Carina Alexandra Rondini Marretto

O Algoritmo de Delaunay em Três Dimensões: Uma Implementação Computacional para o Ambiente Mathematica for Windows

O Algoritmo de Delaunay em Três Dimensões: Uma Implementação Computacional para o Ambiente Mathematica For Windows

Carina Alexandra Rondini Marretto

Dissertação de Mestrado
Pós-Graduação em Matemática Aplicada
MAP-046

O Algoritmo de Delaunay em Três Dimensões: Uma Implementação Computacional para o Ambiente Mathematica For Windows

CARINA ALEXANDRA RONDINI MARRETTO

Dissertação apresentada ao Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de São José do Rio Preto, para a obtenção do título de Mestre em Matemática Aplicada.

Orientador: Prof. Dr. José Márcio Machado

São José do Rio Preto

1999

Resumo

Este trabalho descreve a implementação e testes de validação feitos para um pacote de programas compactos que executa o algoritmo de Delaunay em domínios tridimensionais. Os programas foram desenvolvidos inicialmente para uso no ambiente Mathematica For Windows. O trabalho concentra-se na discussão de aspectos críticos da implementação do algoritmo, como a supressão de degenerescências, e mostra vários exemplos de aplicação dos programas à situações de interesse. Os casos testados mostraram que é viável utilizar o ambiente Mathematica para implementação do algoritmo de Delaunay em domínios tridimensionais, uma vez que os tempos de processamento envolvidos foram aceitáveis. Versões mais sofisticadas dos programas, para discretização de malhas com um grande número de elementos, poderiam ser facilmente obtidas a partir da conversão do software já existente.

Abstract

This work addresses the construction of a compact program package which executes the Delaunay algorithm in three dimensions, giving the implementation steps and tests of validation. The programs were initialy developed in the Mathematica for Windows enviroment. In this regard, crictical aspects of the implementation are discussed, like degenerate cases, and several applications to situations of practical interest are also presented. The results obtained have shown that the implementation mentioned above is promissing, due to the acceptable CPU times obtained for all tested cases. More sophisticated versions of the package (by the way, for the treatment of well refined meshes) could be easily obtaned in the future, by addaptions in the programas here described.

O ponto, que ontem era invisível, é hoje o ponto chegada.

Amanhã , será o de partida ...

(MACAULAY)

Para

Meus pais, João Carlos e Henory,

Meu marido Willian,

infinitamente responsáveis pela minha travessia

do querer fazer ao fazer ser,

dedico este trabalho.

Agradecimentos

Aos meus pais hoje e sempre por tudo que fizeram e ainda fazem por mim, registro aqui, não só o meu agradecimento mais o meu infinito amor por eles.

Ao meu marido, companheiro e incentivador agradeço o seu terno amor e carinho.

Ao meu orientador Prof.Dr.José Márcio Machado que tanto se empenhou para a minha formação profissional quero expressar a minha incondicional admiração.

Ao professor Sebastião e a Marilene pela gentileza de terem lido este trabalho.

A CAPES pela bolsa concedida, a qual possibilitou a dedicação a este trabalho.

A Deus e a Santa Rita, a cujos pés tenho depositado todos os meus projetos, e a todos aqueles que concorreram para o remover das pedras que havia no meio do caminho, entre eles em especial às minhas amigas e companheiras nos congressos Elisa, Lisandra e Telma. Aos meus amigos Clinton e Marcelo Palin, aos quais expresso minha infinita admiração e gratidão pelas inumeras vezes que com paciência me ajudaram a sanar um pouco a deficiência que tenho em computação. Aos amigos que adquiri no decorrer desses anos agradeço por tudo, pelos momentos felizes, pelo companherismo, amizade e preocupação que sempre tivemos uns pelos outros.

E finalmente, entretanto em enorme nível de importância, aos meus irmãos Carlos Eduardo e Cássia Andréia que sempre me incentivaram, a minha querida sobrinha e afilhada Anelise, minha sobrinha Louise, minha sogra Dalila que cuidou de meu marido todas às vezes que eu passava o dia todo na faculdade estudando, e a todos que aqui não agradeci por escrito mas que agradeço do fundo do meu coração.

Índice

Introdução

O grande desenvolvimento experimentado pelos computadores nas últimas décadas tem possibilitado o acesso a máquinas de grande capacidade de processamento e memória a um custo cada vez menor. Para os engenheiros e pesquisadores nos mais diversos campos do conhecimento, essa disponibilidade de máquinas poderosas e baratas estimulou enormemente a simulação computacional de fenômenos naturais e equipamentos.

A base para a construção de todos os programas de simulação é, como se sabe, um conhecimento adequado de técnicas automáticas para a solução numérica de equações diferenciais parciais.

