteriores. A versão utilizada neste trabalho é a 1.60 de 15 de Março de 2005.

Este software foi criado por Christophe Geuzaine, da Case Western Reserve University, em Cleveland, Ohio e Jean-François Remacle, da Catholic University of Louvain, em Louvain, na Bélgica e pode ser copiado, gratuitamente, a partir do endereço:

http://geuz.org/gmsh/#Download

O software tem versões para os sistemas operacionais: Unix (i386, glibc 2.3 e acima), Mac OS X (10.3.9 e acima) e Windows (95/98/NT/2000/XP).

Finalmente, como forma de mostrar que geometrias mais complexas do que as utilizadas neste trabalho podem ser construídos pelo GMSH, apresenta-se a Figura 2.1 elaborada por C. Geuzaine. Esta figura, bem como muitas outras, podem ser encontradas para download em:

http://www.geuz.org/gmsh/.

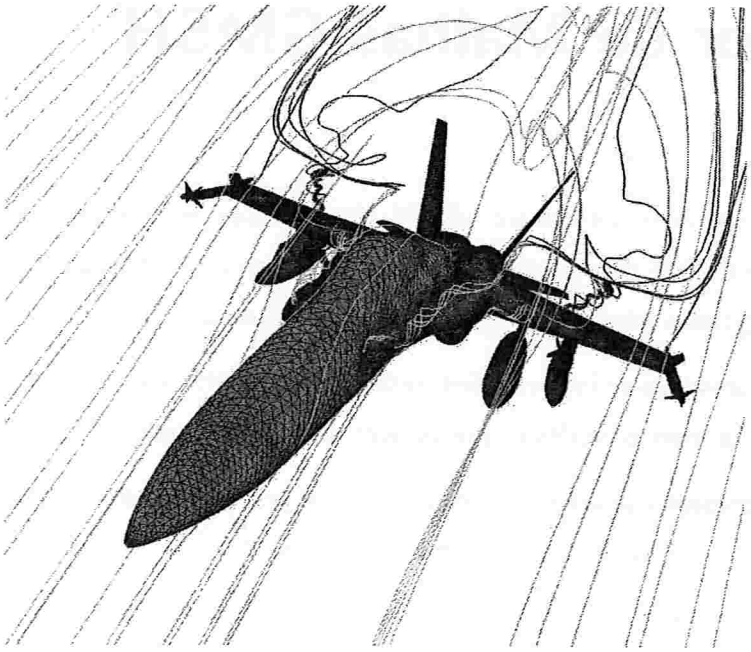


Figura 2.1: Geometria e malha do avião F18 feito por C. Geuzaine

2.2 Descrição do Software

O GMSH é composto por quatro módulos: Geometry, Mesh, Solver e Post-Processing.

No módulo *Geometry* são definas todas as *entidades geométricas* do sólido, tais como: curvas orientadas (e.g. segmentos de retas, círculos, elipses), superfícies orientadas (e.g. planos, superficies de rotação) e volumes. O traçado da geometria do sólido pode ser feito de duas maneiras: a primeira, utilizando uma interface gráfica do tipo CAD (Computer Aided Design) que acompanha o software e que permite traçar, a partir do teclado e do mouse, o contorno do sólido desejado; a segunda, utilizando uma meta-linguagem de programação do

próprio GMSH, a qual, por linhas de comando, permite fornecer todas as entidades geométricas do sólido, bem como todas as suas inter-relações de forma a se obter o traçado final do sólido.

No módulo *Mesh* é gerada uma malha não estruturada. Esta malha representa, na verdade, um conjunto do espaço tridimensional de elementos geométricos elementares (retas, triângulos e tetraedros). Toda malha produzida pelo GMSH é não estruturada, ou seja, as entidades geométricas são definidos apenas por uma lista de vértices, sem que exista uma relação de ordem pré-definida entre elas. Na geração das malhas, as curvas são discretizadas primeiro. As malhas das linhas são utilizadas para a construção das malhas das superfícies que, por sua vez, são utilizadas para a construção das malhas dos volumes. Neste trabalho, usamos apenas malhas bidimensionais, as quais são construídas sobre a superfície do sólido.

No módulo *Solver* são escritos e executados os programas empregados para resolver as equações diferenciais. Existe um exemplo escrito na linguagem C++ e que está disponível no endereço:

http://geuz.org/gmsh/doc/textinfo/gmsh_6.html/#SEC40.

Programas externos, podem ser fornecidos pelo próprio usuário numa das linguagens: C, C++, PERL e PYTHON e podem ser acoplados a este módulo, permitindo que sejam feitas simulações numéricas cujos resultados podem ser utilizados no módulo de Post-Processing. Maiores detalhes sobre o módulo Solver, podem ser encontrados no endereço:

http://www.geuz.org/getdp/.

O módulo *Post-Processing* permite que múltiplos pós-processamentos de quantidades escalares, vetoriais ou mapas tensoriais, possam ser carregados e manipulados individual ou globalmente. As funções de pós-processamentos incluem, mas não se restringem, às seções transversais, fronteiras, mapas de cores e animações.

No Manual de Referência do GMSH são apresentados os seus pontos fortes e fracos do seu atual estágio de desenvolvimento. É importante ressaltar que, aqui, não foi realizada qualquer avaliação destes pontos, pois, além de fugir ao escopo da proposta do trabalho, o software não foi utilizado nem de forma intensiva nem de forma geral.

Apresenta-se um breve resumo dos principais pontos fortes do GMSH:

- O GMSH tem uma interface gráfica simples e amigável, a qual pode ser configurada pelo usuário.

- Existe uma farta documentação e um ótimo tutorial disponível.
- Encontra-se versões para os sistemas operacionais Unix (i386, glibc 2.3 e acima), Mac OS X (10.3.9 e acima) e Windows (95/98/NT/2000/XP).
- O software fornece mecanismos para controlar os elementos da malha final, com grande acurácia, por intermédio de interpolações e de parâmetros de controle definidos pelo próprio usuário.
- O GMSH oferece uma gama variada de possibilidade para visualizações computacionais, podendo mostrar dados escalares, vetorias e tensoriais e ainda gerar animações.
- O software exporta arquivos em diversos formatos: Vector PostScript, LaTex, PS, JPEG, PNG, entre outros.
- Permite fácil integração de programas externos, no módulo *Solver* fornecidos pelo usuário.

Como pontos fracos do GMSH, segundo consta em seu manual de referência, podemos citar resumidamente:

- O método Botton-up (método de escolha de uma opção entre várias possíveis) para descrever a geometria do sólido pode ser bastante inconveniente para modelos complexos.
- Tem recursos limitados para a construções de malhas em superficies muitos recortadas e com cantos.
- Não é um gerador de malhas multi-blocos, ou seja, todas as malhas produzidas pelo GMSH são construídas com base em malhas não estruturadas.
- O GMSH foi construído para resolver problemas acadêmicos e não para resolver grandes problemas industriais. Ele pode se mostrar muito lento para resolver grandes problemas que contenham milhares de elementos geométricos.

A experiência mostra que o Tutorial e o Manual de Referência que acompanham o GMSH, são de suma importância no aprendizado. O Tutorial apresenta um exemplo de construção de um sólido simples e de como é feito a sua malha não estruturada. O Manual de Referência inclui uma completa descrição de todos os comandos utilizados pelo GMSH. Ambos podem ser encontrado para download em:

http://www.geuz.org/gmsh/#Documentation

Finalmente, é importante ressaltar que o grupo do Laboratório de Transferência de Calor e Massa e Dinâmica dos Fluidos da Faculdade de Engenharia Mecânica da Universidade Federal de Uberlândia está desenvolvendo um Tutorial do GMSH em português. A versão deste Tutorial pode ser encontrado para download em:

http://www.ltcm.mecanica.ufu.br/files/LTCM_file00106.pdf

2.3 Exemplo de Aplicação

Para melhor compreender a utilização dos módulos *Geometry* e *Mesh* que foram utilizados neste trabalho, ilustra-se primeiro a elaboração da geometria do sólido; a seguir a construção de sua malha triangular e, finalmente, apresentam-se os arquivos de dados e de saidas gerados pelo GMSH.

A título de ilustração de uso do GMSH, será explicado como é feita a construção do cubo vazado de arestas externas unitárias e de arestas internas de dimensão 0,75, mostrado na Figura 2.2.

Qualquer editor de texto pode ser empregado para se escrever o arquivo contendo os comandos da meta-linguagem que o GMSH utiliza para descrever o sólido. São estes os seguintes passos para descrever a geometria do sólido:

1. Os primeiros entes geométricos do sólido a serem criados são os vértices. Estes entes geométricos têm dimensão zero. Para isto, utiliza-se o comando **Point(a)={b,c,d,lc}**, onde:

   - **a** é o identificador numérico do vértice.
   - **b, c, d** são as coordenadas cartesianas do vértice nas direções x, y e z.
   - **lc** é o tamanho máximo permitido para o lado da malha, neste trabalho, usou-se a malha triangular. Este parâmetro define, indiretamente, o número de triângulos da malha: quanto menor for o valor de **lc**, maior será o número de triângulos gerados.

   Todos os parâmetros **a, b, c, d** e **lc** são dados de entrada definidos pelo usuário. Comentários são permitidos e devem ser precedidos por uma barra dupla, **//**, e toda

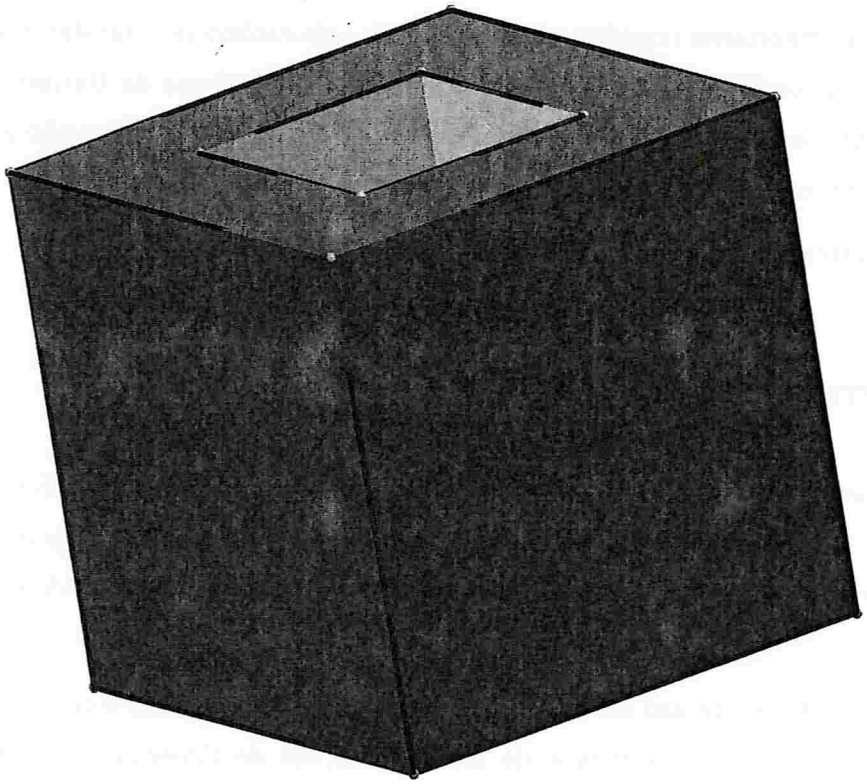


Figura 2.2: Cubo Vazado.

linha de comando deve finalizar com ponto e vírgula **;**.

Para o cubo vazado foram criados oito vértices externos,

Point(1) = {0,0,0,lc};

Point(2) = {1,0,0,lc};

Point(3) = {1,1,0,lc};

Point(4) = {0,1,0,lc};

Point(5) = {0,0,1,lc};

Point(6) = {1,0,1,lc};

Point(7) = {1,1,1,lc};

Point(8) = {0,1,1,lc};

e oito vértices internos,

Point(9) = {0.25,0.25,0,lc};

Point(10) = {0.75,0.25,0,lc};

Point(11) = {0.75,0.75,0,lc};

Point(12) = {0.25,0.75,0,lc};

Point(13) = {0.25,0.25,1,lc};

Point(14) = {0.75,0.25,1,lc};

Point(15) = {0.75,0.75,1,lc};

Point(16) = {0.25,0.75,1,lc};

onde lc=0.5 .

2. Os próximos entes geométricos do sólido a serem criados são as curvas. Estes entes geométricos têm dimensão um. No exemplo aqui estudado, as curvas são segmentos de reta. Para isto, utiliza-se o comando **Line(a)={b,c}**, onde:

   - **a** é o identificador numérico da curva.
   - **b** é o identificador numérico do vértice inicial da curva.
   - **c** é o identificador numérico do vértice final da curva.

   Todos os parâmetros **a, b** e **c** são definidos pelo usuário. Foram criados doze segmentos de retas para representar as doze arestas externas do cubo vazado,

   Line(1) = {1,2};

   Line(2) = {2,3};

   Line(3) = {3,4};

   Line(4) = {4,1};

   Line(5) = {1,5};

   Line(6) = {5,6};

   Line(7) = {6,2};

   Line(8) = {5,8};

   Line(9) = {8,7};

Line(10) = {7,6};

Line(11) = {3,7};

Line(12) = {4,8};

e doze segmentos de retas para representar as doze arestas internas.

Line(13) = {9,10};

Line(14) = {10,11};

Line(15) = {11,12};

Line(16) = {12,9};

Line(17) = {9,13};

Line(18) = {13,14};

Line(19) = {14,10};

Line(20) = {13,16};

Line(21) = {16,15};

Line(22) = {15,14};

Line(23) = {11,15};

Line(24) = {12,16};

3. Os entes geométricos do sólido a serem criados agora, são as faces planas. Estes entes geométricos têm dimensão dois. Para isto, são utilizados dois comandos. O primeiro comando, **Line Loop(a)={b,c,d,e}**, que tem como objetivo indicar o sentido de percurso de todas as curvas que formam uma dada face; o segundo comando, **Plane Surface(f)={g}**, que tem como objetivo identificar a face que é criada pelo primeiro comando **Line Loop(a)={b,c,d,e}**. Neste comando:

   - **a** é o identificador numérico do loop que formará a superfície do sólido.
   - **b, c, d, e** são os identificadores numéricos das curvas.

   Todos os parâmetros **a, b, c, d** e **e** são definidos pelo usuário.

   Os valores **b, c, d** e **e** podem vir acompanhados do sinal negativo, isto significará que a curva está sendo percorrida no sentido contrário à aquele originalmente definido na etapa anterior.

Na definição do comando **Line Loop(a)={b,c,d,e}**, apesar de se estar apresentando apenas quatro curvas **b, c, d** e **e**, podem ser definadas tantas curvas quantas sejam necessárias para a definição do loop.

O comando **Plane Surface(f)={g}** é:

- **f** é o identificador numérico da superfície do sólido que está sendo criada.
- **g** é o identificador numérico de loop que cria a superfície.

Os parâmetros **f** e **g** são definidos pelo usuário.

Na definição do comando **Plane Surface(f)={g}**, apesar de se indicar apenas o loop **g**, outros loops podem ser usados para definir uma superfície. Isto seria o caso de uma superfícies que tenha "buracos" em seu interior.

Foram criados as quatro faces verticais externas do cubo vazado,

Line Loop(25) = {22,19,14,23}; Plane Surface(26) = {25};

Line Loop(27) = {21,-23,15,24}; Plane Surface(28) = {27};

Line Loop(29) = {20,-24,16,17}; Plane Surface(30) = {29};

Line Loop(31) = {18,19,-13,17}; Plane Surface(32) = {31};

quatro faces verticais internas,

Line Loop(33) = {10,7,2,11}; Plane Surface(34) = {33};

Line Loop(35) = {9,-11,3,12}; Plane Surface(36) = {35};

Line Loop(37) = {8,-12,4,5}; Plane Surface(38) = {37};

Line Loop(39) = {6,7,-1,5}; Plane Surface(40) = {39};

e duas faces horizontais, a superior e a inferior, respectivamente.

Line Loop(41) = {22,-18,20,21}; Line Loop(42) = {10,-6,8,9};

Plane Surface(43) = {42,41};

Line Loop(44) = {14,15,16,13}; Line Loop(45) = {2,3,4,1};

Plane Surface(46) = {45,44};

Note que neste caso foram necessários dois loops para definir cada uma das faces. O primeiro loop é o que define a borda interna da face e o outro define a borda externa.

Ao se escrever o comando **Line Loop(a)={b,c,d,e}** deve-se, obrigatoriamente, escolhe-se o sentido de percurso das curvas **b, c, d** e **e**, de tal forma a deixar todos os vetores normais à cada face, sempre apontando para fora (ou para dentro) do sólido.

4. O quarto e último ente geométrico a ser criado é o volume. Este ente geométrico tem dimensão três. Para isto, foram utilizados dois comandos. O primeiro comando, **Surface Loop(a)={b,c,d,e}**, tem como objetivo, indicar o sentido de percurso de um conjunto de faces do sólido, e o segundo comando, **Volume(f)={g}**, tem como objetivo, identificar todos os conjuntos de faces que compõem o sólido. No comando **Surface Loop(a)={b,c,d,e}**:

   - **a** é o identificador numérico de um conjunto específico de faces.
   - **b , c, d, e** são os identificadores numéricos das faces.

   Todos os parâmetros **a, b, c, d** e **e** são definidos pelo usuário e apesar de se indicar apenas quatro faces, outras faces podem ser acrescentadas conforme a necessidade.

   No comando **Volume(f)={g}**:

   - **f** é o identificador númerico do sólido que está sendo criado.
   - **g** é o identificador numérico do conjunto de faces que cria a sólido.

   Os parâmetros **f** e **g** são definidos pelo usuário. Na definição do comando **Volume(f)={g}**, apesar de se estar criando apenas um conjunto de faces, no caso **g**, outros conjuntos de faces podem ser acrescentados caso seja necessário.

   O sólido é criado a partir de um único conjunto formado por dez faces:

   Surface Loop(47) = {26,28,30,32,34,36,38,40,43,46}; Volume(48)={47};

Para visualizar a geometria do sólido, vide Figura 2.3, deve-se acessar o software GMSH e carregar o arquivo com a geometria do sólido. Este código de software pode ser encontrado no Apêndice A.

Para se gerar a malha sobre a superfície do cubo vazado, utilizamos o módulo *Mesh* com a opção: malha *2D*. As malhas triangulares geradas podem ser visualizadas, tanto na Figura 2.4, na qual utiliza-se o parâmetro lc=0.5; quanto na Figura 2.5, na qual utiliza-se o paramêtro lc=0.1.

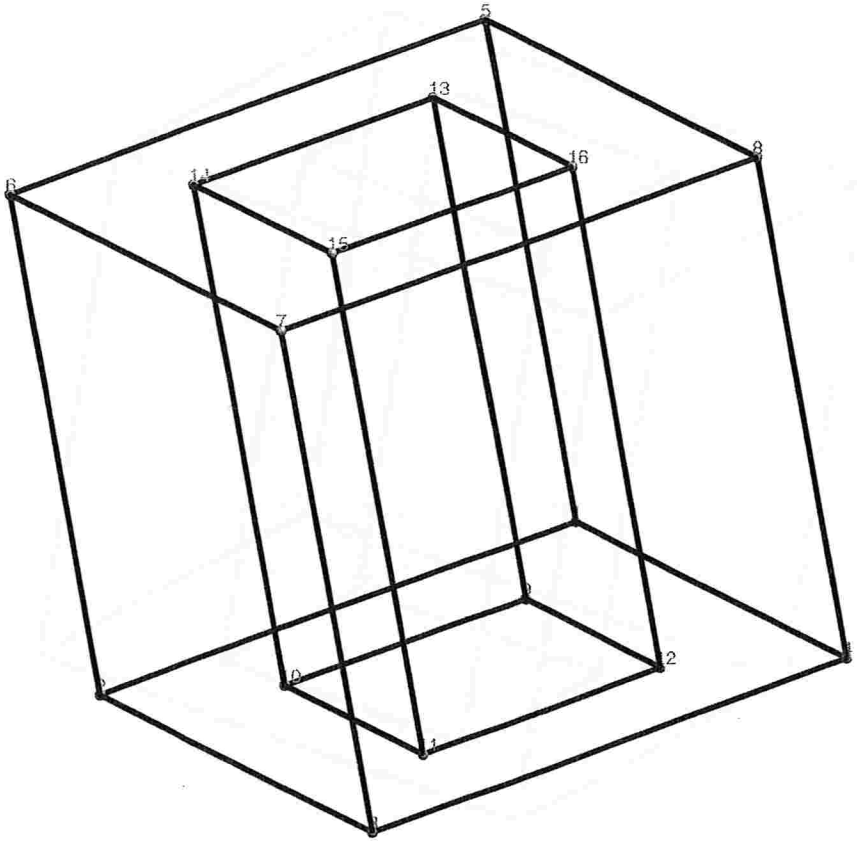


Figura 2.3: Geometria do Cubo Vazado.

A título de ilustração, apresenta-se, no Apêndice B, o conteúdo do arquivo com os dados da malha triangular (lc=0.5), gerado pelo GMSH. Este arquivo é utilizado para alimentar a estrutura de dados Halfedge, que será descrita no Capítulo 3, e seu conteúdo, em resumo, apresenta:

- Todos os vértices com suas respectivas coordenadas cartesianas.
- Todos os triângulos da malha representados por seus três vértices.
- Ao percorrermos os vértices de cada triângulo na ordem com que foram gerados pelo GMSH, teremos o vetor normal, a cada uma das faces do triângulo apontando para o exterior do sólido.

Finalmente, apresenta-se no Apêndice A, três sólidos: Cubo Vazado, Esfera e Octaedro e

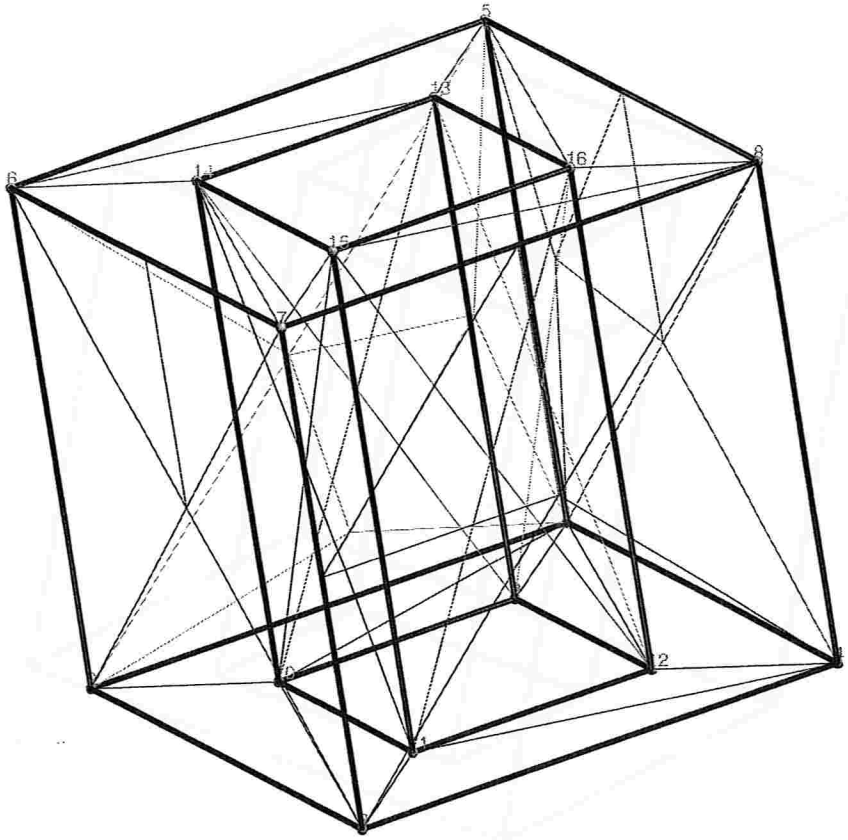


Figura 2.4: Malha do Cubo Vazado, lc = 0.5; Triângulos Gerados = 62.

as malhas triangulares sobre suas superfícies; além dos respectivos programas utilizados para gerar a geometria de cada sólido, escritos na meta-linguagem do GMSH.

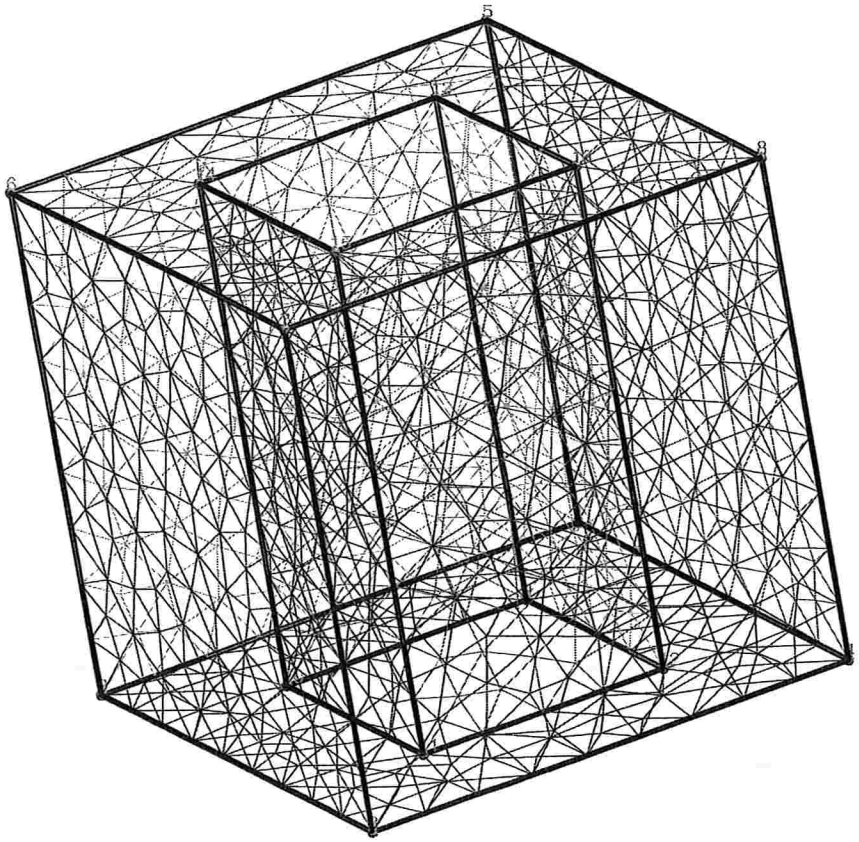


Figura 2.5: Malha do Cubo Vazado, lc = 0.1; Triângulos Gerados = 1650.