Muitas dessas técnicas, embora conhecidas já de longa data, só puderam ter o seu potencial plenamente explorado com o já mencionado avanço na tecnologia da Informática.

Dentre elas, o Método dos Elementos Finitos e suas variantes é uma das que, provavelmente mais atenção tem recebido nos últimos tempos, com uma quantidade impressionante de trabalhos publicados na literatura especializada, cobrindo avanços e resultados de impacto em áreas tão diversas quanto a Engenharia Aeronáutica, as Engenharias de Energia e Automação Elétricas, Eletrônica e Mecânica, a Física do Estado Sólido, etc., para ficar apenas em alguns exemplos.

Contudo uma exigência fundamental para qualquer programa de simulação que empregue tal técnica é a capacidade de gerar modelos geométricos que representem o domínio em estudo, discretizando-o automaticamente a seguir em uma malha de elementos finitos. Esta malha é na realidade,

um conjunto de elementos geométricos topologicamente regulares e disjuntos (não necessariamente iguais) cuja união reproduz o domínio original de modo consistente. Por modo consistente entende-se aqui a ausência de falhas ou superposições entre os elementos.

Embora a construção de malhas de elementos finitos bidimensionais seja amplamente dominada no estágio atual de conhecimento, os algoritmos de geração automática de malhas em três dimensões ainda são todavia pouco robustos, apresentando níveis diferentes de automatização e requerendo, muitas vezes, a intervenção manual do usuário em pontos críticos do domínio.

Mais ainda, não existe um algoritmo que possa ser comodamente usado em todas as situações e domínios geométricos possíveis, apresentando cada um deles, para o caso tridimensional, qualidades e deficiências que se explicitam em situações muito distintas.

Desta forma, a geração automática de malhas de elementos finitos tridimensionais constitui presentemente um tópico de pesquisa dos mais relevantes em simulação computacional. Ao se executar a simulação computacional de um dado problema, as etapas de modelagem geométrica e posterior geração de malha ainda são as que demandam, em um programa típico, a maior parte do esforço interativo e da atenção dedicada pelo usuário.

Dentre as várias abordagens disponíveis para a geração de malhas tridimensionais, uma das mais populares e explorada é o algoritmo de Delaunay. Nesta dissertação, serão discutidos aspectos da versão do algoritmo de Delaunay conhecida como implementação de Watson.

Devido ao fato deste algoritmo ser na realidade um processo de refinamento recursivo de uma malha pré existente, que procura conectar um conjunto discreto de pontos para formar um conjunto de elementos tetraédricos, ele apresenta dificuldades na reprodução dos contornos de um dado domínio geométrico. Especialmente no caso de geometrias com concavidades ou domínios multiplamente conexos, alguns elementos podem cruzar os contornos do domínio, ou partes do mesmo domínio podem situar-se fora da cobertura pelos elementos da malha.

Embora seja difícil construir um gerador automático de malhas com elevada generalidade, baseando-se apenas no algoritmo de Delaunay, basicamente pelas razões já expostas, algumas situações de grande interesse prático podem ser solucionadas com expedientes relativamente simples.

Procurando explorar este fato, as implementações computacionais apresentadas neste trabalho foram todas executadas no ambiente Mathematica for Windows [1]: muito embora a eficiência em tempo de processamento seja bem menor do que se fosse efetuada uma programação em linguagem C, o volume de código produzido é menor, e tanto a implementação propriamente dita quanto a análise gráfica dos resultados, se beneficiam das funções pré existentes no ambiente.

A idéia aqui é tirar partido dos recursos do ambiente para obter um conjunto de programas protótipos, que permitiriam a usuários mais treinados em programação migrar, sem maiores dificuldades, para outras linguagens e sistemas operacionais, buscando uma maior eficiência de processamento.

É preciso mencionar também que o próprio Mathematica for Windows é um produto de software em permanente evolução, podendo-se esperar o aparecimento de versões mais poderosas, onde recursos de compilação de funções permitam gerar códigos de máquina muito mais rápidos do que as versões por ora disponíveis.

Capítulo 1

O algoritmo de Delaunay e suas implementações

A representação de sólidos em computação gráfica pode ser feita por meio de um conjunto com um número finito de pontos adequadamente escolhidos, de forma a descrever de modo aproximado, mas aceitável, a fronteira do sólido (sua "casca" externa) e sua região interior.

O algoritmo de Delaunay em sua versão tridimensional é capaz de, utilizando esses pontos, construir automaticamente um conjunto de tetraedros (a malha) cuja união reproduz o sólido original. A analogia com as peças de um quebra-cabeças corretamente montado é perfeita para ilustrar a situação: nenhuma das peças deve se sobrepor, e também não devem existir lacunas [2].

Além disso, trata-se de um processo recursivo: uma malha inicial com um número bem reduzido de elementos é modificada por meio da inserção sucessiva dos pontos da lista que descrevem o sólido, até que não haja mais nenhum ponto a ser inserido. A malha final assim obtida, também chamada de discretização, deve reproduzir a geometria do sólido original da melhor forma possível.

Este capítulo, trata da parte matemática do algoritmo de De-

launay, fornece alguns dos métodos citados na literatura para a sua implementação, e discute os problemas associados à aplicação desse algoritmo em domínios tridimensionais.

A discusão que se seguirá pode ser integralmente encontrada em [3] de onde foram tiradas as figuras que ilustram esse capítulo. Aqueles que estiverem mais interessados na implementação computacional propriamente dita, podem passar diretamente à seção 1.5 sem grande prejuízo para a compreensão dos exemplos discutidos no Capítulo 2.

1.1 Preliminares

Definição 1.1.1 Um simplexo K é uma lista ordenada de vértices P_i com $1 \leq i \leq d+1$ onde d é a dimensão do espaço euclidiano afim E. Chama-se de $\det(K)$, ao determinante de ordem $d+1$ dado por:

$$\det(K) = \begin{vmatrix} 1 & \cdots & \cdots & \cdots & 1 \\ P_1^1 & \cdots & \cdots & \cdots & P_{d+1}^1 \\ P_1^2 & \cdots & \cdots & \cdots & P_{d+1}^2 \\ \vdots & & & & \vdots \\ P_1^d & \cdots & \cdots & \cdots & P_{d+1}^d \end{vmatrix} \tag{1.1}$$

onde P_i^j é a coordenada j do ponto P_i para $1 \leq i \leq d+1$ e $1 \leq j \leq d$.

Por definição tem-se que, se $\det(K) > 0$, K é dito ser positivamente orientado e se $\det(K) = 0$, K é dito ser degenerado (ou seja, todos os seus vértices pertencem ao mesmo hiperplano) [4].

Definição 1.1.2 Entende-se por "triangulação" de um domínio Ω, que se supõe limitado e poliédrico, um conjunto ζ de simplexos (triângulos em domínios bidimensionais, tetraedros em domínios tridimensionais). Para esse

conjunto temos as seguintes condições:

1. A intersecção de dois elementos de ζ é vazia ou se reduz a um vértice, a uma aresta (no caso bi-dimensional) ou a uma face (no caso tridimensional).

2. A união dos elementos de ζ é igual a Ω.

3. Os elementos de ζ devem ser topologicamente regulares: no caso bi-dimensional por exemplo, esses elementos devem ser o mais próximo possível da equilateralidade, o que não os obriga a serem iguais [5, 6].

4. Eventualmente a incidência de elementos deve ser maior nas regiões do domínio onde isto é necessário [7].

1.2 Poliedros de Voronoi

Definição 1.2.1 Seja E um espaço euclidiano afim de dimensão d e sejam $P_1, \ldots, P_n$, n pontos de E. Chama-se poliedro de Voronoi associado ao ponto P_i o conjunto:

$$V_i = \{P \in E, \forall\ 1 \leq j \leq n, d(P, P_i) \leq d(P, P_j)\} \tag{1.2}$$

onde $d(P, P_j)$ é a distância euclidiana definida no espaço em estudo.

São demonstráveis os seguintes resultados [8, 9]:

- Os poliedros de Voronoi são convexos, de interior não vazio e formam uma cobertura de E tal que dois elementos distintos tem intersecção vazia ou que se reduz a um vértice, a uma aresta ou a uma face (veja figura 1.2.1).

- Para cada vértice v de um poliedro de Voronoi associamos a ele um conjunto que chamamos de envolvente convexa C_v.

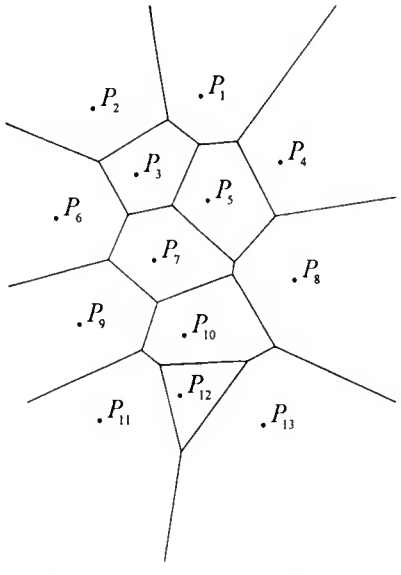

Figura 1.2.1: Poliedros de Voronoi

Definição 1.2.2 Envolvente convexa C_v é um conjunto que consiste de pontos dos poliedros de Voronoi que apresentam v como um dos seus vértices. O conjunto C_v é um politopo inscrito em uma esfera de centro v que não contém nenhum outro ponto do conjunto $P_1, \ldots, P_n$. Define-se então ζ como o conjunto dos politopos C_v.