CAPÍTULO 3

Estrutura de Dados Halfedge

Neste Capítulo,descreve-se, com detalhes, a estrutura de dados Halfedge, a qual é usada neste trabalho para representar um sólido através se sua superfície e mostra-se, também, como é feita a inicialização desta estrutura a partir dos dados de saída do GMSH.

Na Seção 3.1, apresenta-se um breve histórico sobre modelagem de sólidos e a conceituação da estrutura de dados Halfedge.

Na Seção 3.2, é feita uma descrição de todos os campos que compõem a estrutura de dados Halfedge.

Na Seção 3.3, mostram-se os passos necessários à inicialização da estrutura de dados Halfedge, a partir dos dados de saída do GMSH.

3.1 Modelo Computacional de um Sólido

A modelagem de sólidosé uma área da modelagem geométrica que estuda a construção e a representação computacional de sólidos. Esta representação deve permitir que, de forma rápida e precisa, se obtenha tanto *informações geométricas* quanto *informações topológicas* sobre o sólido.

As informações geométricas são aquelas que permitem estudar como o sólido se apresenta dentro do espaço que o envolve. Tais informações permitem saber, por exemplo, o número de vértices ou de arestas ou de faces do sólido, a distância entre dois vértices, ou ainda, o ângulo entre duas arestas.

As informações topológicas são aquelas que permitem estudar as relações de conectividade e adjacência entre os vários elementos de um sólido. Tais informações permitem saber, por exemplo, quais vértices são vizinhos a um dado vértice, quais arestas são comuns a um dado vértice, ou ainda, quais faces são adjacentes a uma dada aresta.

Os modelos que são usados para representar um sólido podem ser classificados em três grupos, chamados de: modelos de decomposição, modelos de construção e modelos de fronteira.

Os *modelos de decomposição*, fazem uso de primitivas básicas e o sólido é descrito por intermédio de operações de colagem.

Os *modelos de construção* fazem uso de primitivas básicas mais elaboradas que as dos modelos de decomposição e os sólidos são descritos por intermédio de operações mais gerais do que simplesmente a colagem, como por exemplo, a união, a interseção e a diferença de conjuntos.

Os *modelos de fronteira* fazem uso de hierarquia e os sólidos são descritos por intermédio de seu contorno. Esta hierarquia pode ser entendida da seguinte forma: a fronteira de um sólido tridimensional é formada por um conjunto de faces, que são bidimensionais. Cada face, é representada em termos das curvas unidimensionais que representam sua fronteira. Cada uma destas curvas pode ser representada por alguns de seus pontos relevantes. Os modelos de fronteira são amplamente utilizados em computação gráfica e serviu de base para este trabalho. Para uma abordagem mais completa sobre o assunto consultar [1].

Conforme foi citado anteriormente, a representação do sólido por intermédio do modelo de fronteira é feita de forma hierárquica. No primeiro nível hierárquico, se encontra o conjunto de todas as faces que compõem o sólido. As faces são entes de dimensão 2 do $\mathbb{R}^3$ e estão situadas sobre superfícies planas, quádraticas ou toroidais, parametrizadas ou não.

No segundo nível hierárquico, encontram-se as arestas que são as curvas, parametrizadas ou não, que representam as fronteiras externa e/ou interna de cada uma das faces. As arestas são entes de dimensão um do $\mathbb{R}^3$.

No terceiro e último nível hierárquico, encontram-se os vértices que são entes de dimensão zero do $\mathbb{R}^3$ e estão situados, geralmente, nos extremos de cada uma das arestas.

Os principais tipos de modelos computacionais de fronteira, são baseados em:

- *polígonos*: utiliza uma lista de faces;
- *vértices*: utiliza uma lista de vértices;
- *arestas*: utilizada a estrutura de dados *Winged Edge* ou *Halfedge*, sendo esta última, uma variante da primeira.

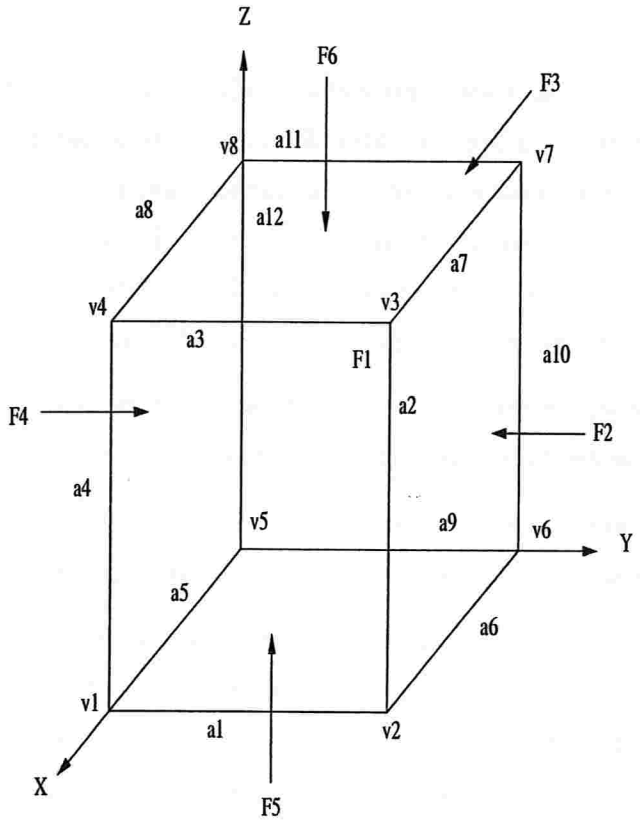
Neste trabalho, adotou-se para representar o sólido, o modelo de fronteira baseado em arestas Halfedge, por sua rapidez na obtenção das informações topológicas e geométricas do sólido e ser amplamente utilizada em computação gráfica para representar superfícies orientáveis. Uma superfície é orientável, se o seu campo de vetores normais for contínuo. É importante ressaltar que, esta estrutura de dados não permite que se represente sólidos, cuja superfície não seja orientável, por exemplo, a faixa de Möbius.

Todas as informações geométricas e topológicas do sólido são obtidas por intermédio do inter-relacionamento destes três entes: faces, arestas e vértices.

Para um melhor entendimento do assunto que será tratado a seguir, algumas definições básicas sobre grafos e sobre poliedros são apresentadas no Apêndice D.

A fronteira de um poliedro convexo tem a mesma estrutura de um grafo planar. Este grafo permite representar todas as relações de adjacências entre os vértices, arestas e faces deste poliedro. As estruturas de dados que descrevem este grafo planar são chamadas de estruturas de dados topológica (para maiores detalhes consulte [3]). A estrutura Halfedge, é uma estrutura de dados topológica por excelência, ela permite que consultas sobre as relações de adjacência entre os elementos do poliedro sejam respondidas em tempo constante por tipo de informação requerida, por exemplo, suponha que dez arestas cheguem ao vértice A e que cinco arestas cheguem ao vértice B, se uma consulta hipotética fosse feita sobre quantas arestas chegam a cada um destes dois vértices, o tempo de processamento gasto para se chegar à resposta sobre o vértice A, será, aproximadamente, o dobro do tempo do vértice B, pois o número de arestas que chegam ao vértice A é o dobro das que chegam ao vértice B.

Apresenta-se na Figura 3.1, uma relação de vértices, de arestas e de faces de um cubo de aresta unitária. Pode-se observar, por exemplo, que o vértice $v1$ tem coordenadas cartesianas (1,0,0), que a aresta $a1$ se incia no vértice $v1$ e termina no vértice $v2$ e que a face $F1$ é formada pelos vértices $v1$, $v2$ $v3$ e $v4$. Na Figura 3.2, são mostrados dois grafos planares que representam o cubo da Figura 3.1. É importante observar que, todas as faces, arestas, vértices



Vertices		Arestas			Faces	
v1: (1,0,0)	v5: (0,0,0)	a1: v1v2	a2: v2v3	a3: v3v4	F1: v1v2v3v4	F2 :v2v6v7v3
v2: (1,1,0)	v6: (0,1,0)	a4: v4v1	a5: v5v1	a6: v2v6	F3: v6v5v8v7	F4 :v5v1v4v8
v3: (1,1,1)	v7: (0,1,1)	a7: v7v3	a8: v4v8	a9: v6v5	F5: v1v5v6v2	F6 :v4v3v7v8
v4: (1,0,1)	v8: (0,0,1)	a10: v6v7	a11: v8v7	a12: v5v8		

Figura 3.1: Cubo Unitário.

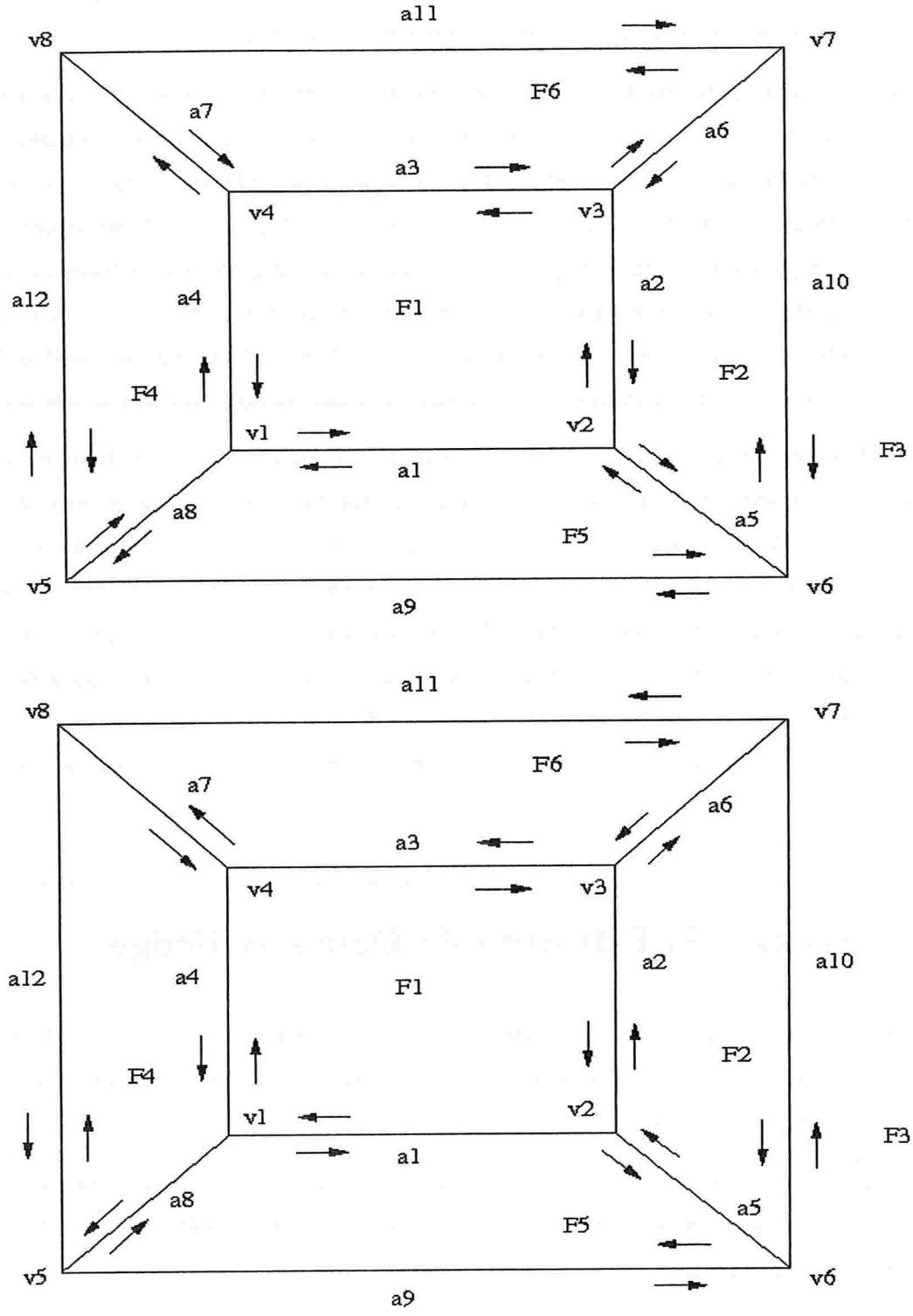


Figura 3.2: Grafo Planar do Cubo: Todo vetor normal à face aponta para fora do cubo (na parte superior); Todo vetor normal à face aponta para dentro do cubo (na parte inferior).

e as relações de adjacências do cubo estão representadas e preservadas pelos grafos planares. A face $F3$ do cubo é representada pela região externa aos grafos.

A grande eficiência da estutura Halfedge está na representação adequada dos ciclos de arestas em torno de cada face e de cada vértice e do fato do grafo planar ser orientável. Ao se percorrer a fronteiras de todas as faces do grafo planar no sentido anti-horário (ou horário), cada aresta será percorrida duas vezes, uma em cada sentido [4]. Isto pode ser observado em qualquer um dos grafos planares da Figura 3.2, por exemplo, no grafo planar superior, a aresta $a3$ é percorrida duas e somente duas vezes; uma no sentido dos vértices $v3 \rightarrow v4$, quando percorrida dentro da face $F1$ e uma no sentido $v4 \rightarrow v3$, quando percorrida dentro da face $F6$. O mesmo ocorre para todas as outras arestas, inclusive as do grafo planar inferior.

Uma outra constatação que pode ser feita a partir da Figura 3.2. O sentido de percurso das arestas do primeiro grafo planar foi escolhido de forma a deixar o vetor normal à cada superfície apontando sempre para fora do cubo, por exemplo, ao percorremos a face $F1$ no sentido dos vértices $v1 \rightarrow v2 \rightarrow v3 \rightarrow v4$, o vetor normal apontará para fora da face $F1$ (regra da mão direita), isto ocorre para todas as outras faces; no segundo grafo, o sentido de percurso foi escolhido de forma a deixar o vetor normal apontando sempre para dentro do cubo, por exemplo, ao percorremos a face $F1$ no sentido dos vértices $v4 \rightarrow v3 \rightarrow v2 \rightarrow v1$, o vetor normal apontará para dentro da face $F1$ (regra da mão direita), isto ocorre para todas as outras faces.

3.2 Descrição da Estrutura de Dados Halfedge

No presente trabalho, partiu-se da estrutura de dados Halfedge definida por [1], que decompõem o sólido em seis níveis hierárquicos, aqui chamados de *nós*. São eles: *Solid*, *Face*, *Loop*, *Edge*, *Halfedge* e *Vertex*.

A esta estrutura básica, foram acrescentados outros campos necessários ao desenvolvimento do trabalho. A relação completa de todos os campos da estrutura Halfedge, em FORTRAN 90, é apresentada no Apêndice C.

A estrutura de dados Halfedge é um estrutura que faz uso intensivo de ponteiros e, visando facilitar sua compreensão, apresenta-se, na Figura 3.3, as principais relações existentes, não todas, entre os diversos ponteiros. Faz-se agora, uma descrição mais detalhada de cada um

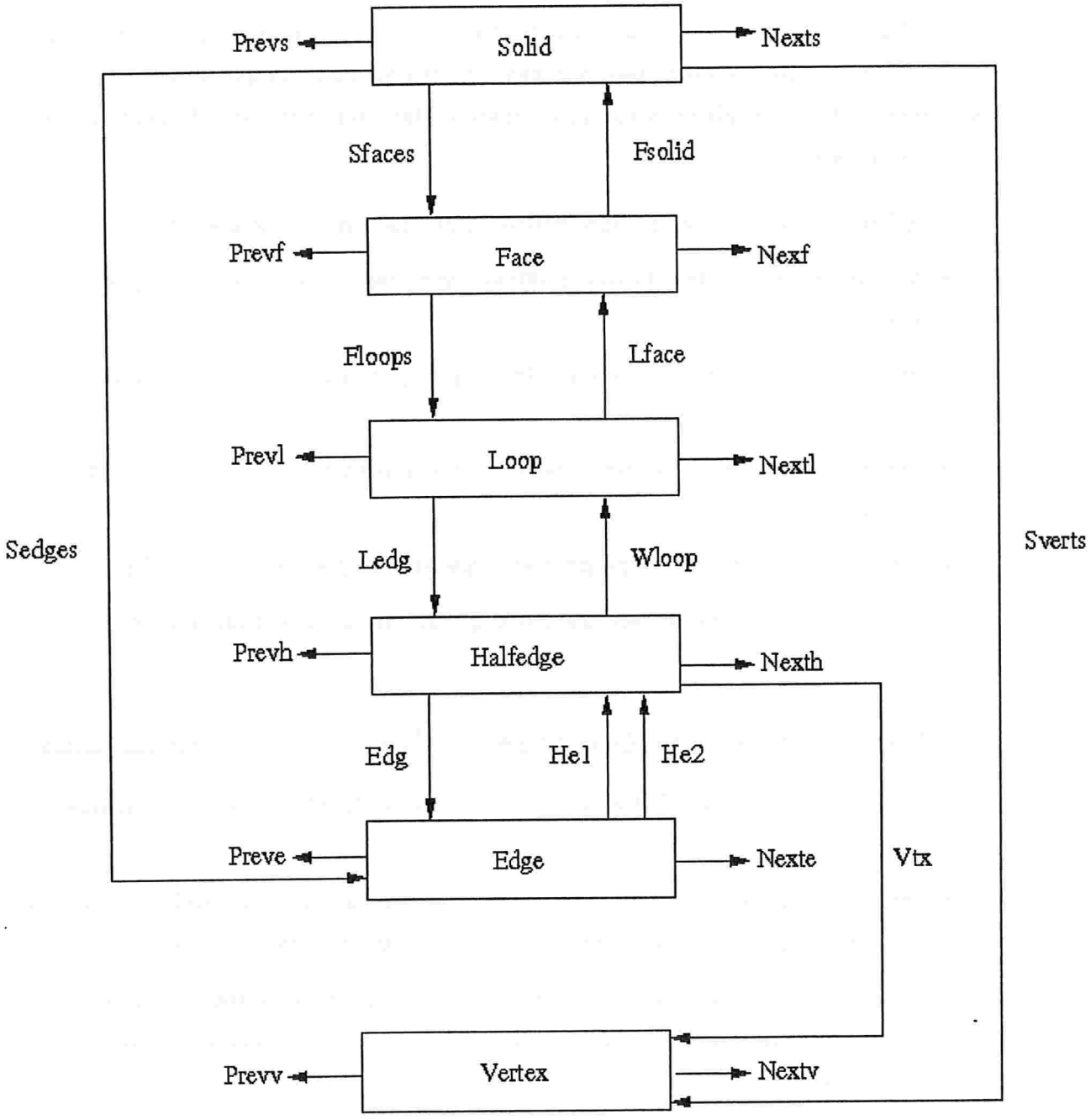


Figura 3.3: Diagrama Esquemática da Estrutura Halfedge.

dos seis nós da estrutura Halfedge e, sempre que necessário, a Figura 3.3 deve ser consultada.

1. Nó **Solid**

   O nó **Solid** é o nó raiz da estrutura Halfedge. Este nó permite que se represente vários sólidos diferentes num mesmo instante de tempo ou ainda que, um mesmo sólido seja representado em vários instantes de tempo diferentes. Este nó é formado por seis campos, a saber:

   - **IdSolid**: É um campo, do tipo inteiro, usado para identificar o sólido.
   - **Sfaces**: É um campo, do tipo ponteiro, que aponta para uma face qualquer do sólido.
   - **Sedges**: É um campo, do tipo ponteiro, que aponta para uma aresta qualquer do sólido.
   - **Sverts**: É um campo, do tipo ponteiro, que aponta para um vértice qualquer do sólido.
   - **Nexts**: É um campo, do tipo ponteiro, que aponta para o próximo sólido.
   - **Prevs**: É um campo, do tipo ponteiro, que aponta para o sólido anterior.

2. Nó **Face**

   O nó **Face** representa uma face plana qualquer do sólido, sendo formado por sete campos:

   - **IdFace**: É um campo, do tipo inteiro, usado para identificar uma face qualquer do sólido.
   - **Fsolid**: É um campo, do tipo ponteiro, que aponta para o nó **Solid**. Cada face deve ter um ponteiro que aponte para o sólido, do qual ela faça parte.
   - **Flout**: É um campo, do tipo ponteiro, que aponta para o nó **Loop**. Cada face deve ter um ponteiro que aponte para qualquer uma das curvas poligonais que formem sua fronteira externa.
   - **Floops**: É um campo, do tipo ponteiro, que aponta para o nó **Loop**. Cada face deve ter um ponteiro que aponte para qualquer uma das curvas poligonais que formem sua fronteira interna. Este campo só será usado, caso a face possua um ou mais "buracos".

- **Feq**: É um campo vetorial com quatro posições (a,b,c,d), cada uma delas é do tipo real com dupla precisão, sendo utilizadas para armazenar a equação do plano que contêm a face do sólido: ax+by+cz+d=0.
- **Nextf**: É um campo, do tipo ponteiro, que aponta para o próxima face do sólido.
- **Prevf**: É um campo, do tipo ponteiro, que aponta para o face anterior do sólido.

3. Nó **Loop**

   O nó **Loop** estabelece um tipo de conectividade entre as fronteiras externa e interna de uma face do sólido. Este nó é formado por quatro campos:

   - **Ledg**: É um campo, do tipo ponteiro, que aponta para um nó **Halfedge**. Cada nó **Loop** deve ter um ponteiro que aponte para qualquer um dos segmentos de curva orientado que pertença a uma das fronteiras da face.
   - **Lface**: É um campo, do tipo ponteiro, que aponta para o nó **Face**. Todas as curvas poligonais que formam a fronteira externa e interna de uma face, devem ter um ponteiro que aponte para esta face.
   - **Nextl**: É um campo, do tipo ponteiro, que aponta para o próximo **loop** dentro da mesma face do sólido.
   - **Prevl**: É um campo, do tipo ponteiro, que aponta para o nó **loop** anterior dentro da mesma face do sólido.

   Note que, o acesso às curvas poligonais, que representam a fronteira externa e interna de uma face, deve ser feito, respectivamente, pelos campos **Flout** e **Floops** do nó **Face**.

   A lista ligada formada pelos ponteiros **Nextl** e **Prevl** permite, apenas, que se caminhem ao longo das curvas poligonais, que representam as fronteiras internas da face. Como a malha gerada pelo GMSH é formada por triângulos e portanto não apresenta buracos, estes campos, na prática, não são utilizados.

4. Nó **Halfedge**

   Intuitivamente, uma Halfedge representa um segmento de curva orientado que pertence à fronteira da face do sólido. O que se quer dizer com segmento de curva orientado é que, um segmento de curva que liga dois pontos A e B pode ser percorrido em dois sentidos. A cada um destes sentidos de percurso, associa-se um nó **Halfedge**. Este nó é composto por cinco campos:

- **Edg**: É um campo, do tipo ponteiro, que aponta para um nó **Edge**. A cada nó **Edg** devem estar associados dois e somente dois nós **Halfedge**.
- **Vtx**: É um campo, do tipo ponteiro, que aponta para o vértice final da Halfedge. O vértice final de uma Halfedge é considerado o vértice inicial da próxima Halfedge dentro do mesmo loop.
- **Wloop**: É um campo, do tipo ponteiro, que aponta para um nó **Loop**, isto é, todas as Halfedges que formam a fronteira de uma dada face devem apontar para o loop que define esta face.
- **Nexth**: É um campo, do tipo ponteiro, que aponta para o próximo nó **Halfedge** dentro do mesmo loop.
- **Prevh**: É um campo, do tipo ponteiro, que aponta para o nó **Halfedge** anterior dentro do mesmo loop.

5. Nó **Edge**

   Intuitivamente, o nó **Edge** cola as duas Halfedges, que estão associadas aos dois sentidos de percurso de um mesmo segmento de curva orientada. Este segmento de curva, representado pelo nó **Edge**, é chamado de aresta e é composto por cinco campos:

   - **IdEdge**: É um campo, do tipo inteiro, usado para identificar uma aresta.
   - **He1**: É um campo, do tipo ponteiro, que aponta para o nó **Halfedge** que representa um dos sentidos de percurso do segmento de curva orientado. Vide Figura 3.4.
   - **He2**: É um campo, do tipo ponteiro, que aponta para o nó **Halfedge** que representa o sentido oposto ao percurso de He1. Vide Figura 3.4.
   - **Nexte**: É um campo, do tipo ponteiro, que aponta para a próxima aresta.
   - **Preve**: É um campo, do tipo ponteiro, que aponta para a aresta anterior.