É demonstrável que a intersecção de dois elementos de ζ é vazia ou reduzida a um vértice, a uma aresta ou a uma face e que a união dos seus elementos é igual à envolvente convexa dos pontos $P_1, \ldots, P_n$.

Assim o conjunto ζ aparece como uma geração de malhas por politopos da envolvente convexa dos pontos $P_1, \ldots, P_n$. Essa malha está representada pelas linhas pontilhadas na figura 1.2.2.

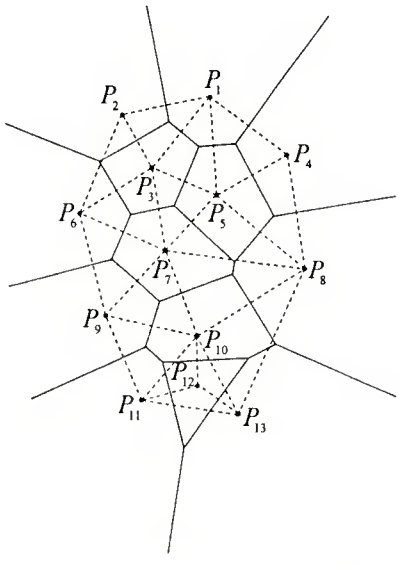

Figura 1.2.2: Malha da envolvente convexa

Definição 1.2.3 Segundo Delaunay [10] dir-se-á que os pontos $P_1, \ldots, P_n$ são especiais se existir um elemento de ζ que não é um simplexo (veja a figura 1.2.3).

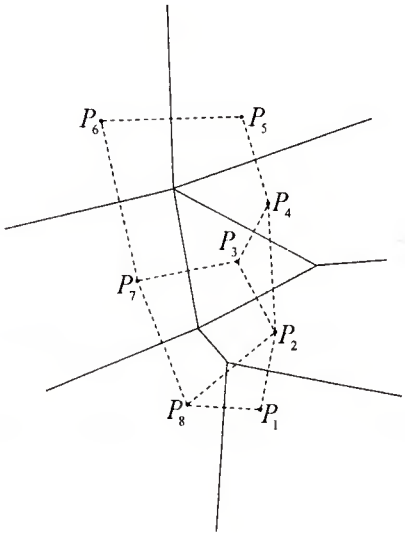

Figura 1.2.3: Divisão em simplexos de pontos especiais

Portanto, se os pontos $P_1, \ldots, P_n$ não forem especiais, concluí-se que ζ é uma triangulação da envolvente convexa dos pontos $P_1, \ldots, P_n$, caso contrário, obtém-se uma triangulação chamada "deduzida" de ζ, dividindo em

simplexos cada elemento de ζ que não seja um simplexo (veja a figura 1.2.4).

Resumidamente, chamar-se-á triangulação de Delaunay associada aos pontos $P_1, \ldots, P_n$ a triangulação ζ (se os pontos $P_1, \ldots, P_n$ não forem especiais), ou toda triangulação deduzida de ζ (se os pontos $P_1, \ldots, P_n$ forem especiais).

No caso da figura 1.2.3, existem 5 (respectivamente 2) maneiras de dividir o polígono $P_3P_4P_5P_6P_7$ (respectivamente $P_2P_3P_7P_8$) em triângulos: existem então 10 triangulações deduzidas de ζ (veja a figura 1.2.4).

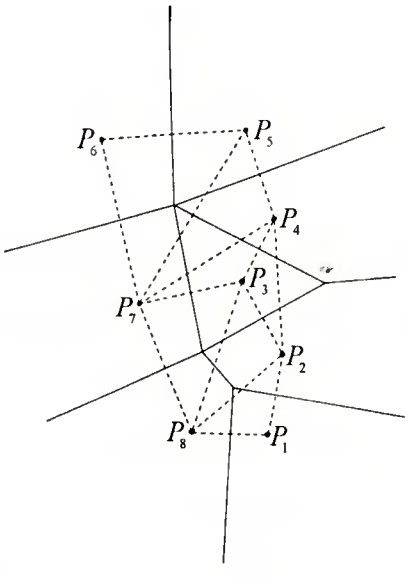

Figura 1.2.4: Triangulação deduzida de ζ

1.3 Algoritmo de Construção de uma triangulação onde todos os vértices são dados

Nesta seção será iniciada a descrição de um algoritmo que permita construir uma triangulação de Delaunay associada aos pontos $P_1, \ldots, P_n$.

A idéia de base do algoritmo consiste em gerar a triangulação de Delaunay ζ ponto por ponto, levando em conta que o $(i+1)$-ésimo ponto modifica localmente a triangulação de Delaunay ζ_i associada aos i primeiros

pontos de modo a obter uma triangulação de Delaunay ζ_{i+1} associada aos $i+1$ primeiros pontos [11].