6. Nó **Vertex**

   O nó **Vertex** representa, intuitivamente, um ponto do $\mathbb{R}^3$. Este nó é composto por nove campos, sendo que quatro deles: **Vcoord2**, **Velo1**, **Velo2** e **MapLagran** foram introduzidos na estrutura Halfedge originalmente definida por [1]. A descrição de cada um destes campos é feita a seguir:

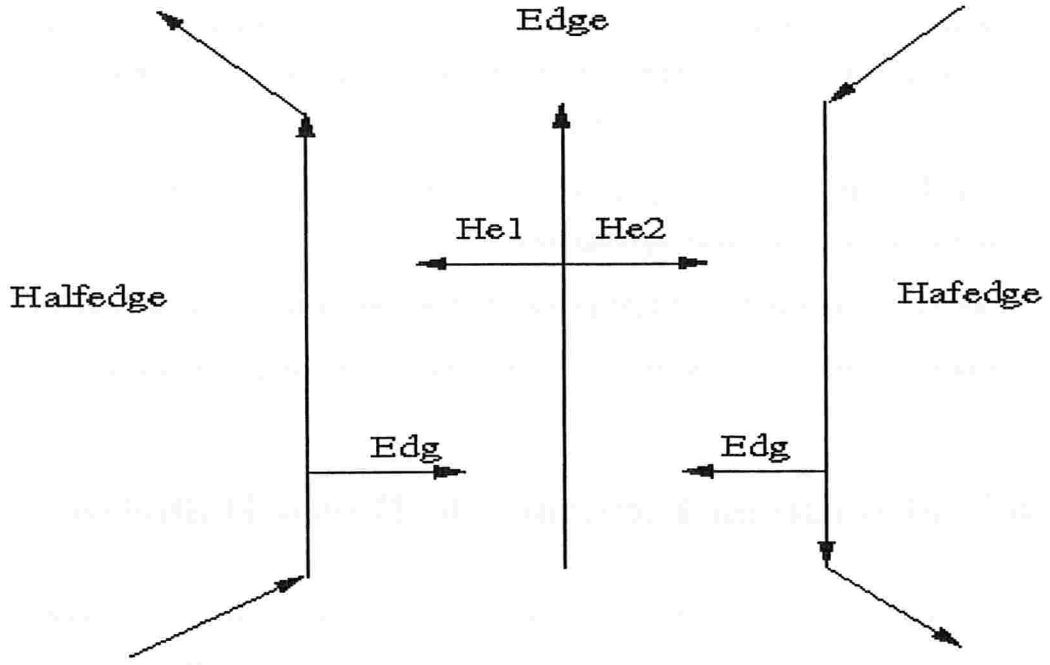


Figura 3.4: Ciclo de Aresta.

- **IdVertex**: É um campo, do tipo inteiro, usado para identificar um vértice.
- **Vedge**: É um campo do tipo ponteiro que aponta para um nó do tipo **Halfedge**. Apesar de podermos ter várias Halfedeges começando em um mesmo vértice, basta escolher qualquer uma delas, sem qualquer critério prévio.
- **Vcoord1**: É um campo vetorial com quatro posições, cada uma delas é do tipo real com dupla precisão. Este campo é utilizado para armazenar as coordenadas cartesianas do vértice no instante $T1$. A última posição deste vetor não esta sendo utilizada, pois não estamos trabalhando com coordenas homogêneas.A definição de coordenadas homogênas é dada no Apêndice D.
- **Vcoord2**: É um campo vetorial com quatro posições, cada uma delas é do tipo real com dupla precisão. Este campo é utilizado para armazenar as coordenadas cartesianas do vértice no instante $T2$. A última posição deste vetor não esta sendo utilizada, pois não estamos trabalhando com coordenas homogêneas.
- **Velo1**: É um campo vetorial com três posições, cada uma delas é do tipo real com dupla precisão. Este campo é utilizado para armazenar as coordenadas x, y e z da

velocidade do vértice no instante $T1$.

- **Velo2**: É um campo vetorial com três posições, cada uma delas é do tipo real com dupla precisão. Este campo é utilizado para armazenar as coordenadas x, y e z da velocidade do vértice no instante $T2$.
- **MapLagran**: É um campo, do tipo ponteiro, que aponta para o endereço do vértice dentro do malha lagrangeana.
- **Nextv**: É um campo, do tipo ponteiro, que aponta para o próximo vértice.
- **Prevv**: É um campo, do tipo ponteiro, que aponta para o vértice anterior.

3.3 Inicialização da Estrutura de Dados Halfedge

É uma tarefa importante desenvolver uma maneira de inicializar a estrutura Halfedge a partir da saida gerada pelo programa GMSH, produzida no formato apresentado no Apêndice B. Os passos executados para se inicializar a estrutura Halfedge são:

- **Passo 1:**

  Cria-se o nó **Solid** e armazena-se no campo **IdSolid** o número um, para identificar o primeiro sólido a ser representado. Caso hajam outros sólidos, a numeração é feita de forma sequencial.

- **Passo 2:**

  Cria-se uma quantidade de nós **Vertex**, igual ao número de vértices do sólido. Na medida em que cada nó é lido do arquivo de saída gerado pelo GMSH, os seguintes campos do nó **Vertex** são inicializados, com:

  IdVertex: O código numérico gerado pelo GMSH para identificar o vértice, exceto para a esfera, onde se adota um código numérico próprio, já que o GMSH inicia a numeração dos vértices da esfera a partir do número 2.

  Vcoord1 e Vcoord2: As coordenadas cartesianas do vértice. Estes dois campos são inicializados com o mesmo conteúdo.

  Nextv e Prevv: É criada uma lista duplamente ligada entre os nós **Vertex**.

- **Passo 3:**

  Todos os vértices, de todos os triângulos da malha, são lidos e armazenados em uma estrutura de dados temporária, criada com o objetivo de auxiliar o preenchimento da estrutura Halfedge. Esta estrutura temporária, é apagada no momento em que toda estrutura Halfedge tiver sido inicializada.

- **Passo 4:**

  Cria-se uma quantidade de nós **Face**, igual ao número de triângulos da malha. A partir da estrutura de dados temporária, criada no passo 3, inicializa-se os seguintes campos do nó **Face**:

  IdFace: Cria-se um código numérico sequencial próprio, para identificar cada face (triângulo da malha).

  Vector: Faz-se o cálculo e armazena-se a equação do plano que contêm a face.

  Nextf e Prevf : É criada uma lista duplamente ligada entre os nós **Face** .

- **Passo 5:**

  Cria-se uma quantidade de nós **Loop**, igual ao número de triângulos da malha. Isto se deve ao fato de não existirem "buracos" no interior dos triângulos gerados pelo GMSH. Cada face é associada à uma única curva que representa sua fronteira externa, ou seja, a face não possui fronteira interna. Os seguintes campos dos nó **Loop** são inicializados:

  Face: Este ponteiro aponta para cada nó **Face** criado no passo 4.

  Nextl e Prevl: Cria-se uma lista duplamente ligada entre os nós **Loop**.

- **Passo 6:**

  Após todos os nós **Loop** terem sido inicializados, conforme descrito no passo anterior, inicializa-se os seguintes campos do nó **Face**:

  Fsolid: Aponta-se este ponteiro para o nó do **Solid** criado no passo 1.

  Flout: Aponta-se este ponteiro para o nó **Loop** que está associado à face. Isto reforça a idéia de que, cada face do sólido está associada a um único nó **Loop** e vice-versa.

  Loops: Este campo está associado à fronteira interna da face e como os triângulos da malha gerados pelo GMSH não posuem fronteira interna, este campo não é efetivamente utilizado.

- **Passo 7:**

  Para cada lado do triângulo da malha, associam-se dois nós **Halfedge**, um para cada sentido de percurso do lado. Cria-se uma quantidade de nós **Halfedge**, igual a três vezes o número total de triângulos da malha. A partir da estrutura de dados temporária criada no passo 3, inicializa-se os seguintes campos do nó **Halfedge**:

  Vtx: Aponta-se este ponteiro para cada nó **Vertex** que represente o vértice final da Halfedge.

  WLoop: Aponta-se este ponteiro para o nó **Loop**, criado no passo 5, ao qual esta Halfedge está associada.

  Nexth e Prevh: Cria-se uma lista duplamente ligada, somente entre as três Halfedges que formam cada triângulo da malha.

- **Passo 8:**

  Cria-se uma quantidade de nós **Edge**, igual à metade dos número de nós do tipo **Halfedge** e se inicializa os seguintes campos do nó **Edge**:

  IdEdge: Cria-se um código numérico sequencial para identificar cada um dos lados do triângulo.

  He1 e He2: Por intermédio de uma busca na estrutura de dados temporária, encontram-se as duas Halfedges que formam um dado lado do triângulo. À primeira Halfedge encontrada, aponta-se o ponteiro **He1** e à segunda, aponta-se o ponteiro **He2**.

  Nexte e Preve: Cria-se uma lista duplamente ligada entre os nós **Edge**.

- **Passo 9:**

  O campo **Edg** do nó **Halfedge**, é apontado para o respectivo nó **Edge**, ao qual a Halfedge está associada.

- **Passo 10:**

  O campo **Ledg** do nó **loop**, pode ser apontado para qualquer uma das três Halfedges que formam um triângulo. Em particular, o campo **Ledg** é sempre apontado para a primeira Halfedge encontrada na estrutura de dados temporária.

- **Passo 11:**

Associa-se o campo **Vedge** do nó **Vertex**, à primeira Halfedge encontrada na estrutura de dados temporária, que tenha este vértice como seu vértice inicial. É importante ressaltar que, apesar de poderem existir várias Halfedges começando num mesmo vértice, qualquer uma delas pode ser escolhida.

- **Passo 12:**

  Finalmente, faz-se a inicialização dos seguintes campos do nó **Solid**:

  Sface: Este ponteiro aponta para uma face qualquer do sólido.

  Sedges: Este ponteiro aponta para uma aresta qualquer do sólido.

  Sverts: Este ponteiro aponta para um vértice qualquer do sólido.

  Nexts e Prevs: Estes campos, que formam uma lista duplamente ligada entre os sólidos, só serão efetivamente utilizados, se houver mais de um sólido a ser representado pela estrutura Halfegde.

Os campos **Velo1**, **Velo2** e **MapLangran** do nó **Vertex**, somente serão inicializados posteriormente pelo programa, quando o sólido estiver se movendo.

Capítulo 4

Formulação Matemática e Esquema Numérico

Neste Capítulo, serão descritos as fórmulas matemáticas, o método numérico e o processo de adaptividade de malhas compostas utilizados para mover a superfície (vértices do sólido gerado pelo GMSH).

Na Seção 4.1, apresenta-se a formulação matemática adotada para se descrever o Método da Fronteira Imersa.

Na Seção 4.2, apresentam-se a definição do domínio computacional e a notação matemática que será utilizada ao longo deste Capítulo.

Na Seção 4.3, são descritos o processo de criação e as estratégias de adaptatividade das malhas compostas.

Na Seção 4.4 e na Seção 4.5, apresentam-se os processos de discretização espacial e temporal, respectivamente, utilizados na movimentação da fronteira imersa.

4.1 Modelo Matemático

No Método da Fronteira Imersa, o escoamento não estacionário de um fluido viscoso incompressível é modelado pelas equações de Navier-Stokes, as quais são escritas em coordenadas eulerianas e são dadas por:

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{\nabla p}{\rho} = \frac{\mu}{\rho}\Delta \mathbf{u} - (\mathbf{u} \cdot \nabla \mathbf{u}) + \frac{\mathbf{f}}{\rho}, \tag{4.1}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{4.2}$$

onde:

- ρ: Densidade de massa do fluido, suposta constante no tempo e no espaço.
- μ: Coeficiente de viscosidade do fluido, suposto constante no tempo e no espaço.
- p: Pressão hidrodinâmica.
- $\mathbf{u}$: Campo de velocidades.
- $\mathbf{f}$: Forças externas (A fronteira imersa se faz presente por intermédio deste termo forçante).

A partir de agora, a fronteira imersa será considerada como sendo a superfície do sólido gerada pelo GMSH, a qual se encontra completamente imersa em fluido. Se esta superfície for suficientemente delgada e se sua massa puder ser desprezada, ela pode ser vista puramente como um "gerador de forças" que não introduz massa nem cobre qualquer região do escoamento.

Sejam (r, s) as coordendas curvilíneas que descrevem a superfície bidimensional, isto é, cada valor fixado (r, s) representa um único ponto material da superfície. $\mathbf{X}(r, s, t)$ será a posição de um ponto da superfície no tempo t em coordenadas cartesianas, dada por

$$\mathbf{X}(r, s, t) = (X_1(r, s, t), X_2(r, s, t), X_3(r, s, t)), \quad r, s \in \mathcal{S}, \tag{4.3}$$

onde r,s são os parâmetros lagrangeanos.

A densidade de força, $\mathbf{F}$, é expressa em termos das coordenadas lagrangeanas da superfície; entretanto, na equação do fluido, (4.1), ela está escrita em coordenadas eulerianas. Para trocar a densidade de força de uma formulação para outra, é preciso lembrar que a força total que

atua sobre a sub-região $\mathcal{R}$ arbitrária do domínio Ω pode ser escrita como:

$$\begin{aligned}\int_{\mathcal{R}} f(\mathbf{x},t)\,d\mathbf{x} &= \int_{\{r,s:\,\mathbf{X}(r,s,t)\,\in\,\mathcal{R}\}} F(r,s,t)\,dr\,ds \\ &= \int_{\mathcal{S}} F(r,s,t)\,\omega_{\mathcal{R}}(\mathbf{X}(r,s,t))\,dr\,ds \\ &= \int_{\mathcal{S}} F(r,s,t)\left[\int_{\mathcal{R}} \delta(\mathbf{x}-\mathbf{X}(r,s,t))\,d\mathbf{x}\right] dr\,ds \\ &= \int_{\mathcal{R}}\int_{\mathcal{S}} F(r,s,t)\,\delta(\mathbf{x}-\mathbf{X}(r,s,t))\,dr\,ds\,d\mathbf{x}, \end{aligned} \tag{4.4}$$

onde, $\omega_{\mathcal{R}}(x) = 1$, $x \in \mathcal{R}$ e 0 caso contrário, e δ é a função Delta de Dirac tridimensional .

Embora estas manipulações sejam formais, elas motivam uma expressão para a densidade de força no fluido devida à presença da superfície em coordenadas eulerianas. Como $\mathcal{R}$ é uma região arbitrária, a expressão é dada por:

$$f(\mathbf{x},t) = \int_{\mathcal{S}} F(r,s,t)\delta(\mathbf{x}-\mathbf{X}(r,s,t))\,dr\,ds. \tag{4.5}$$

Como pode ser observado em (4.5), a densidade de força f tem suporte apenas sobre a superfície, isto é, ela vale zero em todos os pontos do domínio, excetuando-se os pontos da superfície. Outro fato importante que deve ser observado é que, uma vez que o Delta de Dirac tridimensional é integrado sobre a superfície (bidimensional), f tem o mesmo tipo de singularidade definida para uma função Delta de Dirac unidimensional.

Neste trabalho, a interação fluido-estrutura é considerada, apenas, no sentido do fluido para a estrutura; tudo se passa, como se os vértices do sólido pertencessem ao próprio fluido e se movessem ao sabor do escoamento ($\mathbf{f}$=0).

Sendo o fluido viscoso, os pontos da superfície acabam por mover-se com a sua velocidade local ("nonslip condition"). Matematicamente, esta condição pode ser expressa em termos do campo de velocidades como:

$$\frac{\partial \mathbf{X}(r,s,t)}{\partial t} = \mathbf{u}(\mathbf{X}(r,s,t),t). \tag{4.6}$$

Uma vez mais, uma troca entre coordenadas é necessária. Desta vez, deve-se encontrar uma expressão para a velocidade langrangeana dos pontos da superfície em termos das coordenadas eulerianas do fluido. Empregando-se formalmente as propriedades da função Delta de Dirac tridimensional, tem-se

$$\mathbf{u}(\mathbf{X}(r,s,t),\mathrm{t}) = \int_{\Omega} \mathbf{u}(\mathbf{x},\mathrm{t})\delta(\mathbf{x}-\mathbf{X}(r,s,t))\,\mathbf{dx}, \tag{4.7}$$

onde Ω representa todo o domínio.

A equação (4.5) e a equação (4.7) descrevem a interação entre o fluido e a superfície; elas empregam a função Delta de Dirac para alternar entre a formulação euleriana, usada para o fluido e a lagrangeana, usada para a superfícies. A equação (4.5) "espalha" as tensões que agem sobre a superfície para o resto do domínio; por esta razão é denominada como *passo de espalhamento*. A equação (4.7) "interpola" nos pontos da superfície as velocidades definidas no domínio (pontos da malha euleriana); é o *passo de interpolação*. A equação (4.5) não é considerada neste trabalho e a interpolação de velocidades, dada pela equação (4.7), é explicada na Seção 4.4

Finalmente é importante ressaltar que por intermédio da manipulação adequada do termo $F(r, s, t)$, presente na equação 4.4, é que se pode variar a modelagem da superfície, isto é, podemos considerá-la como sendo uma estrutura rígida ou elástica ou ainda como sendo um interface de fluidos multifásicos. Esta abordagem permite que uma gama muito grande de problemas físicos possam ser modelados e resolvidos, utilizando-se o Método da Fronteira Imersa.

4.2 Domínio Computacional

O *domínio físico*, definido pelo produto cartesiano $\Omega = [a_x, b_x] \times [a_y, b_y] \times [a_z, b_z]$, é discretizado por células computacionais pertencentes a uma malha do tipo bloco-estruturada, refinada localmente em regiões de especial interesse. A união de todas as células computacionais forma o *domínio computacional*.

As malhas aqui utilizadas, denominadas *malhas compostas*, são obtidas por intermédio da aplicação do algoritmo desenvolvido por Berger e Rigoutsos [17], o qual se baseia em técnicas de visão computacional e reconhecimento de padrões. O processo de refinamento permite que regiões importantes do escoamento (e.g. camadas limites, regiões de alta vorticidade e/ou turbulência) possam ser tratadas com a acurácia adequada.

Malhas compostas são definidas por uma seqüência hierárquia de malhas aninhadas, progressivamente mais finas, agrupadas em níveis de refinamento $l = 1, 2, \ldots, l_{\text{mais fino}}$. O l-ésimo nível de refinamento, Ω_l, é dado pela união de um conjunto de malhas cartesianas $G_{l,k}$,

(retângulos em duas dimensões e paralelepípedos em três dimensões),

$$\Omega_l = \bigcup_k G_{l,k}, \quad k = 1, 2, \ldots, n_l,$$

as quais não têm pontos internos comuns, isto é, $= {}_{l,j} \cap \dot{G}_{l,k} = \emptyset, \forall\, j \neq k.$

Malhas pertencente a um mesmo nível l possuem os mesmos espaçamentos Δx_l, Δy_l e Δz_l, e tem suas arestas paralelas aos eixos coordenados. Além disso, elas devem estar *apropriadamente aninhadas* no nível mais grosso imediatamente abaixo, $l-1$, isto é, devem satisfazer as seguintes propriedades:

1. Uma malha fina deve começar e terminar num canto de uma célula da próxima malha mais grossa;
2. Deve haver ao menos uma célula no nível $l-1$, pertencente a alguma malha de nível $l-1$, separando uma célula no nível l de uma célula do nível $l-2$ em todas as direções (norte, sul, leste, oeste, frente e fundo), exceto se esta célula tocar o bordo do domínio físico.

A Figura 4.1 mostra dois exemplos de malhas compostas (bidimensionais, por clareza) não aninhadas apropriamente, sendo que a da esquerda viola a propriedade 1 e a da direita viola a propriedade 2.

Observe que um aninhamento apropriado não exige que uma malha mais fina fique contida apenas em uma única malha mais grossa. Por exemplo, na Figura 4.2, há uma malha do nível $G_{3,1}$, apropriadamente aninhada, estando situada sobre a união de duas malhas mais grossas.

Por definição, cada uma das células computacionais do nível l, tem seu centro localizado em

$$\mathbf{x}_{i,j,k} = (x_i, y_j, z_k) = (a_x + (i - \tfrac{1}{2})\Delta x_l, a_y + (j - \tfrac{1}{2})\Delta y_l, a_z + (k - \tfrac{1}{2})\Delta z_l), \qquad (4.8)$$

com $1 \leq i \leq N_{x,l}$, $1 \leq j \leq N_{y,l}$ e $1 \leq k \leq N_{z,l}$, onde, $N_{x,l}$, $N_{y,l}$ e $N_{z,l}$ seriam os números de células computacionais nas direções x, y e z, respectivamente, se o l-ésimo nível de refinamento fosse dado por uma malha uniforme, isto é, $N_{x,l}=(b_x - a_x)/\Delta x_l$, $N_{y,l}=(b_y - a_y)/\Delta y_l$ e $N_{z,l}=(b_z - a_z)/\Delta z_l$(*indexação global*).

Por conveniência, assume-se que as malhas do nível mais fino recobrem de forma uniforme toda a superfície imersa e, até menção em contrário, Δx, Δy e Δz são os espaçamentos das malhas pertencentes ao nível mais fino.

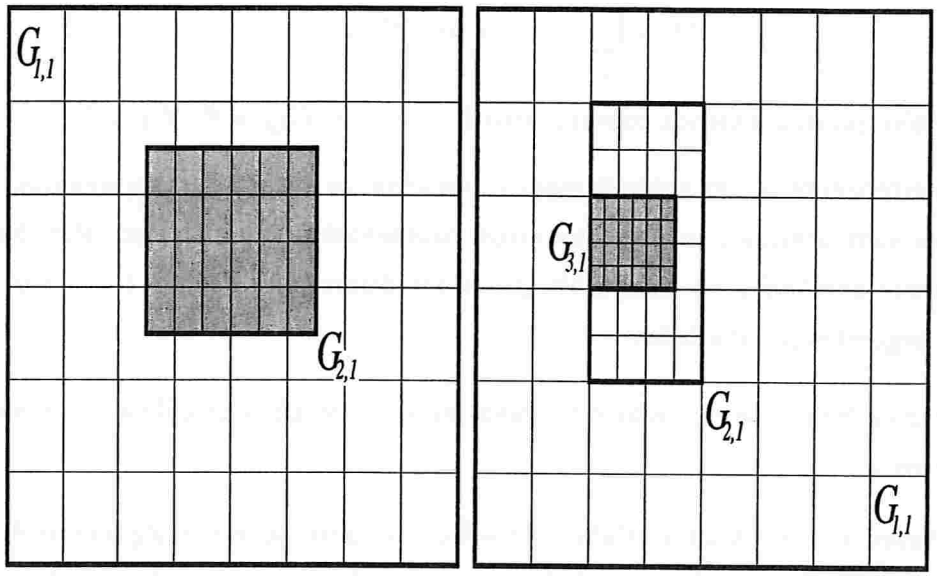


Figura 4.1: Malhas compostas bidimensionais não apropriadamente aninhadas [13].

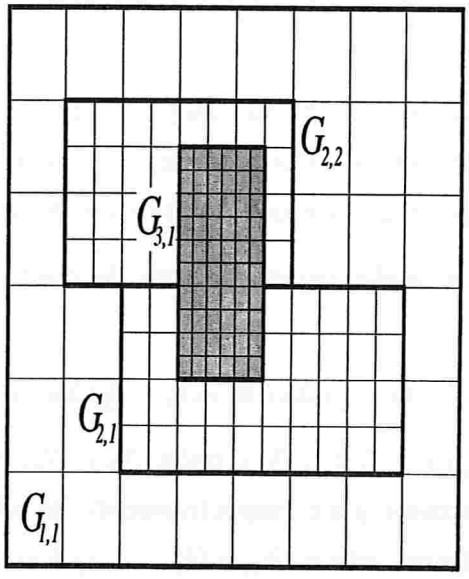


Figura 4.2: Malha composta $G_{3,1}$, colocada sobre duas malhas, estando apropriadamente aninhada [13].

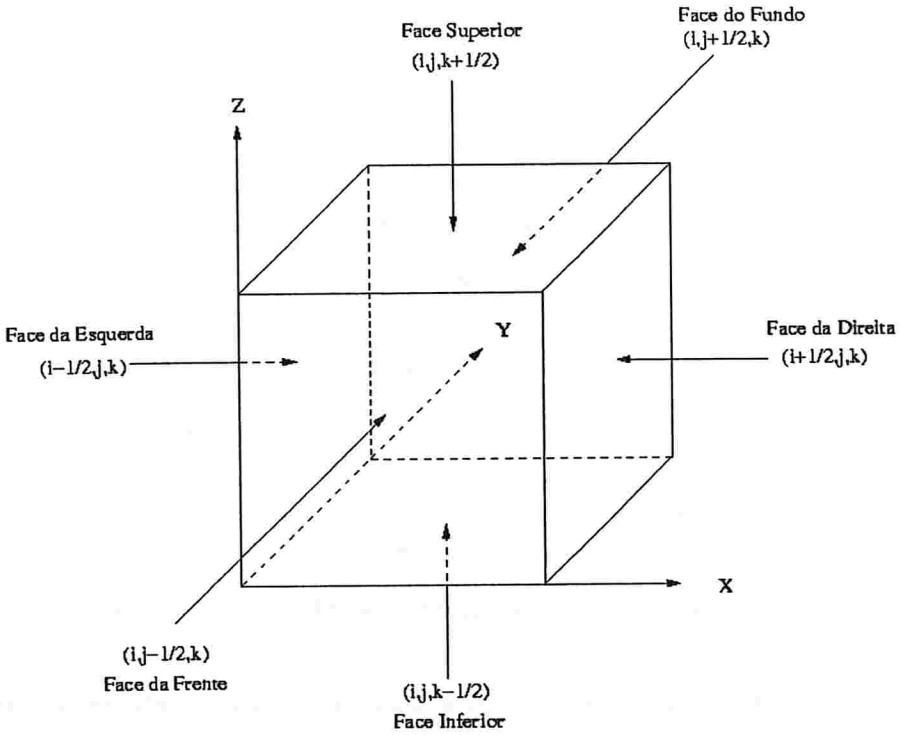


Figura 4.3: Definição dos índices.

Para evitar mudanças nos esquemas numéricos perto do bordo do domínio computacional e dos bordos de cada malha, em cada nível, camadas adicionais de células são acrescentadas em cada uma das direções. Estas células adicionais são denominadas de *células fantasmas*. Para os esquemas numéricos que se seguem, será suficiente adotar duas camadas de células fantasmas, cuja largura é igual ao de duas células computacionais.

A indexação utilizada baseia-se no índice do centro da célula computacional definido em (4.8). Dada a célula computacional de índices (i, j, k), Figura 4.3, tem-se:

- os centros das duas faces paralelas ao plano ZY (esquerda e direita) têm índices $(i - \frac{1}{2}, j, k)$ e $(i + \frac{1}{2}, j, k)$, respectivamente;
- os centros das duas faces paralelas ao plano XZ (frente e fundo) têm índices $(i, j - \frac{1}{2}, k)$ e $(i, j + \frac{1}{2}, k)$, respectivamente;
- os centros das duas faces paralelas ao plano XY (inferior e superior) têm índices $(i, j, k - \frac{1}{2})$ e $(i, j, k + \frac{1}{2})$, respectivamente.