1.3.1 Inicialização

A primeira triangulação de Delaunay construída é, então, ζ_{d+1} que é composta de um só simplexo $K = P_1, \ldots, P_{d+1}$.

Esse simplexo não é degenerado e positivamente orientado desde que: $\det(K) > 0$.

Uma vez que os pontos $P_1, \ldots, P_n$ não pertençam a um mesmo hiperplano esta condição é sempre verificável: basta fazer uma permutação conveniente desses pontos.

1.3.2 Determinação da situação geométrica do $(i+1)$-ésimo ponto em relação a triangulação.

Definição 1.3.1 Seja β_i o conjunto das esferas circunscritas aos elementos de ζ_i. Distinguem-se três tipos de situação geométrica do ponto P_{i+1} em relação aos conjuntos $\bigcup_{K\in\zeta_i} K$ e $\bigcup_{B\in\beta_i} B$ (onde $\bigcup_{K\in\zeta_i} K$ é igual à envolvente convexa dos pontos $P_1, \ldots, P_i$): i. $P_{i+1} \in \bigcup_{K\in\zeta_i} K$

ii. $P_{i+1} \notin \bigcup_{B\in\beta_i} B$

iii. $P_{i+1} \notin \bigcup_{K\in\zeta_i} K$ e $P_{i+1} \in \bigcup_{B\in\beta_i} B$

Na figura a 1.3.5 temos um exemplo onde os pontos P, Q e T estão respectivamente em situação i, ii, e iii em relação aos conjuntos $\bigcup_{K\in\zeta_i} K$ e $\bigcup_{B\in\beta_i} B$.

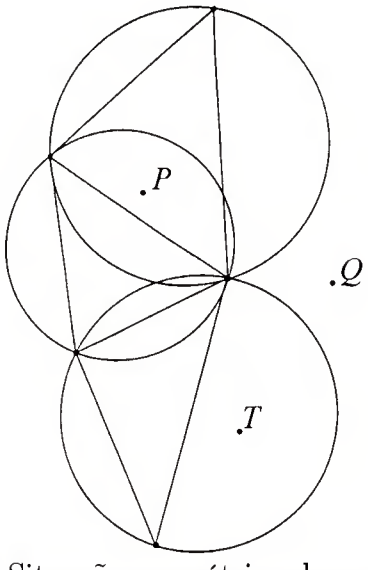

Figura 1.3.5: Situação geométrica dos pontos P, Q e T

Definição 1.3.2 Define-se por $F_1, \ldots, F_w$ as faces externas do poliedro convexo $\underset{K\in\zeta_i}{\cup} K$ e para cada uma dessas faces F_j seja $K = P_1 \ldots P_k \ldots P_{d+1}$ o único elemento de ζ_i do qual F_j é uma face (por exemplo, supondo-se que esse único elemento seja o k-ésimo para o qual tem-se que $F_j = P_1 \ldots P_{k-1} P_{k+1} \ldots P_{d+1}$). Seja $D(P, F_j) = \det(K')$ onde K' é o simplexo $P_1 \ldots P_{k-1} P P_{k+1} \ldots P_{d+1}$. Desde que todos os elementos de ζ_i são positivamente orientados, P pertence à $\underset{K\in\zeta_i}{\cup} K$ se e somente se:

$$D(P, F_j) > 0, \text{ para } \forall \ \ 1 \leq j \leq w.$$

Para cada simplexo $K = P_1 \ldots P_{d+1}$ denota-se por $\Delta(P, K)$ o determinante de ordem $d+2$ como segue:

$$\Delta(P, K) = \begin{vmatrix} l^2 & l_1^2 & \cdots & l_{d+1}^2 \\ 1 & 1 & \cdots & 1 \\ P^1 & P_1^1 & \cdots & P_{d+1}^1 \\ \cdots & \cdots & \cdots & \cdots \\ P^d & P_1^d & \cdots & P_{d+1}^d \end{vmatrix} \tag{1.3}$$

onde $l^2 = \sum_{k=1}^{d} \left(P^k\right)^2$ e $l_j^2 = \sum_{k=1}^{d} (P_j^k)^2$ para $1 \leq j \leq d+1$. Tem-se por definição que $\Delta(P, K) = 0$ é a equação da esfera circunscrita a K, logo, P pertence ao

interior dessa esfera se e somente se:

$$\Delta(P,K)\cdot\begin{vmatrix} 1 & \cdots & \cdots & 1 \\ P_1^1 & \cdots & \cdots & P_{d+1}^1 \\ \cdots & \cdots & \cdots & \cdots \\ P_1^d & \cdots & \cdots & P_{d+1}^d \end{vmatrix} < 0. \tag{1.4}$$

Deste modo, se tivermos $\Delta(P,K)\cdot\det(K) < 0$ conclui-se que $\Delta(P,K) < 0$ uma vez que K é positivamente orientado. Então P pertence à $\underset{B\in\beta_i}{\cup}B$ se e somente se:

$$\exists\ K\in\zeta_i \text{ tal que } \Delta(P,K) < 0.$$

Resumidamente as condições i, ii e iii são equivalentes às relações abaixo:

i. $D(P,F_j) > 0 \ \ \forall\ j\ 1\le j\le w$.

ii. $\Delta(P,K)\ge 0 \ \ \forall\ K\in\zeta_i$.

iii. $\exists\ F_j\ 1\le j\le w$ e $\exists\ K\in\zeta_i$ tal que $\Delta(P,K) < 0$.

1.3.3 Inclusão do $(i+1)$-ésimo ponto

As situações vistas em 1.3.2 induzem a três tipos de modificações da triangulação de Delaunay ζ_i para obter uma triangulação de Delaunay ζ_{i+1} que compreende o ponto P_{i+1}.

De fato, para a situação i, onde $P_{i+1}\in\underset{K\in\zeta_i}{\cup}K$ então $P_{i+1}\in\underset{B\in\beta_i}{\cup}B$. Suponha que λ seja o conjunto dos elementos de ζ_i cuja esfera circunscrita contém P e sejam $F_1,\ldots,F_w$ as faces de elementos de λ que não são comuns a dois elementos de λ.

O algoritmo gera um conjunto $\zeta_{i+1} = (\zeta_i/\lambda)\cup(P_{i+1}F_j)\ 1\le j\le w$ que é uma triangulação de Delaunay associada aos pontos $P_1,\ldots,P_{i+1}$ [9].

Na figura 1.3.6 pode-se observar este processo de construção tomando $P_{i+1} = P_9$.

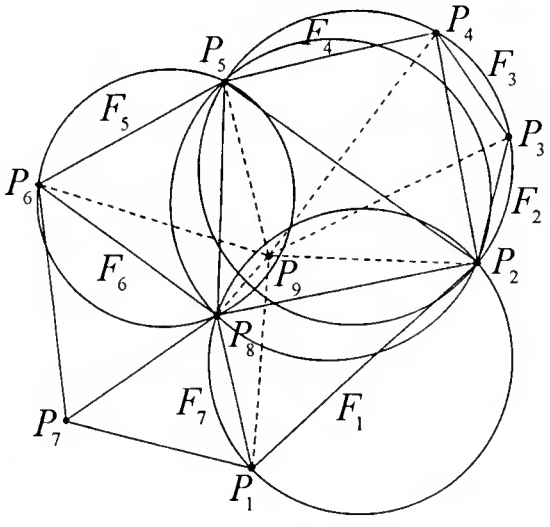

Figura 1.3.6: Inclusão do ponto P_9 (situação i)

Nota-se que a união dos elementos de λ é igual ao polígono $P_1P_2P_3P_4P_5P_6P_8$ ao passo que os 7 novos elementos $P_{i+1}F_j$ aparecem pontilhados.

Para a situação ii, onde $P_{i+1} \notin \underset{B\in\beta_i}{\cup} B$ então $P_{i+1} \notin \underset{K\in\zeta_i}{\cup} K$. Toma-se $F_1, \ldots, F_w$ as faces externas do politopo $\underset{K\in\zeta_i}{\cup} K$ que definem um hiperplano separando estritamente P_{i+1} e $\underset{K\in\zeta_i}{\cup} K$.

O algoritmo gera um conjunto $\zeta_{i+1} = \zeta_i \cup (P_{i+1}F_j)$ $1 \leq j \leq w$ que é uma triangulação de Delaunay associada aos pontos $P_1, \ldots, P_{i+1}$ [9].

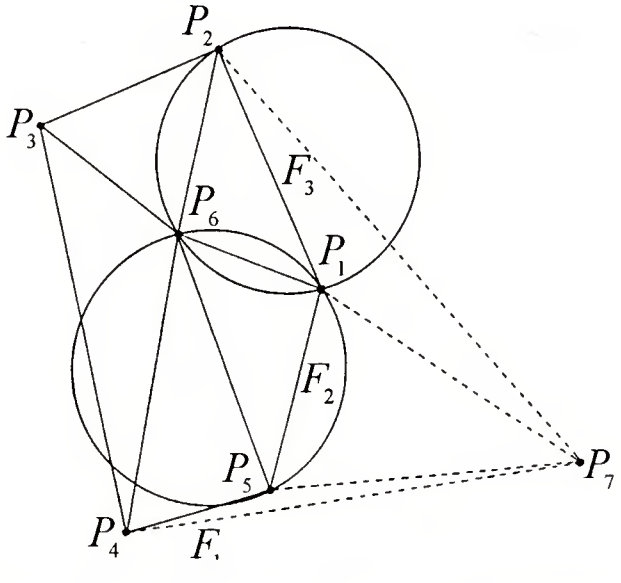

Figura 1.3.7: Inclusão do ponto P_7 (situação ii)

Na figura 1.3.7 pode-se observar esse processo de construção tomando $P_{i+1} = P_7$, onde os 3 novos simplexos $P_{i+1}F_j$ aparecem pontilhados.