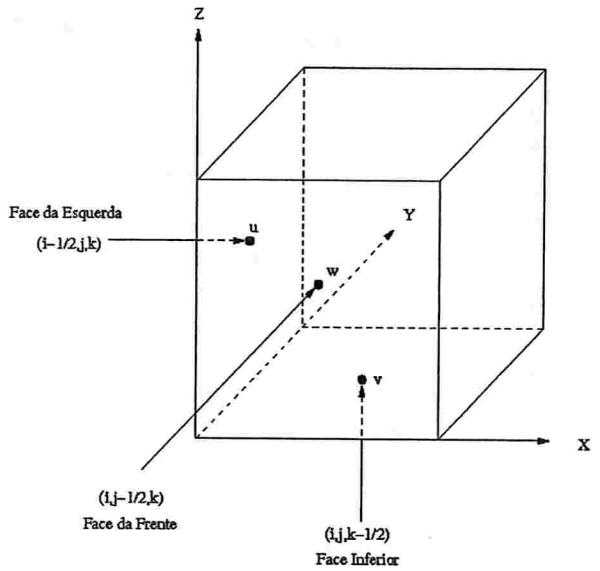


Figura 4.4: Representação da grandeza vetorial (Velocidade).

Uma vez que o domínio físico tenha sido discretizado por este conjunto de células computacionais, ainda há que se decidir a localização do campo de velocidades envolvido no problema. De forma similar a Harlow e Welch [12], as células computacionais são vistas como sendo volumes de controle de massa e as *grandezas vetorias* (e.g. velocidade) têm suas componentes u, v e w definidas nos centros das faces da célula computacional, Figura 4.4, isto é, a velocidade $\mathbf{u}_{i,j,k}$ definida na célula computacional (i,j,k) é:

$$\mathbf{u}_{i,j,k} = (u_{i-\frac{1}{2},j,k},\ v_{i,j-\frac{1}{2},k},\ w_{i,j,k-\frac{1}{2}}),$$

onde, $u_{i-\frac{1}{2},j,k}$ é a primeira componente da velocidade, definida no meio da face esquerda (plano ZY); $v_{i,j-\frac{1}{2},k}$ é a segunda componente, definida no meio da face da frente (plano XZ) e a última componente, $w_{i,j,k-\frac{1}{2}}$), é definida no meio da face inferior (plano XY).

4.3 Geração e Adaptatividade da Malha Euleriana

O Método da Fronteira Imersa, faz uso de dois tipos de malha computacional: a malha euleriana, que é utilizada para representar o fluido e a malha lagrangeana, que é utilizada para representar a superfície do sólido. Nesta Seção, descreve-se o procedimento usado para determinar o conjunto de células que formarão a região onde é necessário que se faça o

refinamento da malha euleriana. Este conjunto de células, não deve incluir somente as células para as quais a solução necessita maior resolução mas, também, deve incluir aquelas células necessárias para manter as malhas retangulares e apropriadamente aninhadas.

Os níveis de refinamento da malha composta são gerados um de cada vez, começando com o nível mais fino, $l_{\text{mais fino}}$, e terminando no nível 2. A(s) malha(s) pertencente(s) ao nível 1 é (são) sempre fixadas e o número total dos níveis a ser usado é estabelecido automaticamente, sendo definido **a priori**. Os níveis de refinamento são gerados nesta ordem particular por ser ela a forma mais simples de se assegurar que todas as malhas da malha composta, fiquem apropriadamente aninhadas no final do processo de refinamento.

Um nível l é gerado por células marcadas no próximo nível mais grosso, nível, $l-1$. Como pode ser observado, isto conflitua com a ordem na qual os níveis são gerados, criando o seguinte impasse: *Como é possível marcar células num nível que não existe ainda* ?

Este impasse pode ser solucionado se um nível hipotético provisório auxiliar, $l-1$, dado por uma única malha uniforme hipotética com espaçamento de malha dado por

$$\Delta x_{l-1} = \frac{\Delta x_1}{2^{(l-1)-1}}, \qquad \Delta y_{l-1} = \frac{\Delta y_1}{2^{(l-1)-1}} \quad e \quad \Delta z_{l-1} = \frac{\Delta z_1}{2^{(l-1)-1}}$$

for "imaginado" recobrindo o domínio.

As considerações acima conduzem a um procedimento de dois passos para se obter o conjunto de células marcadas que serão usadas para gerar as malhas de nível l. No primeiro passo, uma vizinhança no nível $l-1$, é marcada em torno de cada ponto da superfície. A região cinza indicada na Figura 4.5 é um exemplo bidimensional de tal vizinhança para um ponto específico da superfície.

Ao conjunto de células marcadas no final do passo anterior, deve-se adicionar as células necessárias para que as malhas do nível $(l+1)$ estejam adequadamente aninhadas no nível l. As malhas do nível $(l+1)$ não necessariamente se aninham apropriadamente no nível l, se as fronteiras das malhas do nível l, tivessem sido obtidas simplesmente pela adição de uma camada de células do nível l, para as malhas do nível $(l+1)$, em todas as direções, Figura 4.6 (à esquerda). Entretanto, este procedimento poderia produzir uma malha no nível l, que jamais pudesse estar adequadamente aninhada em qualquer nível $(l-1)$. Como pode ser observado na Figura 4.6 (à direita), poderia existir cantos mal posicionados, o que violaria a propriedade 1 na definição de malhas adequadamente aninhadas.

A situação acima pode ser facilmente evitada por intermédio da inclusão de um conjunto

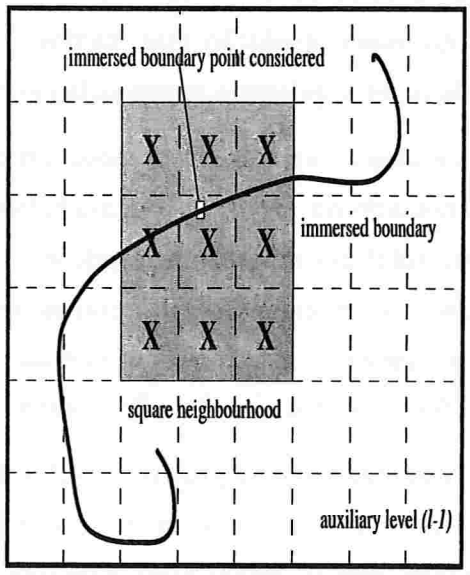


Figura 4.5: Células marcadas ao redor de pontos da superfície (marcadas com X). As células pontilhadas pertencem ao nível auxiliar $(l-1)$ [13].

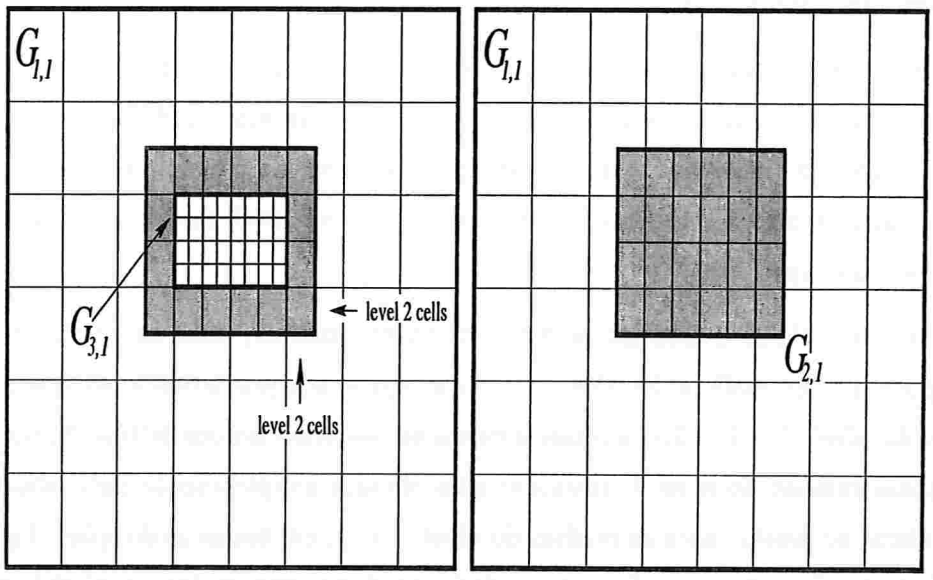


Figura 4.6: Simplesmente extender malhas mais fina de uma "quantidade correta" (à esquerda), pode levar a malhas que não fiquem apropriadamente aninhadas violando a propriedade 1 (à direita) [13].

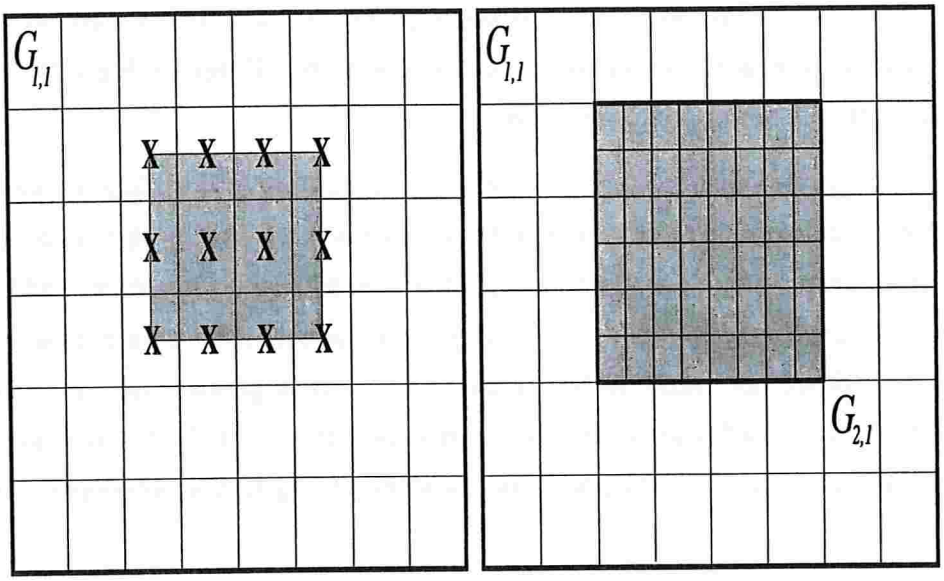


Figura 4.7: Fazendo a interseção de uma malha fina extendida, mostrada em cinza, com células mais grossas (esquerda), nunca irá evitar que, níveis mais grossos tenham malhas sendo geradas adequadamente aninhada (direita) [13].

de células marcadas em todos os níveis $(l-1)$ que fazem intersecção com as malhas estendidas de nível $(l+1)$. A Figura 4.7 mostra que seguindo este procedimento não apenas as malhas de nível $(l+1)$ serão adequadamente aninhadas no nível l mas também o nível $(l-1)$ não será impedido de ter o nível l aninhado adequadamente.

Em resumo, o conjunto de células marcadas é obtido para cada nível l seguindo o procedimento de dois passos:

1. marque as células de erro grande por intermédio das células marcadas de nível $(l-1)$ numa vizinhança quadrada ao redor da superfície (Figura 4.5). O tamanho da vizinhança quadrada não pode ser menor que o tamanho do suporte de função Delta de Dirac;

2. marque as células de nível $(l-1)$ de forma a assegurar que as malhas mais finas, fiquem adequadamente aninhadas no nível l em seu final. Isso pode ser obtido marcando todas as células $(l+1)$ que fazem intersecção com o retângulo obtido por intermédio da extensão das malhas de nível $(l+1)$ por uma célula de nível l em todas as direções, conforme pode ser visto na Figuras 4.6 e 4.7.

Como foi mencionado anteriormente, níveis de refinamento de malhas compostas são gerados um a um, começando pelo nível $l_{\text{mais fino}}$ e terminando no nível 2. Após a obtenção do

conjunto de células marcadas, as malhas, as quais pertencem a um determinado nível l, são criadas por meio da aplicação do algoritmo desenvolvido por Berger e Rigoutsos (17) para geração destas malhas com pontos marcados.

Por intermédio da combinação de elementos tanto da computação gráfica, quanto da teoria de reconhecimento de padrões, dado o conjunto de células marcadas, o algoritmo de Berger e Rigoutsos devolve uma gama de malhas retangulares que não se interceptam, cada um delas, seguindo um certo critério de eficiência. Este algoritmo foi adaptado para o espaço tridimensional, para que pudesse ser usado neste trabalho. Em termos gerais, esse algoritmo detecta transições de regiões marcadas para não-marcadas e encontra o "melhor" lugar para cortar a malha usando uma técnica chamada por seus criadores de "signature technique" (técnica da sinalização).

As funções sinalizadoras horizontal e vertical de uma matriz MxN denominada A, contendo somente entradas 0 e 1 (0's para as posições das células não-marcadas e 1's para as marcadas), são definidas por:

$$S_H(j) = \sum_{i=1}^{M} A(i,j)$$

$$S_V(i) = \sum_{j=1}^{N} A(i,j)\,,$$

que dá o número de entradas não-zero em uma linha ou coluna específica, respectivamente.

Baseado-se em idéias relacionadas à detecção de arestas, a mais importante aresta será aquela responsável pelo maior ponto de inflexão nas sinalizações da matriz A, a qual armazena a imagem da região marcada. Mais especificamente, a aresta mais importante estará localizada no mais importante cruzamento de zero da segunda derivada da função sinalizadora da matriz A. Por exemplo, a Figura 4.8 mostra como essa técnica é usada para identificar a aresta mais importante em uma determinada região marcada.

O símbolo, Δ, que aparece na Figura 4.8, corresponde à discretização da segunda derivada associada a função sinalizadora, para a qual a aproximação de diferenças centradas foi utilizada.

A técnica das sinalizações pode ser usada para gerar uma gama de caminhos retangulares pertencentes à certo nível l, cuja união contém todas as células marcadas. A construção de um nível l de refinamento inicia-se primeiro pela construção de somente uma malha espalhada pela

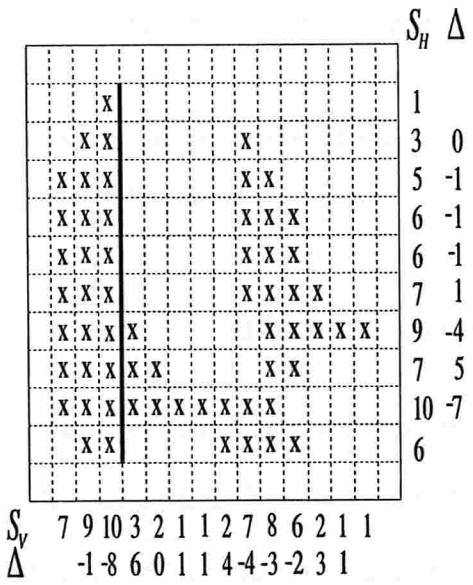


Figura 4.8: A maior ponto de inflexão acontece na linha grossa, a qual faz o corte mais eficiente [13].

região inteira das células marcadas. Esta malha é chamada, por seus criadores, de (rectangular hull) , que por definição, é o menor retângulo alinhado com o eixo x, que contém um certo grupo de células.

Se a malha falha no teste de eficiência:

$$e_{\min} \leq \frac{\text{número de células marcadas}}{\text{número total de células da malha}}, \tag{4.9}$$

onde $0.0 \leq e_{\min} \leq 1.0$.

A técnica de sinalização explicada anteriormente é usada para encontrar a aresta mais proeminente onde a malha deve ser separada em duas (cortada ao meio). Portanto, como resultado, obtém-se a partição da região original de células marcadas em duas sub-regiões. Cada sub-região (rectangular hull) obtida é testada usando 4.9. Se uma malha falha em um teste, então a técnica da sinalização é aplicada a ela. Esse procedimento é feito recursivamente para cada sub-região até que todas as malhas passem no teste 4.9. A única exceção a esta regra, ocorre quando não for possível encontrar nenhuma partição eficiente de um (rectangular hull), usando a técnica da sinalização. Nesse caso, a malha é aceita do jeito que ela é. Mais detalhes podem ser encontrados em (17). Uma vez que a malha pertencente ao nível l foi criada, a coleção de células marcadas necessárias para gerar o nível $(l-1)$ é obtido, seguindo-se os procedimentos já explicados. À essa coleção, é aplicado então o algoritmo Berger e

Rigoutsos, o qual irá retornar o grupo de malhas do nível $l-1$. Isto é repetido até que o nível 2 seja criado.

O processo de remalhagem utilizado neste trabalho é feito automaticamente após um número pré-estabelecido de passos de integração no tempo terem sido executados,

4.4 Discretização no Espaço

A evolução da posição da superfície no tempo é dada pela resolução da equação

$$\frac{\partial \mathbf{X}(r,s,t)}{\partial t} = \mathbf{u}(\mathbf{X}(r,s,t),t) = \int_{\Omega} \mathbf{u}(\mathbf{x},t)\delta(\mathbf{x}-\mathbf{X}(r,s,t))\,d\mathbf{x}, \tag{4.10}$$

onde **u** é a velocidade do fluido, conhecida nos pontos da malha euleriana e que será interpolada para os pontos **X** da superfície, utilizando uma forma discreta tridimensional da função Delta de Dirac[19].

Define-se aqui tal forma como sendo o produto

$$\delta^3_\Delta(\mathbf{x}-\mathbf{X}) = \delta^1_\Delta(x-X_1)\delta^1_\Delta(y-X_2)\delta^1_\Delta(z-X_3) \tag{4.11}$$

onde

$$\delta^1_\Delta(x-X_1) = \frac{1}{\Delta x}\phi\left(\frac{x-X_1}{\Delta x}\right), \tag{4.12}$$

$$\delta^1_\Delta(y-X_2) = \frac{1}{\Delta y}\phi\left(\frac{y-X_2}{\Delta y}\right), \tag{4.13}$$

$$\delta^1_\Delta(z-X_3) = \frac{1}{\Delta z}\phi\left(\frac{z-X_3}{\Delta z}\right), \tag{4.14}$$

sendo δ^1_Δ uma aproximação unidimensional da função delta e ϕ uma função contínua definida por

$$\phi(r) = \begin{cases} \frac{1}{6}(5-3\mid r\mid -\sqrt{-3(1-\mid r\mid)^2+1}\,) & , 0.5 \leq \mid r \mid \leq 1.5, \\ \frac{1}{3}(1+\sqrt{-3r^2+1}\,) & , \mid r \mid \leq 0.5, \\ 0 & , \text{caso contrário,} \end{cases} \tag{4.15}$$

com $r = (x - X_1)/\Delta x$ ou $r = (y - X_2)/\Delta y$ ou $r = (z - X_3)/\Delta z$, $\mathbf{X} = (X_1, X_2, X_3)$, as coordenadas cartesianas de um ponto, q, pertencente à superfície e $\mathbf{x} = (x, y, z)$, um ponto da vizinhança de q, situado sobre a malha euleriana e no qual se conhece a velocidade do fluido.

A função ϕ acima não foi escolhida arbitrariamente. Na verdade, ela foi determinada por imposição de que um certo conjunto de propriedades as quais devem ser satisfeitas para uma versão discreta da função Delta de Dirac (para mais detalhes veja [19]). Em particular, o conjunto de propriedades que serão usadas para determinar esta aproximação são:

1. $\phi(r)$ é contínua, $C^0(\mathbb{R})$, $\forall r \in \mathbb{R}$;
2. $\phi(r) = 0, \quad \mid r \mid \geq 1.5, \forall r \in \mathbb{R}$;
3. $\sum_i \phi(r - i) = 1, \forall r \in \mathbb{R}$;
4. $\sum_i (r - i)\phi(r - i) = 0, \forall r \in \mathbb{R}$;
5. $\sum_i [\phi(r - i)]^2 = \frac{1}{2}, \forall r \in \mathbb{R}$,

onde todas as somas são feitas para os inteiros i, tais que $-\infty < i < +\infty$.

Estas propriedades determinam unicamente a função ϕ e por conseguinte a função δ_Δ. É importante resssaltar que, segundo [19], a função ϕ é continuamente derivável, embora esta condição não esteja explicitamente imposta. A título ilustrativo, a Figura 4.9, mostra a função bidimensional δ^2_Δ discretizada.

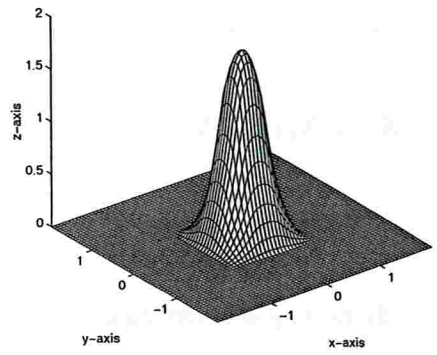


Figura 4.9: Função bidimensional δ^2_Δ discretizada [13].

A propriedade 1, evita saltos nos operadores de interpolação. A propriedade 2 garante que a função delta discretizada tem um suporte finito, o qual, neste estudo, é formado por três células computacionais. Quando combinadas, as propriedades 3 e 4 garantem que a interpolação das funções lineares sejam exatas. Logo se, funções suaves forem interpoladas os resultados obtidos terão precisão de segunda ordem.

Finalmente, a propriedade 5 tem sua origem na consideração de que a força de um ponto da superfície influencia o seu próprio movimento e também da imposição de que tal influência é a mesma qualquer que seja a posição deste ponto relativa a malha. Observe que a constante $1/2$ na quinta propriedade não é arbitrária. Ela pode ser obtida definindo-se $r = 1/2$ e por intermédio de algumas manipulações algébricas. Similarmente, na condição $\phi(r) = 0$ para $\mid r \mid \geq 1.5$, a constante 1.5 não é arbitrária. Esta é a menor valor da constante compatível com as outras condições (detalhes em [5]).

4.5 Discretização no Tempo

Uma vez adotada a discretização tridimensional da função Delta de Dirac, conforme equação (4.11) e sendo, $\mathbf{U}^n_{r,s}(\mathbf{X}(r,s,t),t)$, a velocidade no ponto lagrangeano $\mathbf{X}(r,s,t)$ da superfície no instante $t^n = t^{n-1} + \Delta t$, a ser obtida por interpolação , isto é, o cálculo da velocidade de um ponto $\mathbf{X}(r,s,t)$ da superfície é feita por interpolação, a partir da velocidade dos pontos do fluido (pertencentes a malha euleriana) numa vizinhança deste ponto, vide Figura 4.10, onde cada componente do vetor velocidade $\mathbf{U}^n_{r,s} = (U^n_{r,s}, V^n_{r,s}, W^n_{r,s})$ de um ponto $\mathbf{X}(r,s,t)$ da superfície, no instante t^n, é calculado da seguinte forma:

$$U^n_{r,s} = [\sum_{i-2}^{i+2}\sum_{j-2}^{j+2}\sum_{k-2}^{k+2} \mathbf{u}^n_{i-\frac{1}{2},j,k}\,\delta^3_\Delta(\mathbf{x}_{i-\frac{1}{2},j,k} - \mathbf{X}^n_{r,s})]\Delta x_l \Delta y_l \Delta z_l$$

$$V^n_{r,s} = [\sum_{i-2}^{i+2}\sum_{j-2}^{j+2}\sum_{k-2}^{k+2} \mathbf{v}^n_{i,j-\frac{1}{2},k}\,\delta^3_\Delta(\mathbf{x}_{i,j-\frac{1}{2},k} - \mathbf{X}^n_{r,s})]\Delta x_l \Delta y_l \Delta z_l$$

$$W^n_{r,s} = [\sum_{i-2}^{i+2}\sum_{j-2}^{j+2}\sum_{k-2}^{k+2} \mathbf{w}^n_{i,j-\frac{1}{2},k}\,\delta^3_\Delta(\mathbf{x}_{i,j,k-\frac{1}{2}} - \mathbf{X}^n_{r,s})]\Delta x_l \Delta y_l \Delta z_l$$

É importante observar que os índices i, j e k utilizados nos somatórios acima se referem aos índices das células computacionais da malha euleriana que se situam nas vizinhanças do ponto lagrangeano $\mathbf{X}(r,s,t)$ da superfície

Embora esquemas numéricos de ordem mais elevadas, com propriedades melhores de estabilidade e eficiência, possam ser empregados, por simplicidade e clareza de exposição adotar-se-á aqui a discretização, no tempo pelo Método de Euler Explícito

$$\mathbf{X}_{r,s}^{n+1} = \mathbf{X}_{r,s}^{n} + \Delta t\ \mathbf{U}_{r,s}^{n} \tag{4.16}$$

onde q representa um ponto qualquer pertencente à superfície.

Note que a escolha do passo de integração no tempo está sujeita a seguinte condição,

$$\Delta t = \min\{\Delta t_1, \Delta t_2\},$$

onde, $\Delta t_1 = \min\{\frac{\Delta x_l}{||u_{ijk}^n||_\infty}, \frac{\Delta y_l}{||v_{ijk}^n||_\infty}, \frac{\Delta z_l}{||w_{ijk}^n||_\infty}\}$ e tem o objetivo não permitir que os vértices da superfície se movam mais do que uma celula computacional por vez e
$\Delta t_2 = \max\{0, t_{final} - t^n\}$ e tem o objetivo de assegurar que o passo no tempo não ultrapasse o tempo final t_{final}.

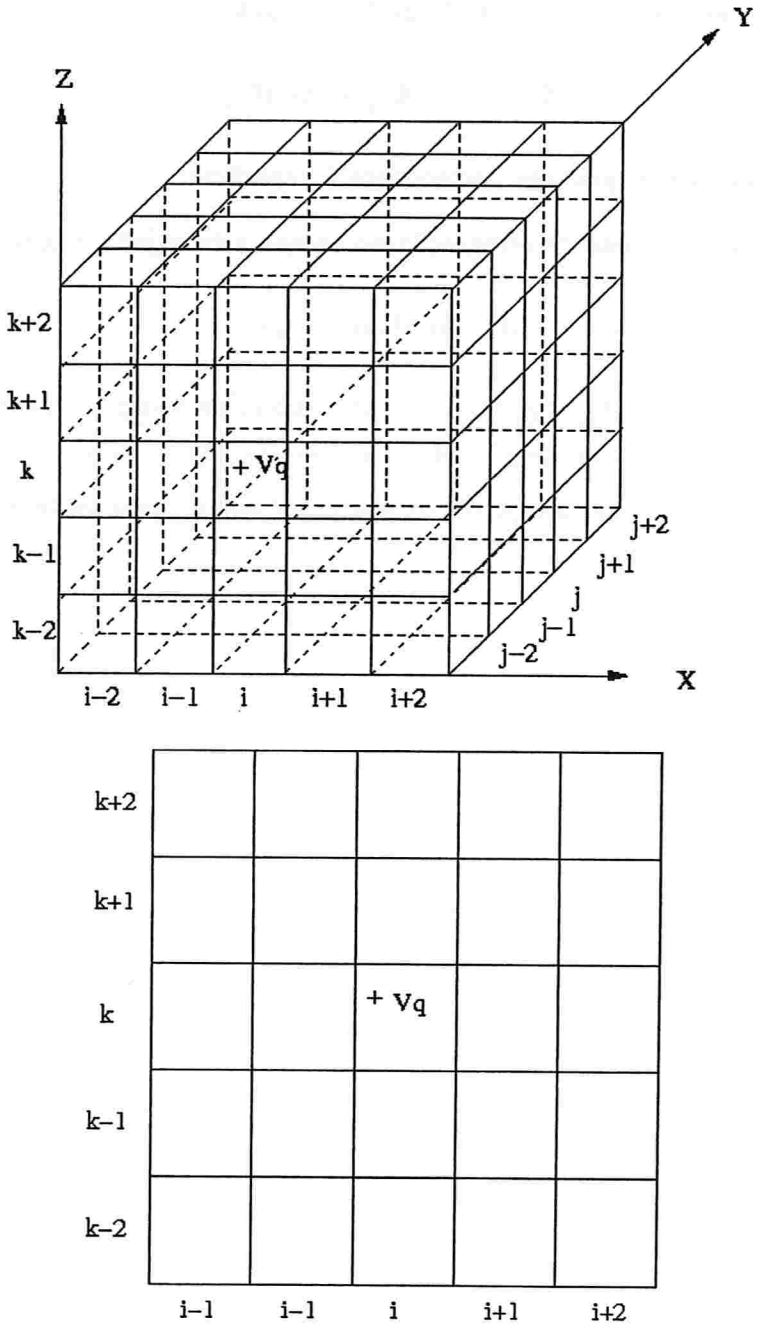


Figura 4.10: Vizinhança de um ponto $Vq = \mathbf{X}(r, s, t)$ da superfície usada na interpolação da velocidade (superior) e Corte paralelo ao plano XZ passando por Vq (inferior).

CAPÍTULO 5

Resultados Numéricos

Neste Capítulo, são apresentados os principais resultados númericos do processo de validação da interpolação da velocidade e do método de integração utilizados na movimentação da superfície.

Na Seção 5.1, apresenta-se o processo de validação da interpolação da velocidade para os pontos da superfície a partir da velocidade dos pontos do fluido na sua vizinhança.

Na Seção 5.2, apresenta-se o processo de validação do método de integração numérica utilizado para mover a superfície.

Na Seção 5.3, apresenta-se alguns testes de movimentação da superfície, nos quais se varia a forma geométrica da superfície e o campo de velocidades do fluido.

5.1 Validação da Interpolação da Velocidade

Para o processo de validação da interpolação da velocidade, utiliza-se para representar a fronteira imersa a superfície de uma esfera de raio $3,000\text{x}10^{-1}$ e centro na origem, Figura 5.1. Esta superfície é representada por uma malha triangular gerada pelo GMSH, a qual é composta por 10.107 vértices, 30.315 faces e 20.210 arestas, sendo o comprimento da menor aresta igual a $6,949\text{x}10^{-3}$, o da maior aresta igual a $2,196\text{x}10^{-2}$ e o da aresta média igual a $1,163\text{x}10^{-3}$. Durante o processo de validação da velocidade varia-se o tipo de malha euleriana empregado, mantendo-se fixa a malha lagrangeana.

Neste estudo, utiliza-se domínio físico Ω=$[-0,5\ ,\ 0,5]\text{x}[-0,5\ ,\ 0,5]\text{x}[-0,5\ ,\ 0,5]$ e um

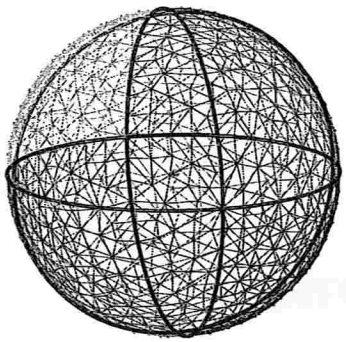


Figura 5.1: Esfera utilizada no validação da interpolação da velocidade.

campo de velocidades definido por $\mathbf{u}_E = (cos(x+y+z),\ cos(x+y+z),\ cos(x+y+z))$.

Os resultados da interpolação da velocidade para malhas compostas são apresentados na Tabela 5.1, na qual pode ser visto na primeira coluna, o tipo de malha euleriana utilizado, por exemplo, a notação $64L3$, significa que a malha composta utilizada tem 3 níveis de refinamento, no primeiro nível, existem 64 células computacionais em cada uma das direções, isto é, o domínio físico é dividido em 64 intervalos de igual espaçamento em cada uma das direções x, y e z, no segundo nível de refinamento, utiliza-se um espaçamento de malha igual à metade daquele utilizado no primeiro nível, no terceiro nível de refinamento, utiliza-se um espaçamento de malha igual à metade daquele utilizado no segundo nível, (Figura 5.2); na segunda coluna, apresenta-se o espaçamento da malha no nível mais fino, na terceira coluna, mostra-se o número de malhas que formam o nível mais fino; na quarta coluna, apresenta-se o valor da norma do máximo da diferença entre as velocidades prevista e interpolada para os vértices da superfície e na quinta coluna, calcula-se a razão entre duas normas consecutivas.

Na Tabela 5.1, pode-se observar ainda, na quarta coluna que a interpolação da velocidade para os pontos da superfície, depende do espaçamento da malha euleriana no seu nível mais fino, isto é, quanto menor for o espaçamento, menor será a norma do máximo da diferença entre as velocidades prescrita e interpolada. Observa-se na quinta coluna que a razão de convergência da norma do máximo tende para quatro o que confirma, que este método de interpolação da velocidade é de segunda ordem.

Malha	$h_{maisfino}$	Malhas no nível mais fino	$\|\|\mathbf{u}_N - \mathbf{u}_E\|\|_\infty$	Razão
8L3	$3,125\text{x}10^{-2}$	1	$7,808\text{x}10^{-4}$	-
16L3	$1,563\text{x}10^{-2}$	1	$1,951\text{x}10^{-4}$	4,002
32L3	$7,812\text{x}10^{-3}$	157	$4,868\text{x}10^{-5}$	4,008
64L3	$3,906\text{x}10^{-3}$	635	$1,204\text{x}10^{-5}$	4,042

Tabela 5.1: Resultado da interpolação da velocidade para malha composta.

Malha	$h_{maisfino}$	Malhas no nível mais fino	$\|\|\mathbf{u}_N - \mathbf{u}_E\|\|_\infty$	Razão
32	$3,125\text{x}10^{-2}$	1	$7,808\text{x}10^{-4}$	-
64	$1,563\text{x}10^{-2}$	1	$1,951\text{x}10^{-4}$	4,002
128	$7,812\text{x}10^{-3}$	1	$4,868\text{x}10^{-5}$	4,008
256	$3,906\text{x}10^{-3}$	1	$1,204\text{x}10^{-5}$	4,042

Tabela 5.2: Resultado da interpolação da velocidade para malha uniforme.

Os resultados da Tabela 5.2 são idênticos àqueles obtidos na Tabela 5.1, o que confirma que a interpolação da velocidade só depende do espaçamento da malha no seu nível mais fino, independente de se utilizar malha composta ou uniforme.

Ressalta-se que neste estudo, as células fantasmas que compõem as malhas compostas eulerianas, foram utilizadas para interpolar a velocidade daqueles pontos da superfície que se encontram próximos ao bordo destas malhas. Por sua vez, a velocidade de cada célula fantasma é feita por interpolação, para maiores detalhes consultar [13].

5.2 Validação da Integração no Tempo

Para o processo de validação do método de integração numérica, utiliza-se para representar a fronteira imersa a mesma esfera usada na validação da interpolação de velocidade, Figura 5.1.

Neste estudo, utiliza-se domínio físico Ω=$[-0,5\ ,\ 17,5]$x$[-0,5\ ,\ 17,5]$x$[-0,5\ ,\ 17,5]$, um campo de velocidades dado por $\mathbf{u}_E = (t+0,1,\ t+0,1,\ t+0,1)$, uma malha composta do tipo 64L3 e o intervalo de tempo igual a [0, 1].

Malha	Divisões	ΔT	$\|\|\mathbf{x}_N - \mathbf{x}_E\|\|_\infty$	Razão
64L3	40	$2,500\text{x}10^{-2}$	$2,165\text{x}10^{-2}$	-
64L3	80	$1,250\text{x}10^{-2}$	$1,083\text{x}10^{-2}$	2,000
64L3	160	$6,250\text{x}10^{-3}$	$5,413\text{x}10^{-3}$	2,000
64L3	320	$3,125\text{x}10^{-3}$	$2,698\text{x}10^{-3}$	2,000

Tabela 5.3: Resultado da integração no tempo para malha composta.

Para estudar o processo de convergência entre a posição exata dos vértices da superfícies e a aquela calculada pelo processo de integração, na medida em que se diminui o intervalo de tempo da integração, divide-se inicialmente o intervalo de tempo em quarenta partes iguais e se executa o processo de integração no tempo. Na seqüência, se repete o procedimento anterior, dividindo-se o intervalo de tempo em um número partes iguais ao dobro daquele utilizado na etapa anterior. Durante todo este processo, se acompanha a relação entre as normas do máximo da diferença entre as posições exatas dos vértices e aquelas obtidas em todas as etapas do processo de integração.

Na Tabela 5.3 tem-se na primeira coluna o refinamento da malha; na segunda, o número de divisões do intervalo de tempo; na terceira, o tamanho do intervalo de tempo; na quarta, a norma do máximo entre a posição exata dos vértices da superfície e aquela calculada pelo processo de integração e na quinta, a razão entre duas normas consecutivas.

A validação do processo de integração numérica pode ser observado pelos resultados apresentados na quinta coluna da Tabela 5.3, isto é, os valores convergem para 2, à medida em que o número de divisões do intervalo de tempo vai dobrando (segunda coluna) o que confirma que o Método de Integração de Euler Explícito tem ordem 1.

5.3 Outros Testes

Nesta Seção, apresenta-se algumas aplicações de movimentação da superfície, na quais se varia a forma geométrica da superfície, o campo de velocidades do fluido, o número de superfícies movimentadas e também são exibidas as malhas euleriana e lagrangeana durante esta movimentação.

Simulação 01	
Parâmetros	**Valores**
Fronteira Imersa	Superfície Esférica
Número de Vértices	10.107
Número de Arestas	20.210
Número de Faces	30.315
Comprimento da Maior Aresta	$2,196\text{x}10^{-2}$
Comprimento da Menor Aresta	$6,949\text{x}10^{-3}$
Comprimento da Médio da Aresta	$1,163\text{x}10^{-3}$
Domínio Físico	Ω=[−0,5 , 10,5]x[−0,5 , 0,5]x[−0,5 , 0,5]
Tipo de Malha Composta	16L3
Espaçamento da Malha no Nível mais Fino	$4,297\text{x}10^{-2}$ x $3,906\text{x}10^{-2}$ x $3,906\text{x}10^{-2}$
Intervalo de Tempo	[0,5]
Número de Passos no Tempo	100
Campo de Velocidade	(1,0,0)

Tabela 5.4: Parâmetros utilizados na primeira simulação.

A primeira simulação tem como objetivo movimentar a superfície de uma esfera, Figura 5.3, utilizando-se um campo de velocidades que a faça se mover em uma única direção e sem que haja deformação da superfície (movimento de corpo rígido). As principais informações desta superfície, bem como os parâmetros utilizados nesta simulação, podem ser encontrados na Tabela 5.4

Os resultados da primeira simulação são apresentados da Figura de 5.4 à Figura 5.6. Para permitir a visualização de todo o domínio computacional, os eixos coordenados foram representados em diferentes escalas, pois a dimensão do domínio na direção X (direção do movimento) é 25 vezes maior que as outras duas dimensões. Com esta mudança de escala, as malhas eulerianas e a superfície da esfera aparecem distorcidas na Figura 5.4 e na Figura 5.5.

Na Figura 5.4 e na Figura 5.5, podem ser vistos os três níveis da malha euleriana 64L3 e também a malha lagrangeana nos instantes inicial e final da simulação. É importante observar que as malhas dos níveis 2 (intermediário) e 3 (mais fino) somente são construídas nas vizinhanças da superfície e não em todo domínio, o que reduz consideravelmente a memória

Simulação 02	
Parâmetros	**Valores**
Fronteira Imersa	Superfície Esférica
Número de Vértices	10.107
Número de Arestas	20.210
Número de Faces	30.315
Comprimento da Maior Aresta	$2,196\text{x}10^{-2}$
Comprimento da Menor Aresta	$6,949\text{x}10^{-3}$
Comprimento da Médio da Aresta	$1,163\text{x}10^{-3}$
Domínio Físico	Ω=[−0,5,2]x[−0,5,2]x[−0,5,2]
Tipo de Malha Composta	16L3
Espaçamento da Malha no Nível mais Fino	$9,766\text{x}10^{-3}$
Intervalo de Tempo	[0, 10]
Número de Passos no Tempo	100
Campo de Velocidade	$(cos(x+y+z),\ cos(x+y+z),\ cos(x+y+z))$

Tabela 5.5: Parâmetros utilizados na segunda simulação.

computacional utilizada.

Com o objetivo de mostrar, durante um certo intervalo de tempo, a movimentação da malha euleriana que acompanha e envolve a superfície esférica se movendo, apresenta-se na Figura 5.6, cortes da malha euleriana feitas por um plano perpendicular à direção do movimento. Da esquerda para a direita, de cima para baixo, pode-se observar que o corte da malha euleriana pelo plano (X=4.0) apresenta no primeiro e no oitavo quadros, dois níveis de refinamento; do segundo ao sétimo quadros, três níveis de refinamento e no último quadro, um nível de refinamento.

A segunda simulação tem com objetivo movimentar a superfície de uma esfera, Figura 5.3, utilizando-se um campo de velocidades que faça a superfície se deformar durante sua movimentação. As principais informações desta superfície, bem como os parâmetros utilizados nesta simulação, podem ser encontrados na Tabela 5.5.

Os resultados da segunda simulação são apresentados da Figura 5.7 à Figura 5.9. Na Figura 5.7 e na Figura 5.8, podem ser vistos os três níveis da malha euleriana 64L3 e também

a malha lagrangeana nos instantes inicial e final da simulação. Na Figura 5.9 pode ser visto um ampliação da terceiro nível da malha eulerina e da malha lagrangeana nos instantes inicial e final da simulação. As malhas eulerianas do nível mais fino que aparecem na Figura 5.9 estão com contornos em branco e a malha lagrangeana aparece em preto, para facilitar a visualização das mesmas.

A terceira simulação tem como objetivo movimentar, ao mesmo tempo, as superfícies de um octaedro e de um paralelepípedo, Figura 5.10, utilizando-se um campo de velocidades que as façam se mover periodicamente em círculos, sem se deformarem (movimento de corpo rígido). As principais informações destas superfícies, bem como os parâmetros utilizados nesta simulação, podem ser encontrados na Tabela 5.6.

Os resultados da terceira simulação são apresentados da Figura 5.11 à Figura 5.13. Na Figura 5.11 e na Figura 5.12, podem ser vistos os três níveis da malha euleriana 64L3 e também a duas malhas lagrangeanas (superfície do paralelepípedo e do octaedro) nos instantes inicial e final da simulação. Pode-se observar na Figura 5.13 uma ampliação da terceiro nível da malha eulerina e da malha lagrangeana no instante final da simulação. As malhas eulerianas do nível mais fino que aparecem na Figura 5.13 estão com contornos em branco e a malha lagrangeana aparece em preto, para facilitar a visualização das mesmas.

Simulação 03	
Parâmetros	**Valores**
Fronteira Imersa	Superfície do Octaedro
Número de Vértices	5.779
Número de Arestas	17.331
Número de Faces	11.554
Comprimento da Maior Aresta	$3,975\text{x}10^{-2}$
Comprimento da Menor Aresta	$1,673\text{x}10^{-2}$
Comprimento da Médio da Aresta	$2,677\text{x}10^{-2}$
Fronteira Imersa	Superfície do Paralelepípedo
Número de Vértices	6.628
Número de Arestas	9.942
Número de Faces	3.316
Comprimento da Maior Aresta	$1,190\text{x}10^{-1}$
Comprimento da Menor Aresta	$4,829\text{x}10^{-2}$
Comprimento da Médio da Aresta	$8,070\text{x}10^{-2}$
Domínio Físico	Ω=[−8 , 8] x[−8 , 8]x[−2 , 8]
Tipo de Malha Composta	64L3
Espaçamento da Malha no Nível mais Fino	$6,250\text{x}10^{-2}$ x $6,250\text{x}10^{-2}$ x $3,906\text{x}10^{-2}$
Intervalo de Tempo	[0, 10]
Número de Passos no Tempo	100
Campo de Velocidade	$(sin(\frac{\pi}{2}t),\ cos(\frac{\pi}{2}t),\ 0)$

Tabela 5.6: Parâmetros utilizados na terceira simulação.

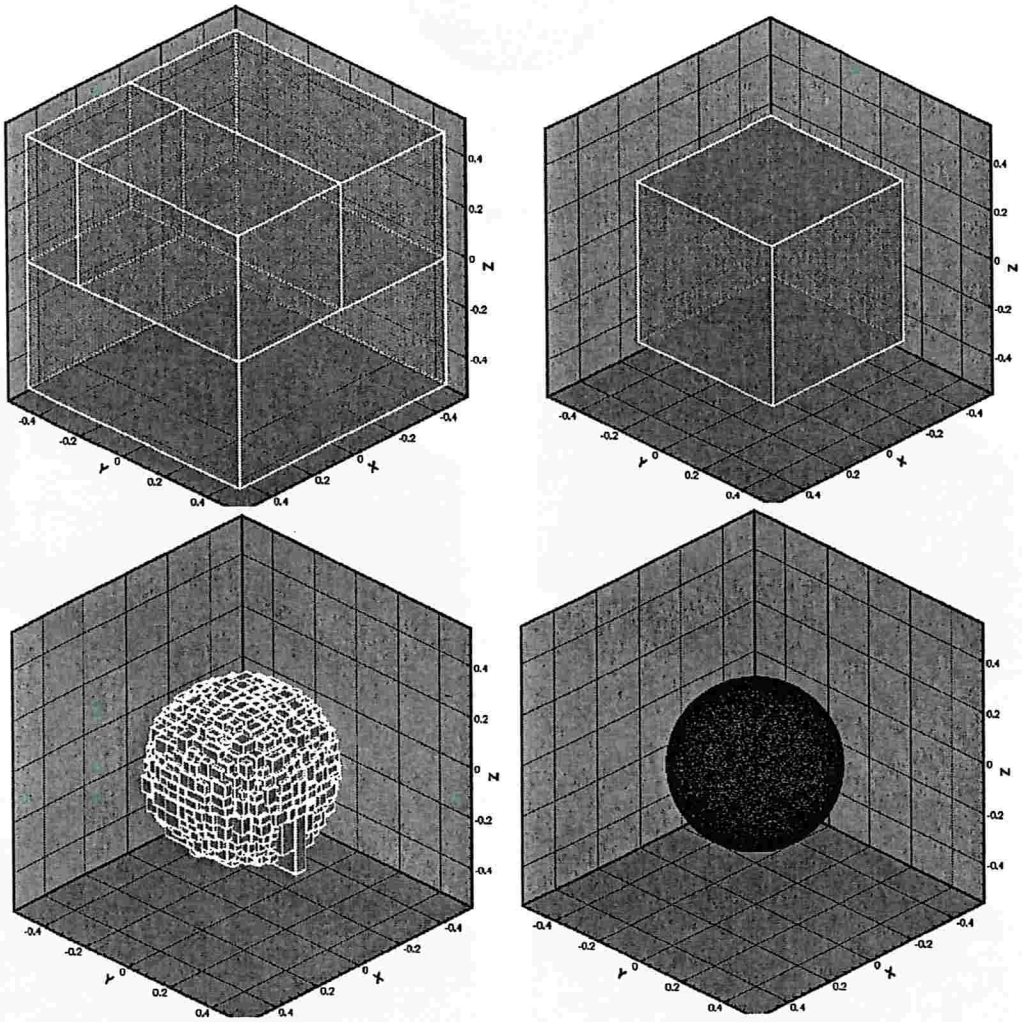


Figura 5.2: Da esquerda para a direita, de cima para baixo, visualização dos três níveis da malha euleriana 64L3, cujos espaçamentos são: $1,563\text{x}10^{-2}$, $7,812\text{x}10^{-3}$ e $3,906\text{x}10^{-3}$ e da malha lagrangeana que representa a esfera.

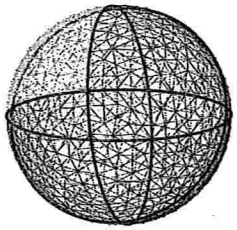


Figura 5.3: Esfera utilizada nas duas primeiras simulações.

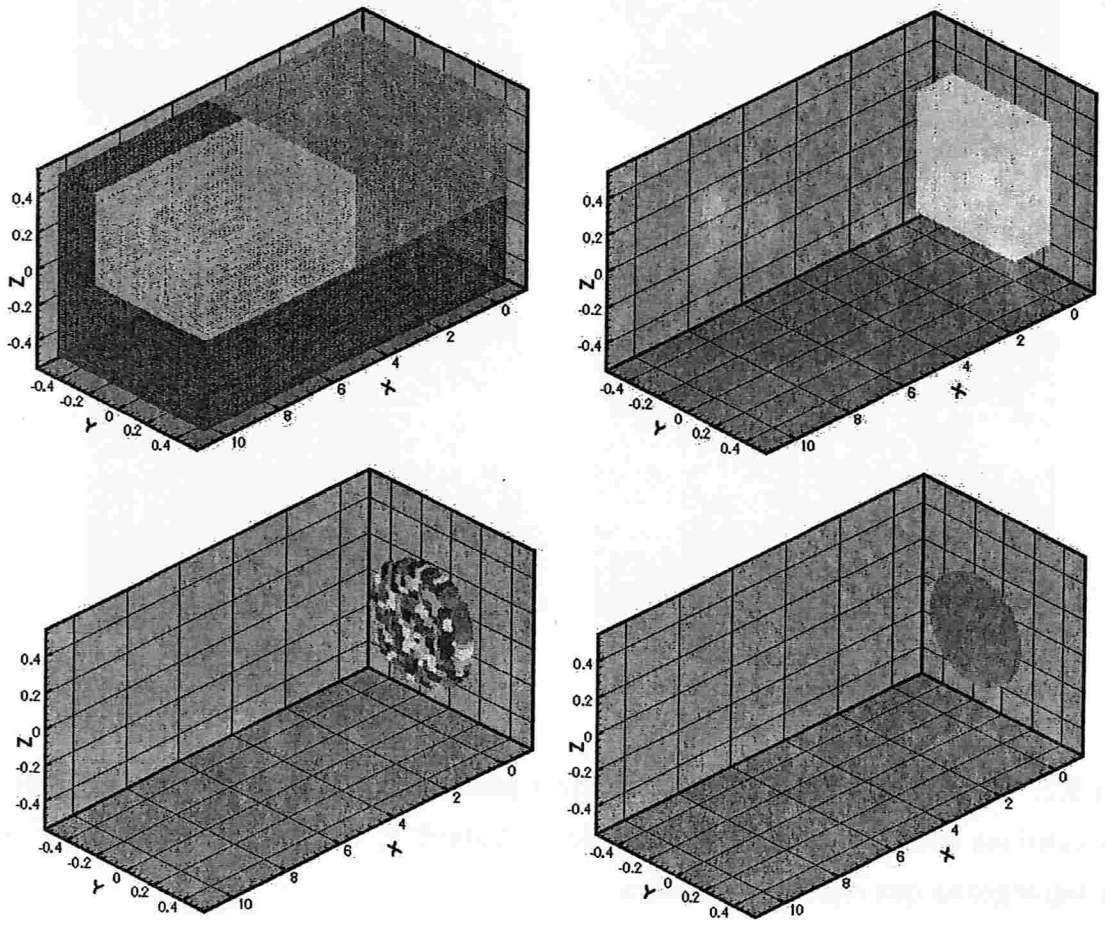


Figura 5.4: Da esquerda para a direita, de cima para baixo, estão representados o primeiro, o segundo e o terceiro nível da malha euleriana 64L3 e a malha lagrangeana que representa a superfície da esfera, no instante inicial $t = 0$.