A situação iii, coincide com a exclusão das situações i e ii. Neste caso tem-se que $P_{i+1} \notin \underset{K\in\zeta_i}{\cup} K$ mas P_{i+1} pode pertencer à fronteira de $\underset{K\in\zeta_i}{\cup} K$.

Denota-se por λ o conjunto dos elementos de ζ_i cuja esfera circunscrita contém P_{i+1} e sejam:

- $F_1, \ldots, F_w$ as faces de elementos de λ que não são comuns a dois elementos de λ e que determinam um hiperplano não separando P_{i+1} e $\underset{K\in\zeta_i}{\cup} K$.
- $F_{w+1}, \ldots, F_{w+q}$ as faces externas de $\underset{K\in\zeta_i}{\cup} K$ que não são as faces dos elementos de λ e que determinam um hiperplano separando estritamente P_{i+1} e $\underset{K\in\zeta_i}{\cup} K$.

O algoritmo gera um conjunto $\zeta_{i+1} = (\zeta_i/\lambda) \cup (P_{i+1}F_j)$ $1 \leq j \leq w+q$ que é uma triangulação de Delaunay associada aos pontos $P_1, \ldots, P_{i+1}$ [9].

A figura 1.3.8 permite observar este processo de construção partindo de $P_{i+1} = P_{12}$, $w = 5$ e $q = 2$.

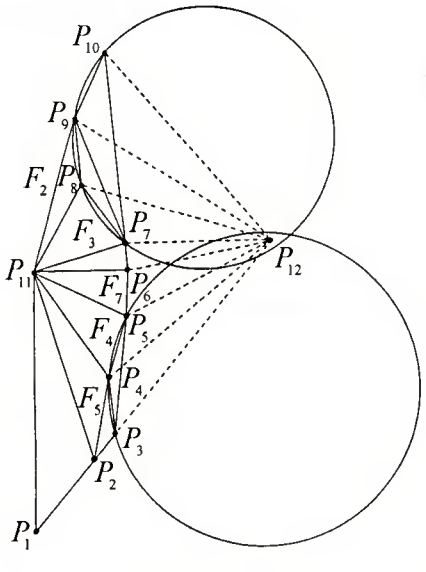

Figura 1.3.8: Inclusão do ponto P_{12} (situação *iii*)

Nota-se que a união dos elementos de λ é igual à reunião dos polígonos $P_7P_{10}P_9P_8$ e $P_3P_5P_4$ enquanto que os 7 novos elementos aparecem pontilhados.

Em todos os casos vistos anteriormente deve-se ter cuidado para que os "novos" simplexos $P_{i+1}F_j$ sejam positivamente orientados.

1.4 Triangulação de poliedros quaisquer

Seja Ω um poliedro qualquer. Descreve-se nesta seção um algoritmo que gera uma triangulação "conveniente" de Ω, isto é, verifica-se as condições 3 e 4 da seção 1.1.

1.4.1 Descrição da fronteira de Ω

Em geral a descrição de um poliedro coincide com a de sua fronteira. Precisa-se então saber quais são os dados mínimos necessários para determinar esta fronteira.

1.4.2 Poliedro convexo

A fronteira neste caso é descrita por uma lista de pontos onde Ω é a envolvente convexa. Segundo o teorema de Krein e Milman [12], é necessário fornecer todos os pontos extremos de Ω. Ainda que essa condição seja suficiente, não é exigido na seqüência que apenas os pontos extremos de Ω sejam fornecidos: outros pontos da fronteira de Ω podem ser estabelecidos.

1.4.3 Poliedro "quase convexo"

Definição 1.4.1 Dir-se-á que um poliedro Ω é "quase convexo" se existir uma seqüência $\Omega_1, \ldots, \Omega_w$ de poliedros convexos tais que para todo i, j com

$1 \leq i \neq j \leq w$, $\Omega_i \cap \Omega_j$, esteja contido num subespaço de dimensão $d-2$ e tais que $\Omega = C(\Omega) \cap \left(\bigcap_{1 \leq i \leq w} \complement_E \Omega_i \right)$ onde $C(\Omega)$ designa a envolvente convexa de Ω e $\complement_E \Omega_i$ designa o complemento de Ω_i com relação ao conjunto universo E. Pode-se descrever a fronteira de Ω por meio de uma lista de pontos tais que, para cada um dentre eles e para cada poliedro Ω, o valor 1 ou 0 lhe seja atribuído caso pertença ou não à fronteira de Ω_i.