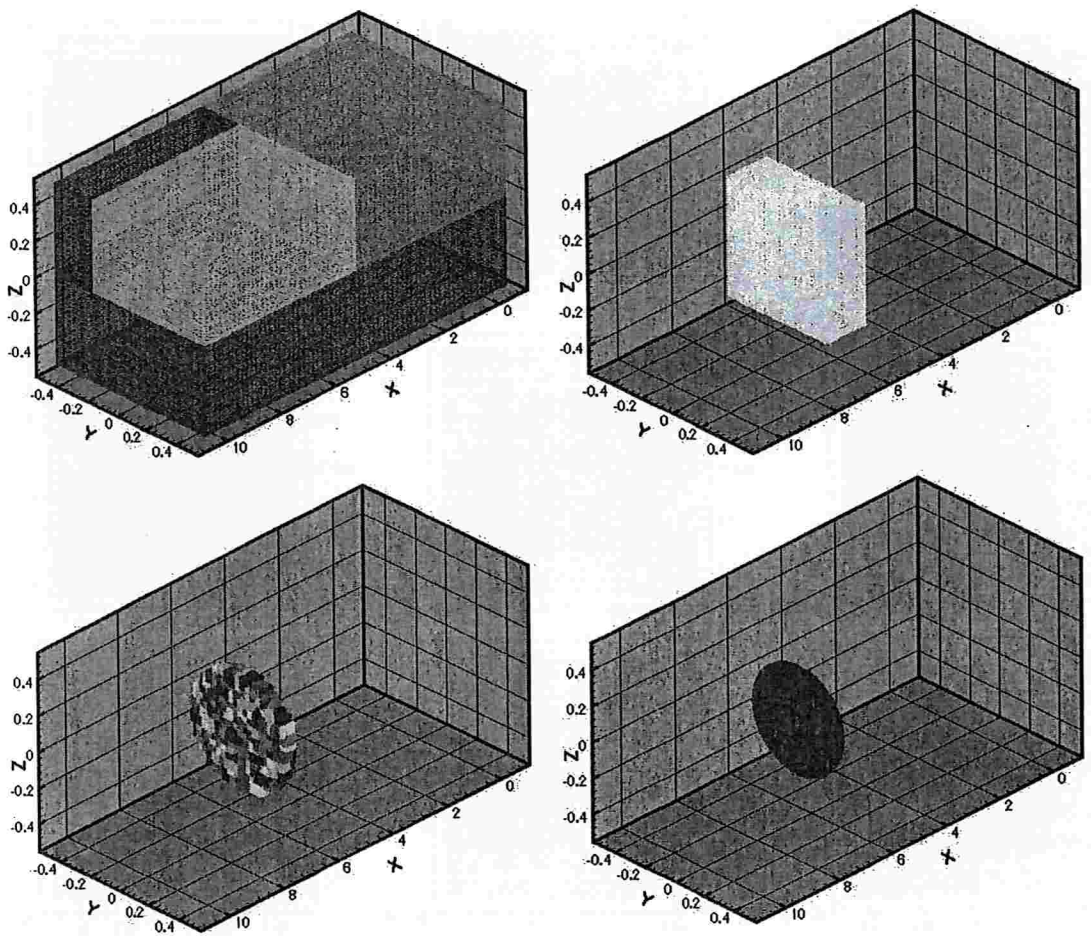


Figura 5.5: Da esquerda para a direita, de cima para baixo, estão representados o primeiro, o segundo e o terceiro nível da malha euleriana 64L3 e a malha lagrangeana que representa a superfície da esfera, no instante final $t = 5$.

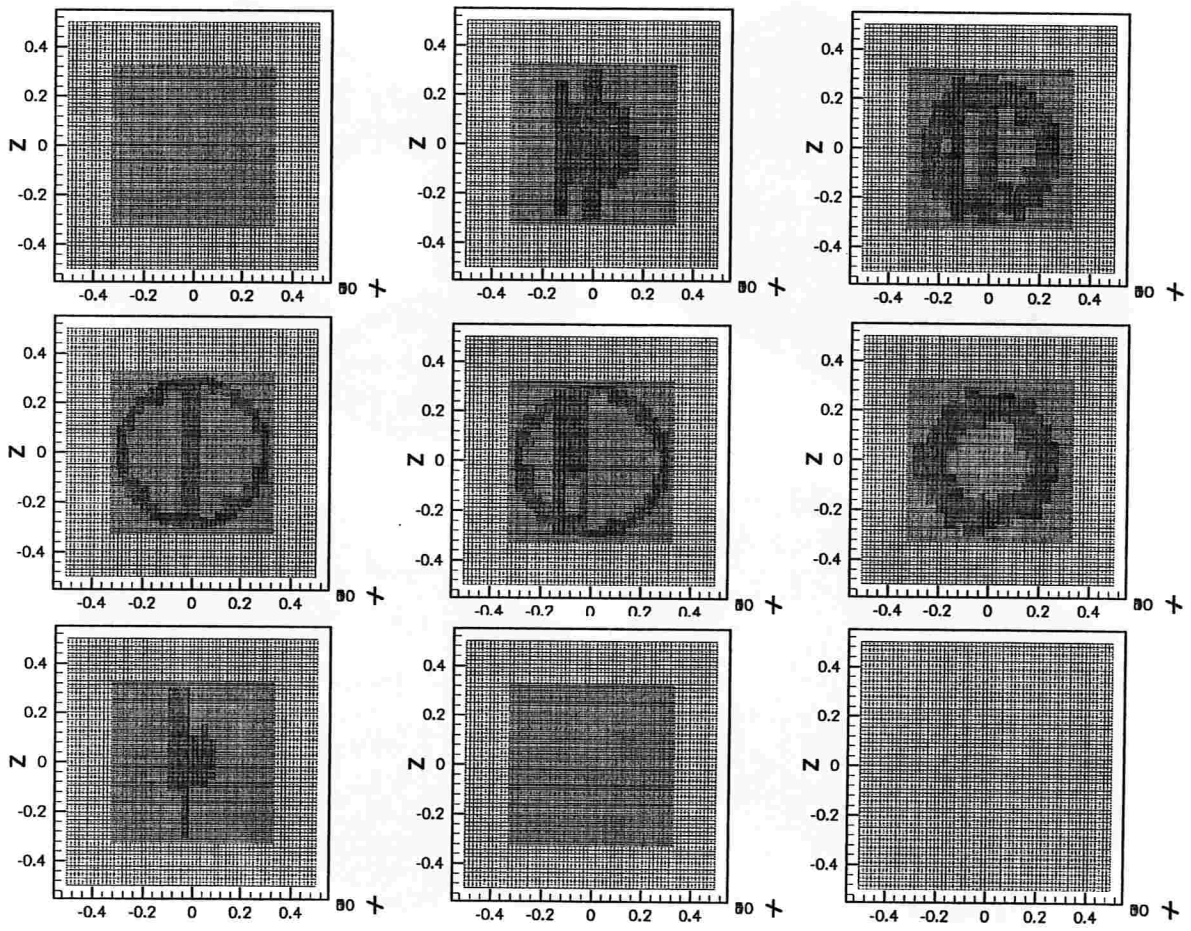


Figura 5.6: Da esquerda para a direita, de cima para baixo, estão representados cortes paralelos ao plano X=0.4, da malha euleriana 64L3, no instantes: $0,0$, $0,2$, $0,4$, $0,6$, $0,8$, $1,0$, $1,2$, $1,4$ e $1,8$.

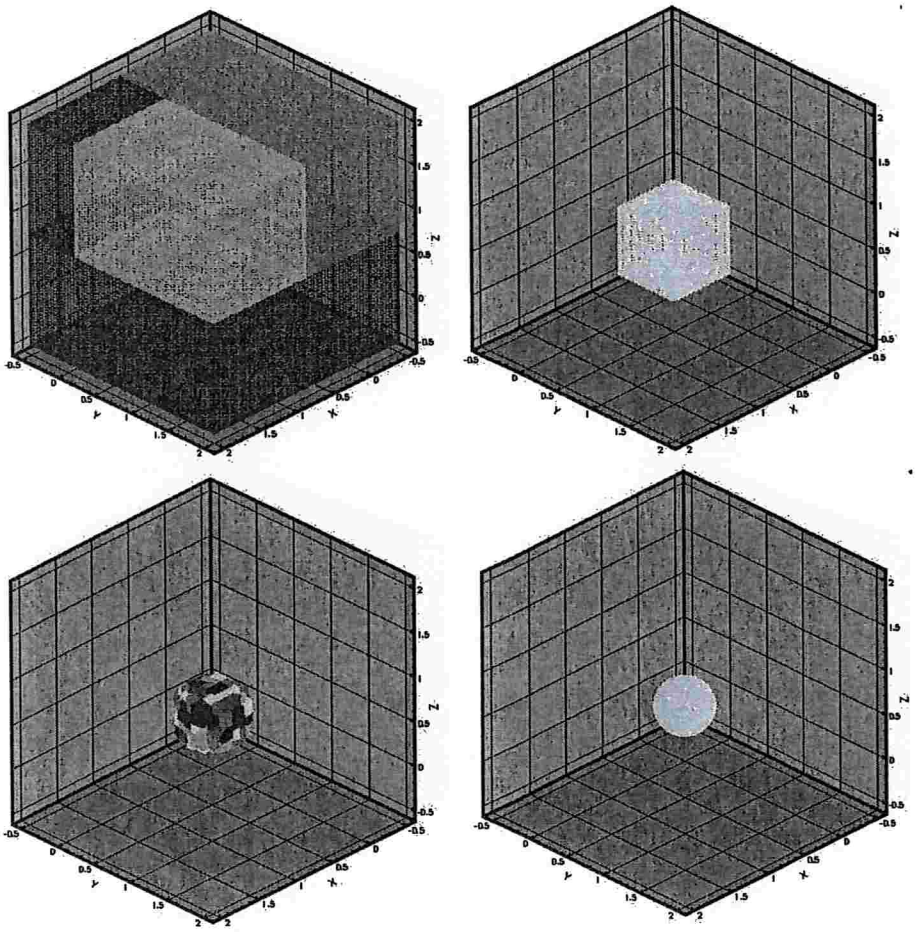


Figura 5.7: Da esquerda para a direita, de cima para baixo, estão representados o primeiro, o segundo e o terceiro nível da malha euleriana 64L3 e a malha lagrangeana que representa a superfície da esfera, no instante inicial $t = 0$.

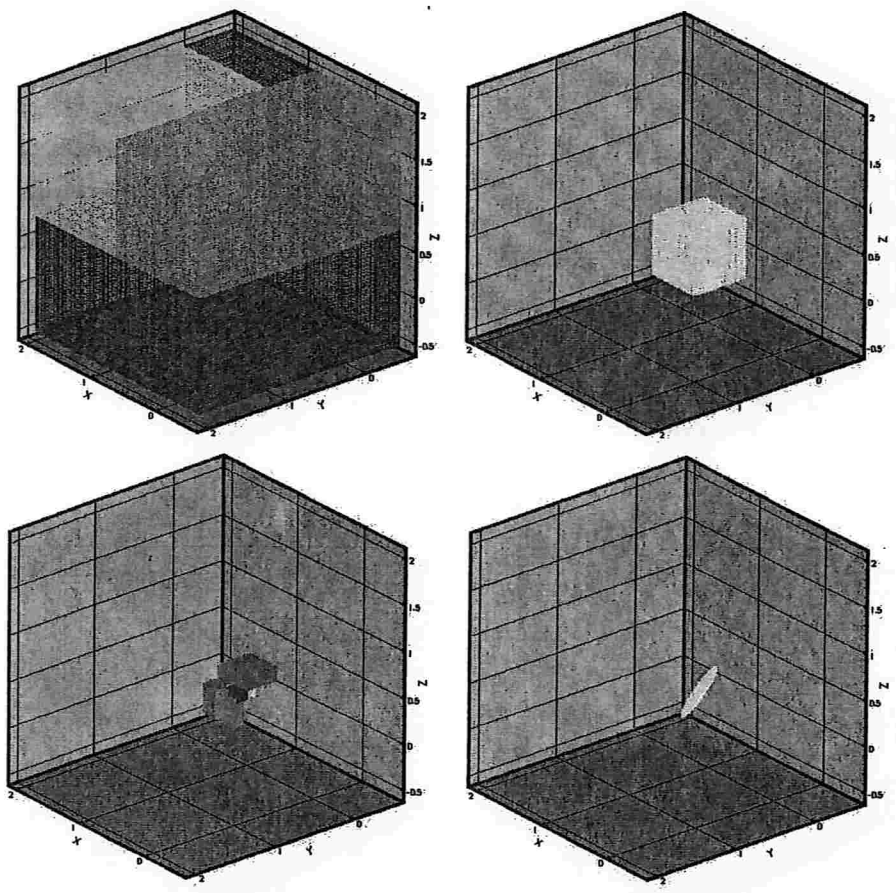


Figura 5.8: Da esquerda para a direita, de cima para baixo, estão representados o primeiro, o segundo e o terceiro nível da malha euleriana 64L3 e a malha lagrangeana que representa a superfície da esfera, no instante final $t = 10$.

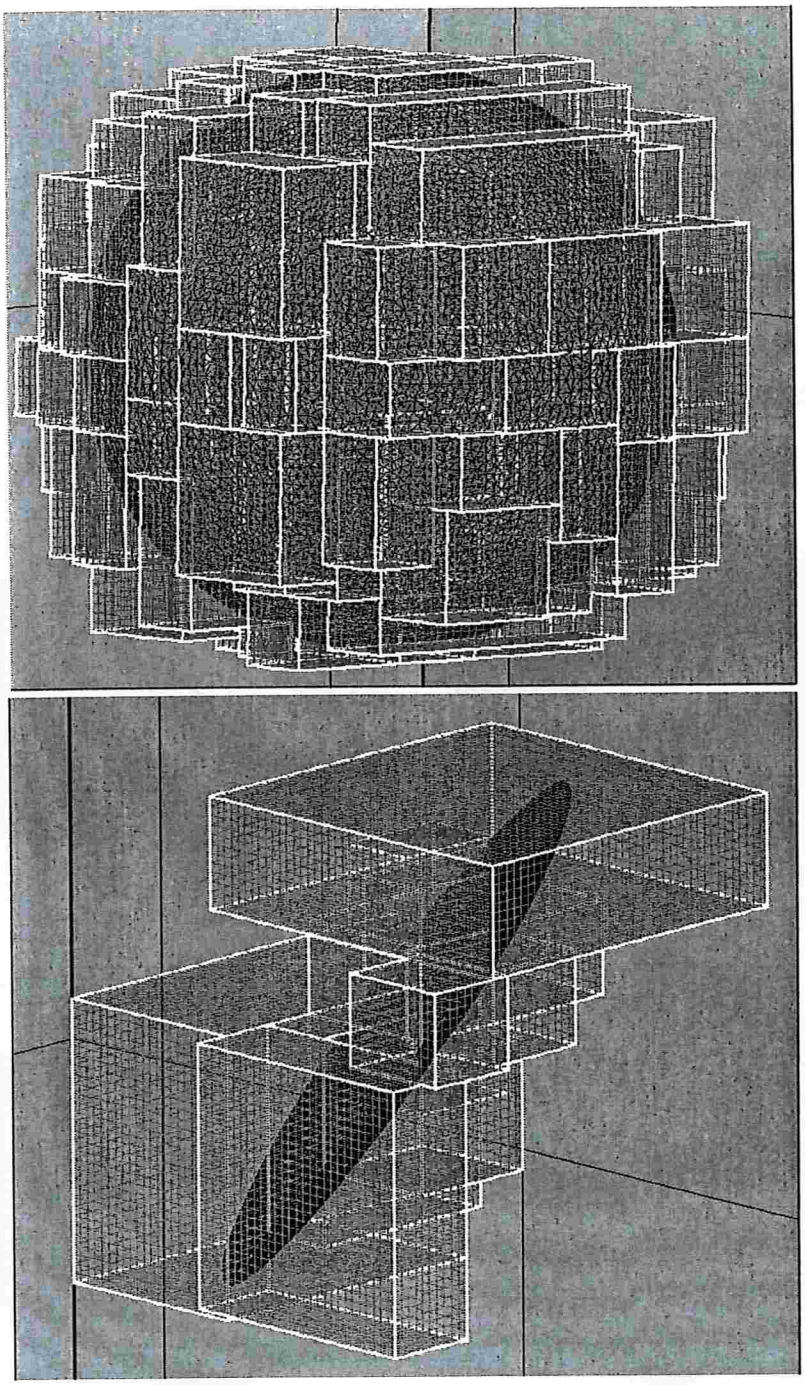


Figura 5.9: De cima para baixo, estão representados a ampliação do terceiro nível da malha euleriana 64L3 (com contorno em branco) e da malha lagrangeana (em preto) que representa a superfície da esfera, nos instantes inicial e final da simulação.

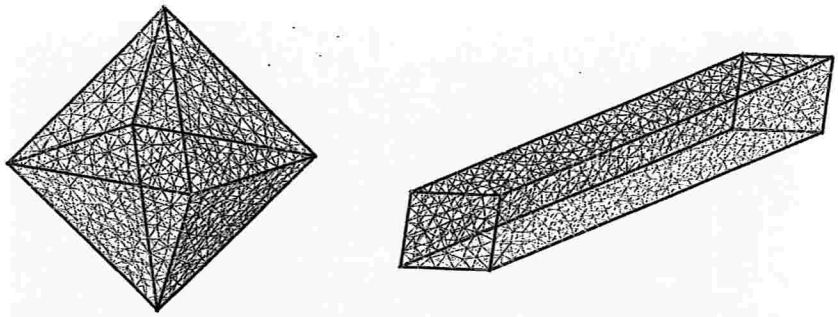


Figura 5.10: Octaedro e o paralelepípedo utilizados na terceira simulação.

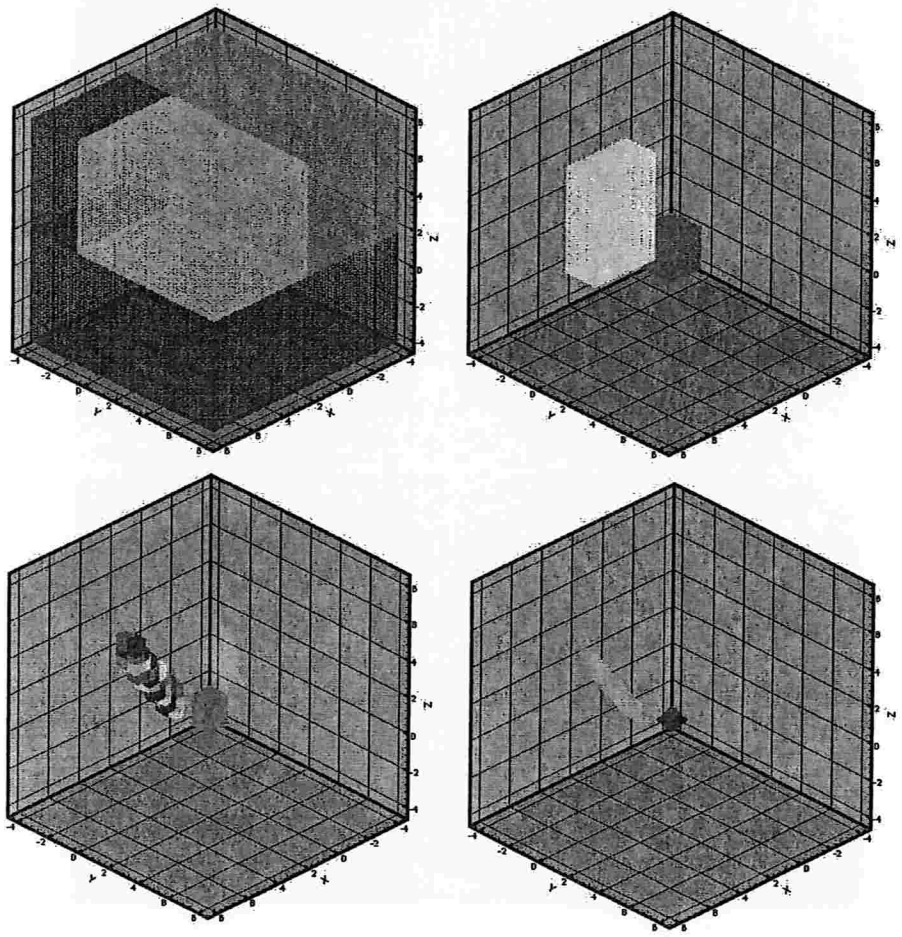


Figura 5.11: Da esquerda para a direita, de cima para baixo, estão representados o primeiro, o segundo e o terceiro nível da malha euleriana 64L3 e as malhas lagrangeanas que representam as superfícies do paralelepípedo e do octaedro, no instante inicial $t = 0$.

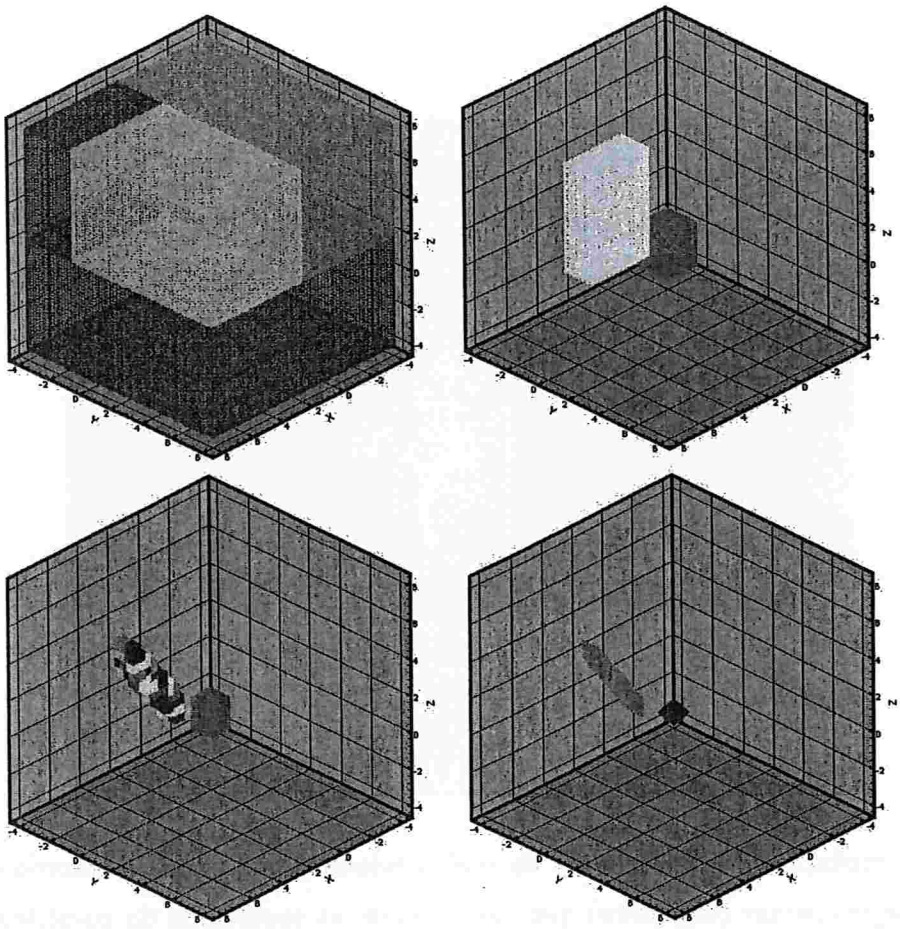


Figura 5.12: Da esquerda para a direita, de cima para baixo, estão representados o primeiro, o segundo e o terceiro nível da malha euleriana 64L3 e as malhas lagrangeanas que representam as superfícies do paralelepípedo e do octaedro, no instante final $t = 10$.

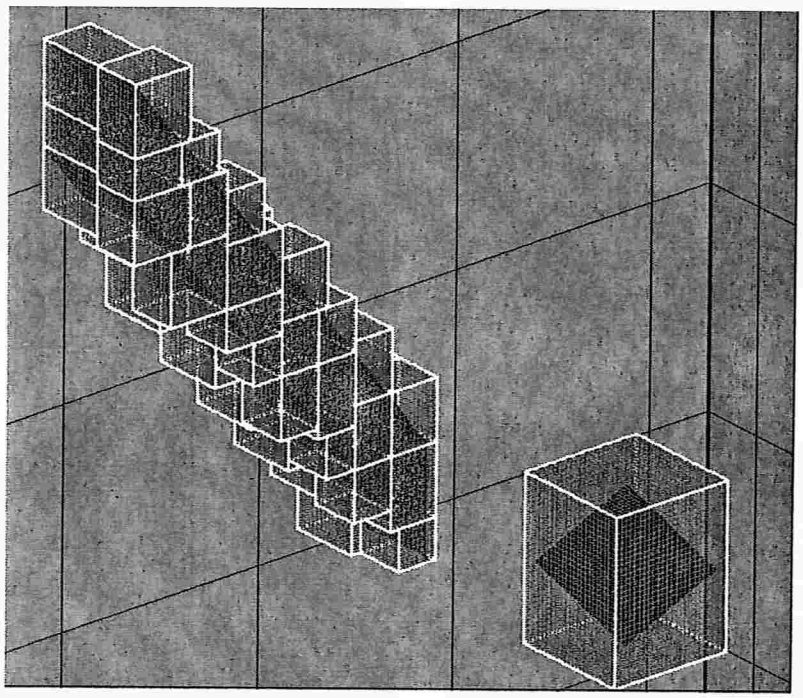


Figura 5.13: Ampliação do terceiro nível da malha euleriana 64L3 (com contorno em branco) e as malhas lagrangeanas (em preto) que representam as superfícies do paralelepípedo e do octaedro, no instante final da simulação.

CAPÍTULO 6

Conclusão

O objetivo deste trabalho, era explorar uma ferramenta que permitisse construir uma grande variedade de fronteiras imersas e também implementar uma estrutura de dados que permitisse armazenar, convenientemente, as informações desta fronteira.

O software GMSH, utilizado para a construção das fronteiras imersas deste trabalho, se mostra bastante promissor na construção de superfícies com geometrias mais complexas, permitindo que uma gama maior de problemas de mecânica dos fluidos computacional possam ser modelados de uma forma simples.

Com relação a estrutura de dados Halfedge, utilizada para representar as superfíces de sólidos, e que foi implementada em FORTRAN 90, atendeu bem ao proposito inicial deste trabalho, isto é, ela é uma estrutura que ocupa pouco memória computacional e permite que se obtenha rapidamente informações de adjacências sobre os elementos que representam a superfície do sólido. Entretanto, outras estruturas de dados mais eficientes devem ser pesquisadas para cobrir a deficiência desta estrutura, quanto a representação de superfícies não orientáveis e quanto ao uso de memória e tempo de resposta computacional.

A pequena parcela de contribuição deste trabalho no estudo da mecânica de fluidos computacional e servir de base para estudos mais complexos envolvendo aplicações com fronteira imersa. Para estes estudos, se faz necessário encontrar uma representação computacional adequada para a fronteira imersa, que permita ao mesmo tempo utilizar pouca memória computacional e que tenha rapidez de processamento das informações.

APÊNDICE A

Malhas Geradas pelo GMSH

Neste Apêndice, são mostrados três sólidos: Cubo Vazado, Esfera e Octaedro e as malhas triangulares sobre suas superfícies; além dos respectivos programas utilizados para gerar a geometria de cada sólido, escritos na meta-linguagem do GMSH.

Primeiramente, mostra-se o cubo vazado e as malhas geradas pelo GMSH:

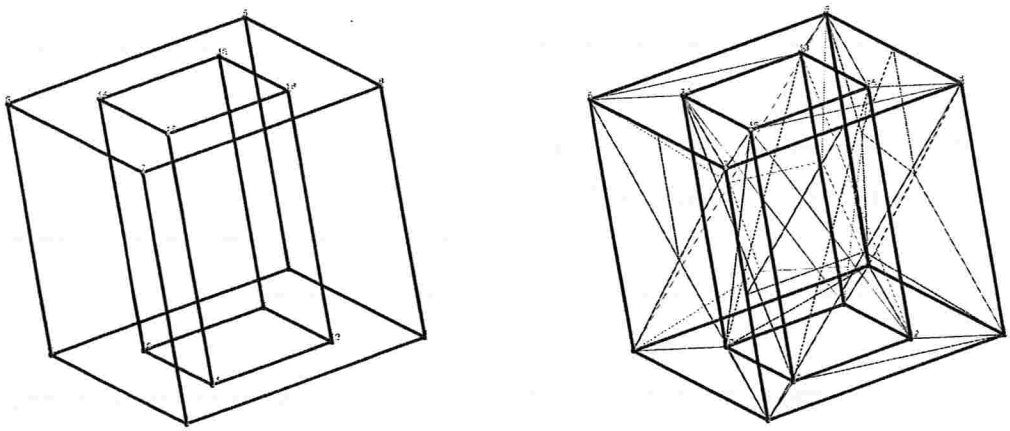


Figura A.1: Geometria do cubo vazado (à esquerda) ; Malha triangular para lc=0.5, com 62 triângulos gerados (à direita).

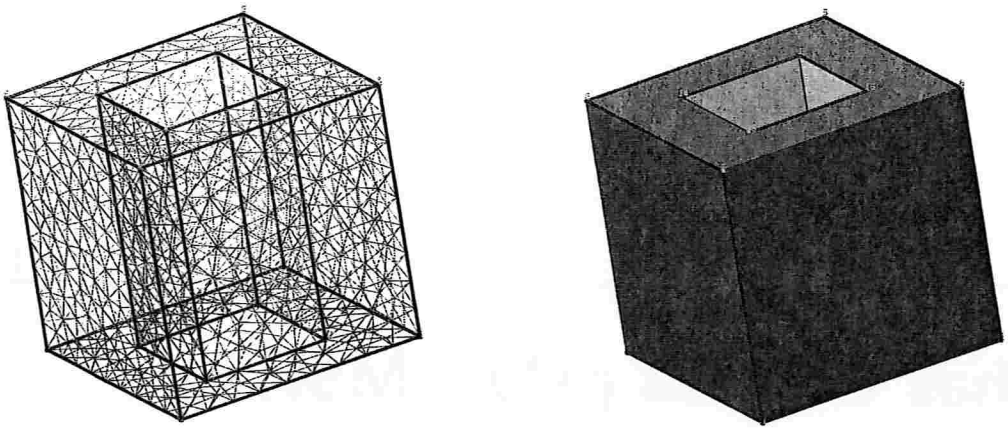


Figura A.2: Malha triangular para lc=0.1, com 1650 triângulos gerados (à esquerda) ; Cubo vazado renderizado (à direita).

Apresenta-se a seguir, uma cópia do programa escrito na meta-linguagem do GMSH, utilizado para construir a geometria do cubo vazado de arestas externas unitárias e arestas internas medindo 0.75,

```
//--------------------------------------------------------------------
//
//  Desenho da Geometria do Cubo Vazado
//  Data da Última Alteração: 01/02/2006
//
//--------------------------------------------------------------------

// A variável lc permitira definir indiretamente o número de triângulos
// que serão gerados pelo programa. Quanto menor o valor de lc,
// maior será o número de triângulos gerados.

lc=0.5;

// Definiremos as coordenadas dos pontos de cada um dos 8 vértices
// externos do cubo vazado.

Point(1) = {0,0,0,lc};
```

```
Point(2) = {1,0,0,lc};
Point(3) = {1,1,0,lc};
Point(4) = {0,1,0,lc};
Point(5) = {0,0,1,lc};
Point(6) = {1,0,1,lc};
Point(7) = {1,1,1,lc};
Point(8) = {0,1,1,lc};

// Definiremos as coordenadas dos pontos de cada um dos 8 vértices
// internos do cubo vazado.

Point(9) = {0.25,0.25,0,lc};
Point(10) = {0.75,0.25,0,lc};
Point(11) = {0.75,0.75,0,lc};
Point(12) = {0.25,0.75,0,lc};
Point(13) = {0.25,0.25,1,lc};
Point(14) = {0.75,0.25,1,lc};
Point(15) = {0.75,0.75,1,lc};
Point(16) = {0.25,0.75,1,lc};

// Definiremos os vértices inicial e final de cada uma das 12 arestas
// externas do cubo vazado.

Line(1) = {1,2};
Line(2) = {2,3};
Line(3) = {3,4};
Line(4) = {4,1};
Line(5) = {1,5};
Line(6) = {5,6};
Line(7) = {6,2};
Line(8) = {5,8};
Line(9) = {8,7};
Line(10) = {7,6};
```

```
Line(11) = {3,7};
Line(12) = {4,8};

// Definiremos os vértices inicial e final de cada uma das 12 arestas
// internas do cubo vazado.

Line(13) = {9,10};
Line(14) = {10,11};
Line(15) = {11,12};
Line(16) = {12,9};
Line(17) = {9,13};
Line(18) = {13,14};
Line(19) = {14,10};
Line(20) = {13,16};
Line(21) = {16,15};
Line(22) = {15,14};
Line(23) = {11,15};
Line(24) = {12,16};

// O sentido de percurso do loop das arestas de cada uma das faces foi
// escolhido de tal forma a termos o vetor normal apontando sempre para
// fora da face.
// Definiremos as arestas que formarão cada uma das 4 faces externas
// do cubo vazado.

Line Loop(25) = {22,19,14,23};    Plane Surface(26) = {25};
Line Loop(27) = {21,-23,15,24};   Plane Surface(28) = {27};
Line Loop(29) = {20,-24,16,17};   Plane Surface(30) = {29};
Line Loop(31) = {18,19,-13,17};   Plane Surface(32) = {31};

// Definiremos as arestas que formarão cada uma das 4 faces internas
// do cubo vazado.
```

```
Line Loop(33) = {10,7,2,11};      Plane Surface(34) = {33};
Line Loop(35) = {9,-11,3,12};  Plane Surface(36) = {35};
Line Loop(37) = {8,-12,4,5};   Plane Surface(38) = {37};
Line Loop(39) = {6,7,-1,5};    Plane Surface(40) = {39};

// Definiremos as arestas que formarão cada uma das 2 faces: superior
// e inferior. Foram utilizados dois loops para definir cada uma das duas
// faces, já que existem buracos nelas. O primeiro loop representa a
// fronteira interna e o segundo a fronteira interna de cada face.

Line Loop(41) = {22,-18,20,21}; Line Loop(42) = {10,-6,8,9};
Plane Surface(43) = {42,41};

Line Loop(44) = {14,15,16,13}; Line Loop(45) = {2,3,4,1};
Plane Surface(46) = {45,44};

// Definiremos o cubo vazado como um sólido formado por um conjunto
// de 10 faces: quatro externas, quatro internas, uma superior e
// outra inferior.

Surface Loop(47) = {26,28,30,32,34,36,38,40,43,46};      Volume(48) ={47};
```

Agora é mostrado a esfera e as malhas triangulares geradas pelo GMSH:

Figura A.3: Geometria da esfera (à esquerda) ; Malha triangular para lc=0.5, com 72 triângulos gerados (à direita).

Apresenta-se a seguir, uma cópia do programa escrito na meta-linguagem do GMSH, utilizado para construir a geometria da esfera de raio unitário e centro na origem. A esfera apresenta uma particularidade em relação aos outros dois sólidos aqui construídos. O GMSH, no caso da esfera, ao gerar o arquivo de saida da malha no formato descrito no Apêndice B, começa a enumerar os vértices a partir do número 2, reservando internamente o número 1 para o centro da esfera. Nos demais sólidos, a enumeração dos vértices começa a partir do número 1. No programa FORTRAN 90, utilizado para fazer a leitura dos dados de saída do GMSH, os vértices da esfera são automaticamente reindexados, isto é, os vértices começam a ser contados a partir do número 1 como qualquer outro sólido.

```
//-------------------------------------------------------------
//
//  Desenho da Geometria de um Esfera
//  Data da Última Alteração: 03/02/2006
//
//-------------------------------------------------------------

// A variável lc permitira definir indiretamente o número de triângulos
// que serão gerados pelo programa. Quanto menor o valor de lc,
// maior será o número de triângulos gerados.
```

Figura A.4: Malha triangular para lc=0.1, com 2224 triângulos gerados (à esquerda) ; Esfera renderizada (à direita).

```
lc=0.5;

// Definiremos as coordenadas dos pontos de cada octante, sendo o
// ponto 1 o centro da esfera de raio unitário.

Point(1) = {0, 0, 0, lc};
Point(2) = {1, 0, 0, lc};
Point(3) = {0, 1, 0, lc};
Point(4) = {-1, 0, 0, lc};
Point(5) = {0, -1, 0, lc};
Point(6) = {0, 0, -1, lc};
Point(7) = {0, 0, 1, lc};

// Definiremos as coordenadas das semi-circunferências em cada octante.
// Os números laterais são os pontos onde as circunferências, devem
// passar e o ponto do meio é o centro de cada uma das semi-circunferências.

Circle (1) = {2, 1, 3};
Circle (2) = {3, 1, 4};
Circle (3) = {4, 1, 5};
```

```
Circle (4) = {5, 1, 2};
Circle (5) = {3, 1, 6};
Circle (6) = {6, 1, 5};
Circle (7) = {5, 1, 7};
Circle (8) = {7, 1, 3};
Circle (9) = {2, 1, 7};
Circle (10) = {7, 1, 4};
Circle (11) = {4, 1, 6};
Circle (12) = {6, 1, 2};

// Fazemos a rotação de cada uma das semi-circunferências anteriores em torno
// do eixos coordenados, gerando assim várias calotas esféricas.

Line Loop (13) = {2, -10, 8};   Ruled Surface (14) = {13};
Line Loop (15) = {10, 3, 7};    Ruled Surface (16) = {15};
Line Loop (17) = {-8, -9, 1};   Ruled Surface (18) = {17};
Line Loop (19) = {-11, -2, 5};  Ruled Surface (20) = {19};
Line Loop (21) = {-5, -1, -12}; Ruled Surface (22) = {21};
Line Loop (23) = {-3, 11, 6};   Ruled Surface (24) = {23};
Line Loop (25) = {-7, 4, 9};    Ruled Surface (26) = {25};
Line Loop (27) = {-4, -6, 12};  Ruled Surface (28) = {27};


// Definiremos a esfera como sendo um sólido formado por um conjunto
// de 8 calotas esféricas.

Surface Loop (29) = {28, 26, 16, 14, 20, 24, 22, 18};  Volume (30) = {29};
```

Finalmente é mostrado o octaedro e as malhas triangulares geradas pelo GMSH:

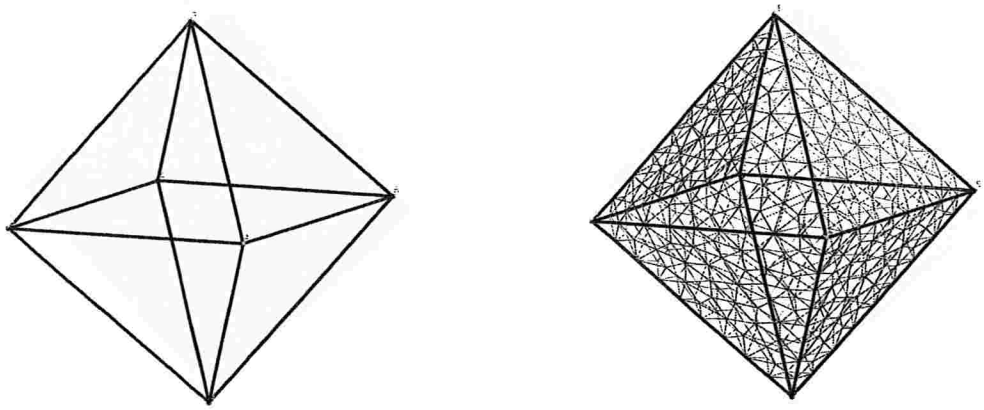


Figura A.5: Geometria do octaedro (à esquerda) ; Malha triangular para lc=0.1, com 744 triângulos gerados (à direita).

Apresenta-se a seguir, uma cópia do programa escrito na meta-linguagem do GMSH, utilizado para construir o octaedro de aresta unitária.

```
//-------------------------------------------------------------
//
//  Desenho da Geometria de um Octaedro.
//  Data da Última Alteração: 05/03/2006
//
//-------------------------------------------------------------

// A variável lc permitira definir indiretamente o número de triângulos
// que serão gerados pelo programa. Quanto menor o valor de lc,
// maior será o número de triângulos gerados.

lc=0.1;

// Definiremos as coordenadas dos pontos de cada um dos 6 vértices do octaedro.

Point(1) = {0.5, 0.5, 0.707106781186548, lc};
Point(2) = {0.0, 0.0, 0.0, lc};
Point(3) = {1.0, 0.0, 0.0, lc};
```

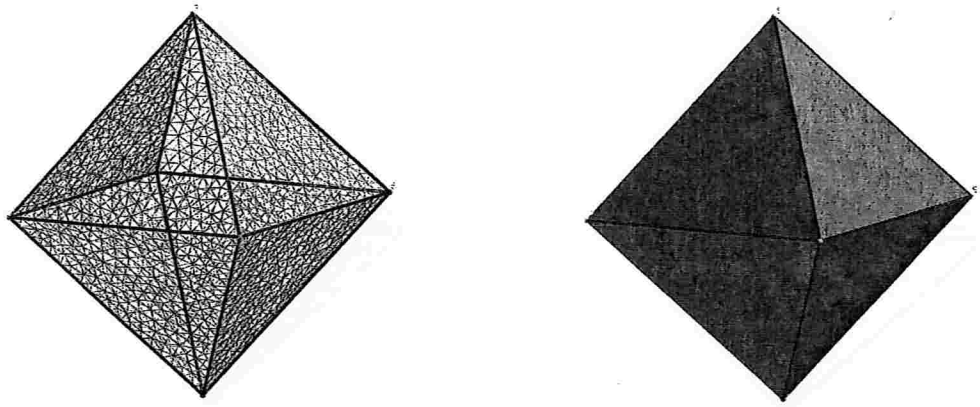


Figura A.6: Malha triangular para lc=0.05, com 2890 triângulos gerados (à esquerda) ; Octaedro renderizado (à direita).

```
Point(4) = {1.0, 1.0, 0.0, lc};
Point(5) = {0.0, 1.0, 0.0, lc};
Point(6) = {0.5, 0.5, -0.707106781186548, lc};

// Definiremos os vértices inicial e final de cada uma das 12 arestas
// do octaedro.

Line (1) = {2, 1};
Line (2) = {3, 1};
Line (3) = {4, 1};
Line (4) = {5, 1};
Line (5) = {2, 6};
Line (6) = {3, 6};
Line (7) = {4, 6};
Line (8) = {5, 6};
Line (9) = {2, 3};
Line (10) = {3, 4};
Line (11) = {4, 5};
Line (12) = {5, 2};
```

```
// Definiremos as arestas que formarão cada uma das 8 faces do octaedro.

Line Loop (13) = {-1, 9, 2};      Plane Surface (14) = {13};
Line Loop (15) = {-2, 10, 3};     Plane Surface (16) = {15};
Line Loop (17) = {-3, 11, 4};     Plane Surface (18) = {17};
Line Loop (19) = {-4, 12, 1};     Plane Surface (20) = {19};
Line Loop (21) = {5, -6, -9};     Plane Surface (22) = {21};
Line Loop (23) = {6, -7, -10};    Plane Surface (24) = {23};
Line Loop (25) = {-11, 7, -8};    Plane Surface (26) = {25};
Line Loop (27) = {-12, 8, -5};    Plane Surface (28) = {27};

// O sentido de percurso do loop das arestas de cada uma das faces foi
// escolhido de tal forma a termos o vetor normal apontando sempre para
// fora da face. Definiremos o octaedro como sendo um sólido formado
// por um conjunto de 8 faces.

Surface Loop (29) = {28, 26, 16, 14, 20, 24, 22, 18};   Volume (30) = {29};
```

Dados Gerados pelo GMSH

Neste Apêndice é apresentado o modelo do arquivo gerado pelo GMSH, com os dados da malha triangular do Cubo Vazado, utilizou-se lc=0.5. Este formato de arquivo é utilizado para alimentar a estrutura de dados Halfedge e seu conteúdo basicamente contém:

- Todos os vértices com suas respectivas coordenadas cartesianas.
- Todos os triângulos da malha representados por seus três vértices.

```
$NOD
31
1 0 0 0
2 1 0 0
3 1 1 0
4 0 1 0
5 0 0 1
6 1 0 1
7 1 1 1
8 0 1 1
9 0.25 0.25 0
10 0.75 0.25 0
11 0.75 0.75 0
12 0.25 0.75 0
13 0.25 0.25 1
```

```
14 0.75 0.25 1
15 0.75 0.75 1
16 0.25 0.75 1
17 0.75 0.5000093083449829 0.5000730171018372
18 0.5001235069791571 0.75 0.4999892684186042
19 0.25 0.500012806027227 0.5001135984398223
20 0.5000285728240533 0.25 0.4999224223513546
21 1 0.5000785307885491 0.5000692791896159
22 1 0.5000687042766009 1.000044046787429
23 0.5000189324414708 1 0.4998083798927854
24 0.5001244753994236 1 0.9997946516894973
25 1.000000653795106 1 0.4999339195792187
26 0 0.5000642015831038 0.5000494910003412
27 0 0.1250629366675016 0.5625732789470159
28 0 0.5000516888654455 1.000040630666678
29 0.5000924352463236 0 0.4999920487637453
30 0.4375710534736685 0 0.1249961942641477
31 0.117290688357783 0 0.4531083733736823
$ENDNOD
$ELM
86
1 1 1 1 2 1 2
2 1 2 2 2 2 3
3 1 3 3 2 3 4
4 1 4 4 2 4 1
5 1 5 5 2 1 5
6 1 6 6 2 5 6
7 1 7 7 2 6 2
8 1 8 8 2 5 8
9 1 9 9 2 8 7
10 1 10 10 2 7 6
11 1 11 11 2 3 7
12 1 12 12 2 4 8
```

```
13 1 13 13 2 9 10
14 1 14 14 2 10 11
15 1 15 15 2 11 12
16 1 16 16 2 12 9

.....

18 1 18 18 2 13 14
19 1 19 19 2 14 10
20 1 20 20 2 13 16
21 1 21 21 2 16 15
22 1 22 22 2 15 14
23 1 23 23 2 11 15
24 1 24 24 2 12 16
25 2 26 26 3 11 17 10
26 2 26 26 3 14 17 15
27 2 26 26 3 10 17 14
28 2 26 26 3 15 17 11
29 2 28 28 3 12 18 11
30 2 28 28 3 15 18 16
31 2 28 28 3 11 18 15
32 2 28 28 3 16 18 12
33 2 30 30 3 13 16 19
34 2 30 30 3 12 9 19
35 2 30 30 3 9 13 19

.....

82 2 46 46 3 1 9 12
83 2 46 46 3 2 3 10
84 2 46 46 3 10 3 11
85 2 46 46 3 1 10 9
86 2 46 46 3 11 4 12
```

```
ENDELM
```

Estrutura Halfedge

Neste Apêndice é mostrado, o diagrama que representa as principais relações, não todas, entre os diversos ponteiros da estrutura Halfedge e todos os campos desta estrutura de dados, escritos na linguagem FORTRAN 90.

Apresenta-se todos os campos da estrutura de dados Halfedge , escrito em FORTRAN 90, que são utilizados no programa de movimentação da fronteira imersa:

```
MODULE Half_Edge
 USE DATA_TYPES
 IMPLICIT NONE
 SAVE

 TYPE Vector
   DOUBLE PRECISION,DIMENSION(4) ::  Vec
 END TYPE

 TYPE Velocity
   DOUBLE PRECISION,DIMENSION(3) ::  Vel
 END TYPE

 TYPE GRID_COMPONENTS1
   INTEGER :: IX, IY, IZ, MX, MY, MZ, IU, BW, BE, BS, BN, BB, BT
 END TYPE GRID_COMPONENTS1
```

```
TYPE LEVEL_COMPONENTS1
  DOUBLE PRECISION :: HX, HY, HZ
  TYPE(GRID_COMPONENTS1) :: GRID
  TYPE(LEVEL_COMPONENTS1), POINTER :: NEXT
END TYPE LEVEL_COMPONENTS1

TYPE Solid
  INTEGER                :: IdSolid ! Identificador do sólido.
  TYPE(Face),POINTER     :: Sfaces  !Ponteiro para a lista de faces.
  TYPE(Edge),POINTER     :: Sedges  !Ponteiro para lista de arestas.
  TYPE(Vertex),POINTER   :: Sverts  !Ponteiro para a lista de vértices.
  TYPE(Solid),POINTER    :: Nexts   !Ponteiro para o próximo sólido.
  TYPE(Solid),POINTER    :: Prevs   !Ponteiro para o sólido anterior.
END TYPE Solid


TYPE Face
   INTEGER              :: IdFace !Identificador da face.
   TYPE(Solid),POINTER  :: Fsolid !Ponteiro para o sólido ao qual pertence a face.
   TYPE(Loop),POINTER   :: Flout  !Ponteiro para o loop externo da face.
   TYPE(Loop),POINTER   :: Floops !Ponteiro para a lista de loops da face.
   TYPE(Vector)         :: Feq    !Equação do plano da Face : ax+by+cz+d=0.
   TYPE(Face),POINTER   :: Nextf  !Ponteiro para a próxima face do sólido.
   TYPE(Face),POINTER   :: Prevf  !Ponteiro para a face anterior do sólido.
 END TYPE Face


 TYPE Loop
   TYPE(Halfedge),POINTER :: Ledg  !Ponteiro para uma das semi-arestas do Loop.
   TYPE(Face),POINTER     :: Lface !Ponteiro para a face ao qual pertence o Loop.
   TYPE(Loop),POINTER     :: Nextl !Ponteiro para o próximo "loop" da face.
   TYPE(Loop),POINTER     :: Prevl !Ponteiro para o "loop" anterior da face.
```

```
 END TYPE Loop


 TYPE Halfedge
   TYPE(Edge),POINTER  :: Edg !Ponteiro para aresta que é associada à semiaresta.
   TYPE(Vertex),POINTER :: Vtx !Ponteiro para o vértice final da semi-aresta.
   TYPE(Loop),POINTER::Wloop !Ponteiro para o Loop ao qual pertence à semiaresta.
   TYPE(Halfedge),POINTER :: Nexth !Ponteiro para próxima semi-aresta.
   TYPE(Halfedge),POINTER :: Prevh !Ponteiro para a semi-aresta anterior.
 END TYPE Halfedge

TYPE Edge
   INTEGER                  :: IdEdge  !Identificador da aresta.
   TYPE(Halfedge),POINTER   :: He1    !Ponteiro para semi-aresta direita.
   TYPE(Halfedge),POINTER   :: He2    !Ponteiro para a semi-aresta esquerda.
   TYPE(Edge),POINTER       :: Nexte   !Ponteiro para a próxima aresta.
   TYPE(Edge),POINTER       :: Preve   !Ponteiro para a aresta anterior.
END TYPE Edge


TYPE Vertex
  INTEGER                :: IdVertex !Identificador do vértice.
  TYPE(Halfedge),POINTER::Vedge!Ponteiro para a semiaresta que começa no vértice.
  TYPE(Vector)   :: Vcoord1  !Coordenadas cartesianas do vértice no instante t1.
  TYPE(Vector)   :: Vcoord2  !Coordenadas cartesianas do vértice no instante t2.
  TYPE(Velocity) :: Velo1    !Velocidade do vértice no instante t1.
  TYPE(Velocity) :: Velo2    !Velocidade do vértice no instante t2.
  TYPE[LEVEL_COMPONENTS1],POINTER::MapLagran ! Ponteiro para o mapa lagrangeano.
  TYPE(Vertex),POINTER   :: Nextv  !Ponteiro para próximo vértice.
  TYPE(Vertex),POINTER   :: Prevv  !Ponteiro para o vértice anterior.
END TYPE Vertex

END MODULE Half_Edge
```

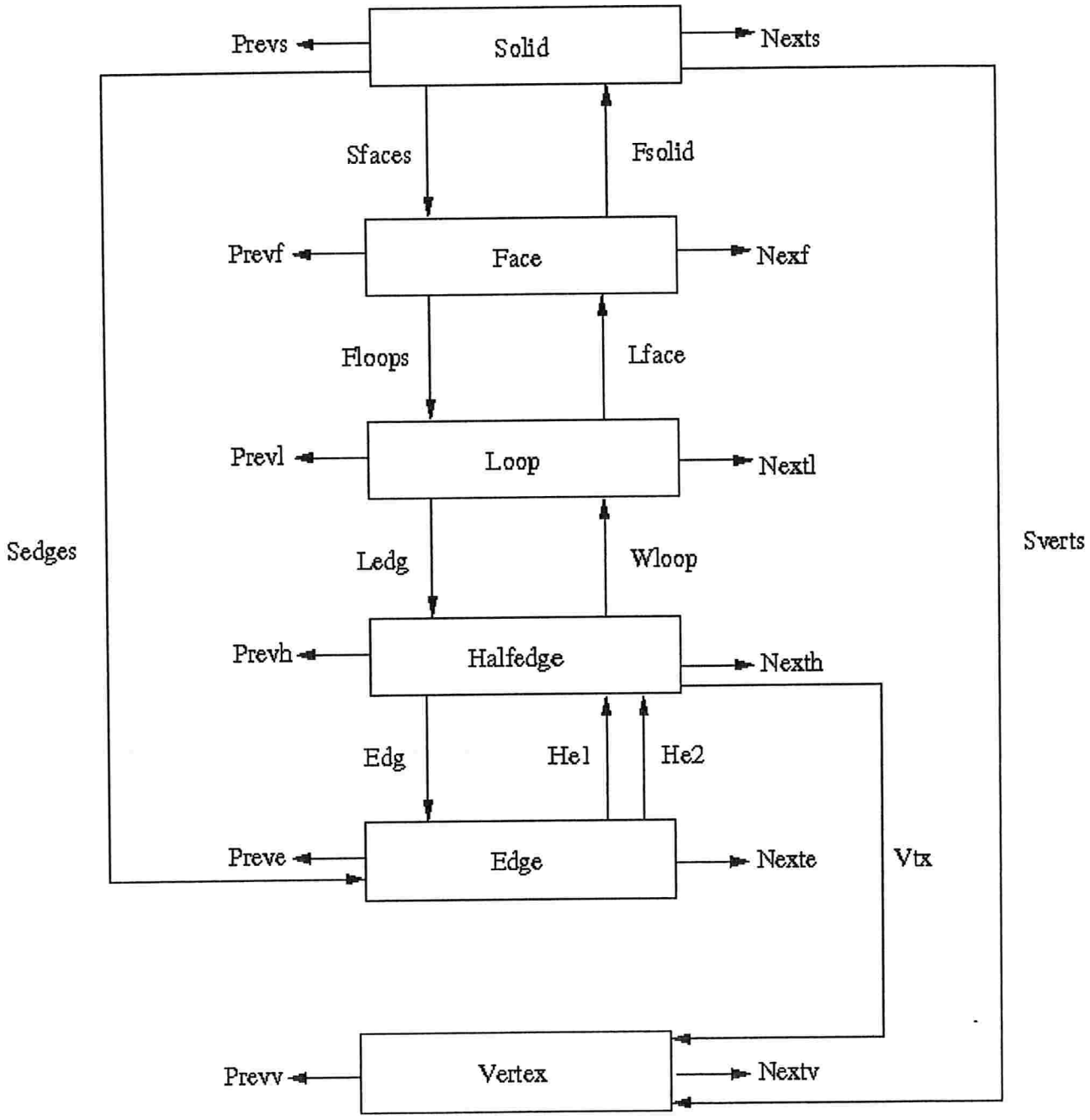


Figura C.1: Diagrama da Estrutura Halfedge.

APÊNDICE D

Definições Básicas sobre Grafos e Geometria

Neste Apêndice, são apresentadas as definições básicas sobre grafos e geometria, com o objetivo de auxiliar o leitor pouco familiarizado com a terminologia empregada no capítulo 3 sobre a estrutura de dados Halfedge e, portanto, sua leitura é opcional.

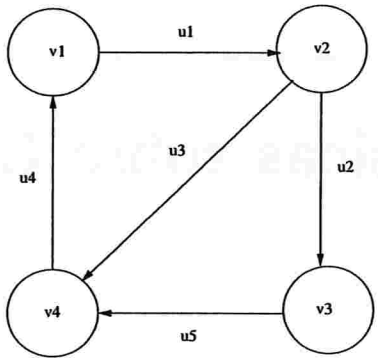
Termos utilizados na teoria dos grafos:

1. **Grafo Orientado**: Um *grafo orientado*, G=(V,U), também chamado de *digrafo*, é um par de conjuntos, onde:

   - **V** é um conjunto não vazio, finito ou não, cujos elementos são chamados de *vértices*.
   - **U** é um conjunto formado por pares ordenados u=(v,w), com v e w $\in V$, chamados de *arco*. A primeira coordenada v, é a extremidade inicial do arco e a segunda coordenada, w, é a extremidade final do arco.

   Um grafo orientado pode se visualizado através de uma *representação geométrica*, na qual seus vértices correspondem a pontos distintos do plano colocados em posições arbitrárias, enquanto o arco u=(v,w) é associada a uma curva orientada arbitrária, unindo o vértice inicial v ao vértice final w.

   O grafo superior, apresentado na Figura D.1, é orientado, sendo os conjuntos, V={v1,v2,v3,v4} e U={u1,u2,u3,u4,u5}, onde u1=(v1,v2), u2=(v2,v3), u3=(v2,v4), u4=(v4,v1), u5=(v3,v4).



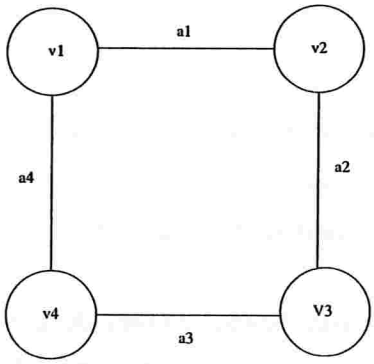


Figura D.1: Grafo Orientado (superior) e Grafo Não-Orientado (inferior)

2. **Grafo Não-Orientado**: Um *grafo não-orientado*, G=(V,U), é um par de conjuntos, onde:

   - **V** é um conjunto não vazio, finito ou não, cujos elementos são chamados de *vértices*.
   - **U** é um conjunto formado por pares não-ordenados u={v,w}, com v e w $\in V$, chamados de *aresta*. Não há orientação entre as arestas que unem os vértices.

   O grafo inferior, apresentado na Figura D.1, é não-orientado, sendo os conjuntos, V={v1,v2,v3,v4} e U={u1,u2,u3,u4}, onde u1={v1,v2}, u2={v2,v3}, u3={v3,v4}, u4={v4,v1}.

3. **Adjacência**: Dois vértices v e w de um grafo não-orientado, G(V,U), são adjacentes, se existir uma aresta u={v,w} em G. Esta aresta é dita ser *incidente* a ambos os vértices e eles são chamados de vértices *adjacentes*.

   No caso do grafo ser orientado, a definição de adjacência é especializada:

   - **Sucessor**: Um vértice w é dito ser *sucessor* de um vértice v, se existe um arco que parte de v e chega em w.
   - **Antecessor**: Um vértice v é dito ser *antecessor* de um vértice w, se existe um arco que parte de w e chega em v.

4. **Grafos Isomorfos**: Dois grafos $G_1(V, U1)$ e $G_2(W, U2)$ são isomorfos se houver uma correspondência biunívoca entre o conjunto de seus vértices que preserva suas adjacências. Isto é, se dois vértices de G_1 são adjacentes, então os seus correspondentes vértices de G_2, também são adjacentes e vice-versa.

   Na Figura D.2 apresenta-se dois grafos isomorfos, sendo a correspondência biunívoca de: $V = \{v1, v2, v3, v4, v5, v6\} \rightarrow W = \{w1, w2, w3, w4, w5, w6\}$, dada por,

   $\{(v3 \rightarrow w2), (v5 \rightarrow w5), (v6 \rightarrow w6), (v4 \rightarrow w1), (v2 \rightarrow w3), (v1 \rightarrow w4)\}$.

5. **Grafo Planar**: Um grafo, G(V,U), é dito ser *planar*, quando existir alguma forma de se dispor seus vértices em um plano de tal forma que nenhum par de arestas se cruze. É importante ressaltar que, podem existir dois grafos isomorfos, cuja representação geométrica de um apresente arestas que se interceptem e no outro não. Nestes casos,

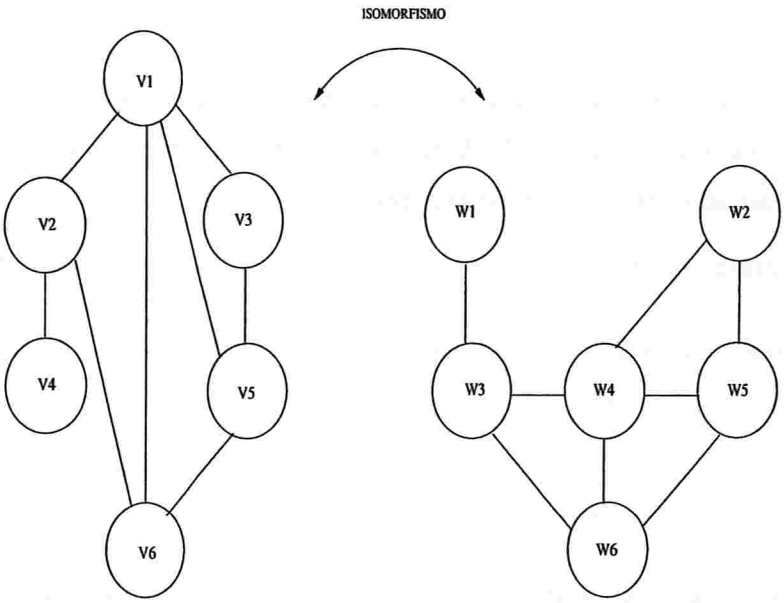


Figura D.2: Isomorfismo entre Grafos

a definição de grafo planar continua válida, ou seja, basta encontrar um único grafo isomorfo em que as aresta não se cruzem, para que a definição seja válida.

6. **Grau**: Para um grafo não-orientado, defini-se *grau* de um vértice com sendo o número de arestas que lhe são incidentes. No caso do grafo ser orientado, a definição de grau é especializada:
   - **Grau de Emissão**: O *grau de emissão* de um vértice, corresponde ao número de arcos que partem deste vértice.
   - **Grau de Recepção**: O *grau de recepção* de um vértice, corresponde ao número de arcos que chegam a este vértice.
7. **Grafo Regular**: Um grafo é dito ser *regular*, quando todos os seus vértices tem o mesmo grau.
8. **Grafo Completo**: Um grafo é dito ser *completo* quando existe uma aresta entre cada par de seus vértices, isto é, de qualquer vértice, partem arestas para todos os demais vértices do grafo. Tais grafos são designados por k_n, onde n é a ordem do grafo, logo, um grafo de ordem n tem todos os vértices com grau n-1, desta forma, todo grafo completo é um grafo regular.
9. **Cadeia**: Uma *cadeia* é uma sequência qualquer, finita ou não, de arestas adjacentes unindo dois vértices. Este conceito também é extendido para os grafos não-orientados, bastando que se ignore o sentido de percurso. Um cadeia é dita *elementar* se não passar duas vezes pelo mesmo vértice e é dita *simples* se não passar duas vezes pela mesma aresta ou arco. Define-se o *comprimento* de uma cadeia como sendo o número de aresta ou arcos que a compõe.
10. **Grafo Conexo**: Um grafo, G, é dito ser *conexo*, se existir um cadeia entre qualquer par de seus vértices.
11. **Caminho**: Um *caminho* é um cadeia, finita ou não, na qual todos os arcos possuem a mesma orientação. Esta definição só se aplica a grafos orientados.
12. **Ciclo**: Um *ciclo* é uma cadeia finita simples e fechada, isto é, os vértices inicial e final são coincidentes.

13. **Circuito**: Um *circuito* é uma cadeia finita simples e fechada. Este conceito só se aplica a grafos orientados.

14. **Grafo Euleriano**: Um grafo, G(U,V), é dito ser *euleriano*, se existir um ciclo em G, que contenha todas as arestas, sendo que cada aresta só deve aparecer, uma única vez, no ciclo. Isto é, saindo-se de um vértice, percorre-se todo o grafo e retorna-se a este mesmo vértice, passando, um única vez, por cada aresta. Este ciclo é chamado de *ciclo euleriano*. É importante observar que:

    - Um grafo só admite um ciclo euleriano se e somente se for conexo e só tiver vértices com grau par.(Teorema de Euler)
    - Um grafo completo com n vértices só terá ciclo euleriano se e somente se **n** for par.

15. **Grafo Hamiltoniano**: Um grafo, G(U,V), é dito ser *hamiltoniano*, se existir um ciclo em G, que contenha todos os vértices, sendo que cada vértice só deve aparecer, uma única vez, no ciclo. Este ciclo é chamado de *ciclo hamiltoniano*.

    Não existe condição necessária e/ou suficiente de verificação simples, como no caso do ciclo euleriano, que permita afirmar se existe ou não um ciclo hamiltoniano em um dado grafo.Vale ressaltar que há grafos onde somente existem ciclos hamiltonianos ou somente ciclos eulerianos ou ambos os ciclos ou nenhum deles.

Termos utilizados em Geometria:

1. **Coordenadas Homogêneas**: Uma das principais desvantagens de se utilizar coordenadas cartesianas em geometria computacional, é que elas não conseguem representar um ponto no infinito. Esta limitação, obriga os algoritmos geométricos a tratar separadamente muitos casos particulares. Segundo [16], ao se elaborar um algoritmo que precise calcular a interseção de duas retas no plano, utilizando-se a geometria cartesiana, tem de se considerar três casos distintos: as retas serem coincidentes, paralelas ou concorrentes. Isto não ocorreria, caso se utiliza-se coordenadas homogêneas.

    Se por definição, (X, Y, Z), são as coordenadas cartesianas de um ponto do $\mathbb{R}^3$, as *coordenadas homogêneas* deste ponto, são uma quádrupla de números reais [w, x, y, z], tais que, $X = \frac{x}{w}$, $Y = \frac{y}{w}$ e $Z = \frac{z}{w}$. A coordenada w é chamada de *peso*.

Esta definição permite que, um mesmo ponto do $\mathbb{R}^3$, possa ser representado por uma infinidade de quádruplas de coordenadas homogêneas. Por exemplo, as quádruplas [4 ,-6, 10, 2] , [-2, 3, -5, -1] e [0.2, -0.3, 0.5, 0.1] representam o mesmo o ponto, cujas coordenadas cartesianas são (2, -3, 5). As coordenadas homogêneas de um ponto não tem significado individual; apenas as razões $\frac{x}{w}$, $\frac{y}{w}$ e $\frac{z}{w}$, tem sentido geométrico.

É importante observar que, quando o peso w tende para zero, mantendo-se x ,y e z fixos, o ponto [w, x, y, z] tende a se afastar infinitamente da origem, na direção do vetor (x, y, z). Considera-se a quádrupla homogênea [0, x, y, z], que tem peso nulo, como sendo um ponto infinitamente distante da origem, um ponto *infinito*, na direção do vetor (x, y, z). Este ponto é denotado por ∞(x, y, z). O comprimento do vetor (x, y, z) é irrelevante, desde que não seja zero; isto é, as coordenadas homogêneas $[0, \alpha x, \alpha y, \alpha z]$ representam o mesmo ponto infinito, $\forall \alpha \in \mathbb{R}^+$. Por outro lado, o sentido do vetor é importante, ou seja, as quádruplas [0, x, y, z] e [0, -x, -y, -z] representam pontos infinitos distintos, segundo a geometria projetiva. Estes dois pontos infinitos são chamados de pontos *antipodais*, diz-se que um é o *antípoda* do outro.

Finalmente, ainda segundo [16], a quádrupla [0, 0, 0, 0] é um caso especial e de difícil interpretação geométrica, sendo melhor considerá-la com inválida.

2. **Polígono**: *Polígono* é uma figura plana formada por um número finito de segmentos de reta. Cada segmento de reta é chamado de *lado* e seus extremos, de *vértices*. Os lados devem ser tais que:

   - Cada vértice de cada lado é vértice de exatamente dois lados.
   - Os lados se intersectam apenas nos vértices.
   - Lados consecutivos do polígono não estão contidos na mesma reta.

3. **Interior do Polígono**: Um polígono divide o plano em duas regiões, uma limitada e outra ilimitada. A região limitada é chamada de *interior do polígono*.

4. **Polígono Convexo**: Um polígono é dito ser *convexo*, se dados dois pontos quaisquer A e B no interior do polígono, o segmento de reta AB fica inteiramente contido em seu interior.

5. **Polígono Equilátero**: Um polígono é dito ser *equilátero*, quando possui todos os lados congruentes.

6. **Polígono Equiângulo**: Um polígono é dito ser *equiângulo*, quando possui todos os ângulos internos congruentes.

7. **Polígono Regular**: Um polígono é dito ser *regular*, quando for ao mesmo tempo equilátero e equiângulo.

8. **Poliedro**: É a reunião de um número finito de polígonos planos, chamados de *face*. Cada lado do polígono é chamado de *aresta* e seus extremos, de *vértices*. As faces devem ser tais que:

   - Cada lado desses polígonos é também lado de um, e apenas um, outro polígono.
   - A interseção de dois polígonos quaisquer ou é um lado comum, ou é um vértice comum, ou é vazia.

9. **Interior do Poliedro**: Um poliedro divide o espaço em duas regiões, uma limitada e outra ilimitada. A região limitada é chamada de *interior do poliedro*.

10. **Poliedro Convexo**: Um poliedro é dito ser *convexo*, se dados dois pontos quaisquer A e B no interior do poliedro, o segmento de reta AB fica inteiramente contido em seu interior.

    Todo poliedro convexo obedece à seguinte equação: $V - A + F = 2$, conhecida como equação de Euler e onde V, A e F são os números de vértices, arestas e faces do poliedro, respectivamente.

11. **Poliedro Regular**: Um poliedro é dito ser *regular*, se for um poliedro convexo em que as faces são polígonos regulares congruentes e que em todos os vértices concorrem o mesmo número de arestas. Existem apenas cincos poliedros regulares, conhecidos como poliedros de Platão, são eles: tetraedro, hexaedro, octaedro, dodecaedro, icosaedro.

Referências Bibliográficas

[1] Jamil J. Shah and Martti Mantyla, Parametric Featture-Based CAD/CAM Concepts Thechniques and Applications, A Wiley-Interscience Publication, (1995)

[2] Mantyla Martti, An Introduction to Solid Modelling, Computer Science Press, (1988)

[3] Michael Henle, A combinatorial Introduction to Topology, Dover Publications, INC, (1977)

[4] Luiz Henrique de Figueiredo e Paulo César Pinto Carvalho, Introdução à Geometria Computacional - 18 Colóquio Brasileiro de Matemática, IMPA - Instituto de Matemática Pura e Aplicada, (1991)

[5] Peskin, C.S. Flow Patterns Around Heart Valves: A Digital Computer Method for Solving the Equation of Motion. PhD thesis, Albert Einstein College of Medicine-Yeshiva University. (July 1972)

[6] Peskin, C.S. Flow Patterns Around Heart Valves: A numerical method. Journal of Computational Physics, 10:252-271. (1972)

[7] Mayo, A.A. Peskin, C.S. An Implicit Numerical Method For Fluid Dynamics Problems With The Immersed Elastic Boundaries. Contemporary Mathematics 141:261-277. (1993)

[8] Peskin, C.S. McQueen, D.M. Computational Biofluid Dynamics. Contemporary Mathematics l41:161-186 (1993)

[9] Berger,M.J. Adaptative Mesh Refinement for Hiperbolic Partial Differential Equation. PhD thesis, Stanford University. (1982)

[10] Howell, L.H. A Multilevel Adaptive Projection Method for Unsteady Incompressible Flow. Technical Report UCRL-JC-112327. Lawrence Livermore National Laboratory, Livermore, C.A. 94550. (May 1993)

[11] Howell, L.H. A Adaptive-Mesh Projection Method for Viscous Incompressible Flow. Technical Report UCRL-JC-105181. Lawrence Livermore National Laboratory, Livermore, C.A. 94550. (June 1994)

[12] Harlow, F.H. Welch, J.E. Numerical Calculation of Time-Dependent of Viscous Incompressible Flow of Fluids with Free Surfaces. Physics of Fluids. 8(12):251-263 (1965)

[13] Roma, A.M. A Multilevel Self Adaptive Version of The Immersed Boundary Method. PhD thesis, New York University. (January 1996)

[14] Fox, Robert W. McDonald, Alan T. Introdução à Mecância dos Fluidos - 5 Edição. LTC - Livros Técnicos e Científicos Editora S.A.(1:25) (1998)

[15] Santos, Alberto Enrique Remigio, Simulação Numérica Bidimensional da Interação Fluido-Estrutura através do Método Físico Virtual. Tese de Doutoramento, Universidade de São Paulo. (Setembro 2005)

[16] Resende, Pedro J. e Stolfi, Jorge, Fundamentos de Geometria Computacional. Departamento de Ciência da Computação, Universidade Estadual de Campinas, IX Escola de Computação (25:28) (1994)

[17] Berger, M.J. Rigoutsos, I. An algorithm for point clustering and grid generation. IEEE Transactions on Systems, Man and Cybernetics,21(5):1278-1286 (September/October 1991)

[18] Lima e Silva, A.L.F; e Silveira-Neto, A; Damasceno, J.J.R. Numerical simulation of two-dimensional flows over a circular cylinder using the immersed boundary method CoÄmput, Phys. 189:351-370 (2003)

[19] Peskin C.S. The immersed boundary method. Acta Numerica(2002) pp.479-517 Cambridge University Press (2002)

[20] CockBurn,B. Yenikaya,B. An Adaptive Method with Rigorous Error Control for the Hamilton-Jacobi Equations. Journal of Computational, Volume 209, Issue 2, (391-405) (November 2005)

[21] Bakosi,J. Modeling of Turbulent flows using a probabilistic approach. PhD thesis, George Mason University . (September 2005)

[22] Kemmler,D. Kolditz, O. Panagiotis,A. Rabenseifner,R. Application of High Performance Computing Techniques (Parallel Processing) to The Modeling of Complex Coupled Geo-Processes Using a Finite Element Approach (Geosys/RockFlow). University of Tübingen, Center for Applied Geosciences, GeoSystemsResearch (January 2005)

[23] Xunlei,W. Tendick,F. Multigrid Integration for Interactive Deformable Body Simulation. Simuation Group, CIMIT / Harvad University . (2004)

[24] Berger,M.J. Jameson,A. Automatic Adaptative Grid Refinement for the Euler Equations. Journal AIAA, Vol 23, Issue 2 (April 1985)

[25] Ruo,L. Tang, T. Zhang,P. Moving Mesh Methods in Multiple Dimensions Based on Harmonic Maps. Journal of Computational Physics 170, 562-588. (2001)

[26] Quian,Y.H. d'Humiéres,D. Lallemand,P. Europhysics Letters 17, 479. (1992)

[27] Sauro,S. The Lattice Boltzmann Equation for Fluid Dynamics and Beyond. Oxford University Press, Oxford, 288p. (2001)

[28] Wolf-Gladrow,D. Lattice Gas Celular Automata and Lattice Boltzmann Models: An Introduction. Springer-Verlag, Berlin, 308p. (2000)

[29] Souza,F.S. Mangiavacchi,N. Nonato,L.G. Castelo,F.A. Tomé,M.F. Ferreira V.G. Cuminato,J.A. McKee,S. A Front-tracking / Front-capturing Method for the simulation of 3D Multifluid Flows with Free Surfaces. Journal of Computational Physics Volume 198, Issue 2, 469-499. (2004)

[30] Melton,J.E. Berger,B.J. Aftosmis,M.J. Robust an Efficient Cartesian Mesh Generation Component-Based Geometry. Journal AIAA, 97-0196. (January 1997)

[31] Bell,J. Berger,B.J. Wellcome,M. Three-Dimensional Adaptive Mesh Refinement for Hyperbolic Conservation Laws. Journal SIAM, Vol 15,N.1, pp 127-138. (January 1994)

[32] Thang,N.T. Davis,M.R. The Structure of Bubbly Flows Through Venturis. International Journal of Multiphases Flows,5, 17-37. (January 1979)

[33] Drew,D.A. Lahey,R.T. Some Supplemental Analysis Concerning The Virtual Mass and Lift Force on a sphere in Rotation and Straining Inviscid Flow. International Journal of Multiphases Flows,16(6), 1127-1130. (1990)

[34] Ishii,R. Umeda,Y. Murata,S. Bubby Flows Though a Converging-Diverging Nozzle. Phys. Fluid A. 5(7), 1630-1643. (1993)

[35] Karena,H. Lo. S. Eficiency of Interphase Coupling Algorithms in Fluidized Bed Condition. Computers and Fluids 28, 323-360. (1999)

[36] Mittal,R. Laccarino,G. Immersed Boundary Methods. Annual Reviews of Fluid Mechanics, Vol. 37:239-261. (January 2005)

[37] Unverdi,S. Tryggvason,G. A front-tracking method for viscous, incompressible, multi-fluid flows Journal of Computational Physics, Vol. 100, Issue 1:25-37 (May 1992)

[38] Khadra,K. Angot,P. Parneix,S. Caltagirone,J.P. Fictitious domain approach for numerical modelling of Navier-Stokes equations. International Journal for Numerical Methods in Fluids, Vol. 34, Issue 8:651-684. (December 2000)

[39] Angot,P. Bruneau,C.H. Fabrie,P. A penalization method to take into account obstacles in incompressible viscous flows. Journal Numerische Mathematik,Vol. 81, Issue 4:497-520, Springer Berlin / Heidelberg (February 1999)

[40] Mohd-Yusof,J. Combined Immersed-Boundary/B-spline Methods for Simulations of Flow in Complex Geometries. Annual Research Briefs (1997)

[41] Verzicco,R. Camussi,R. Prandtl number effects in convective turbulence Journal of Fluid Mechanics 383:55-73,Cambridge University Press (1999)

[42] Majumdar,S. Iaccarino,G. Durbin,P. RANS Solvers With Adaptive Structured Boundary Non-Conforming Grids. Annual Research Briefs, Center for Turbulence Research (2001)

[43] Tseng,Y.H. Ferziger,J.H. A ghost-cell immersed boundary method for flow in complex geometry. Journal of Computational Physics, Vol.192, Issue 2:593-623 (December)

Índice Remissivo