Deste modo, no exemplo da figura 1.4.9 onde $w = 3$, a fronteira de Ω pode ser descrita pela lista das coordenadas dos 27 pontos $P_1, \ldots, P_{27}$ e pela tabela anexa, que permite reconhecer em quais lugares a fronteira de Ω é "convexa" ou "côncava":

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9
Ω_1	0	0	0	0	0	0	0	0	0
Ω_2	1	0	0	0	0	0	0	0	0
Ω_3	0	0	0	0	0	0	0	0	0
	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}	P_{17}	P_{18}
Ω_1	0	0	0	0	0	0	0	0	0
Ω_2	0	0	0	0	0	0	0	0	1
Ω_3	0	0	0	0	1	1	1	1	1
	P_{19}	P_{20}	P_{21}	P_{22}	P_{23}	P_{24}	P_{25}	P_{26}	P_{27}
Ω_1	0	0	1	1	1	1	1	1	1
Ω_2	1	1	0	0	0	0	0	0	0
Ω_3	0	0	0	0	0	0	0	0	0

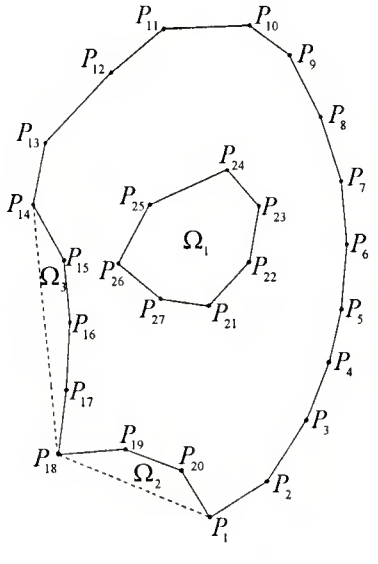

Figura 1.4.9: Fronteira de um poliedro quase convexo

1.4.4 Poliedro qualquer

A fronteira de Ω é descrita pela lista $F_1, \ldots, F_w$ que são suas faces externas e pela lista dos vértices dessas faces. Suponha-se, além disso, que essas faces são orientadas de uma mesma maneira (isto é, de uma maneira tal que seu vetor normal unitário seja dirigido até o exterior de Ω, por exemplo).

1.4.5 Geração de uma triangulação da envolvente convexa de Ω

Qualquer que seja o tipo do poliedro Ω, dispõe-se sempre de uma lista de pontos $P_1, \ldots, P_m$ que servem para definir a fronteira de Ω e entre os quais encontram-se todos os pontos extremos de Ω. Obviamente, esses pontos não pertencem a um mesmo hiperplano. Sendo assim, o algoritmo da seção 1.3 pode ser aplicado de maneira a obter uma triangulação de Delaunay ζ associada aos pontos $P_1, \ldots, P_n$. Sabe-se que ζ é uma triangulação da envolvente convexa dos pontos $P_1, \ldots, P_n$, isto é, da envolvente convexa de Ω.

1.4.6 Geração de uma triangulação "fronteiriça" de Ω

Definição 1.4.2 Entende-se por triangulação "fronteiriça" toda triangulação cujos vértices pertencem à fronteira de Ω. Se Ω for convexa, toda triangulação de Delaunay associada aos pontos $P_1, \ldots, P_n$ que descrevem a fronteira de Ω é uma triangulação "fronteiriça". Se Ω for não convexa precisa-se:

- Determinar as condições para que a triangulação ζ "respeite" a fronteira de Ω (isto é, para que toda face dessa fronteira seja também uma reunião de faces de elementos de ζ que não se sobrepõem).
- Determinar os elementos de ζ que não estão contidos em Ω.

Obtém-se dessa forma uma triangulação de Ω como subconjunto

de ζ.

1.4.7 Condições para que a triangulação de Delaunay ζ "respeite" a fronteira de Ω

Definição 1.4.3 Dir-se-á que a fronteira de Ω é bem descrita se nenhuma das esferas circunscritas às faces dessa fronteira contiver pontos de $P_1, \ldots, P_n$. Nota-se quando $d \leq 2$ e somente nesse caso, que a fronteira de um poliedro qualquer é sempre "bem descrita".

Apresenta-se dois exemplos de poliedro em dimensão 3 cuja fronteira é mal descrita (das pirâmides) na figura 1.4.10: as faces 123 e 134 devem ser substituídas pela face 1234 ou pelas faces 124 e 234 no segundo caso.

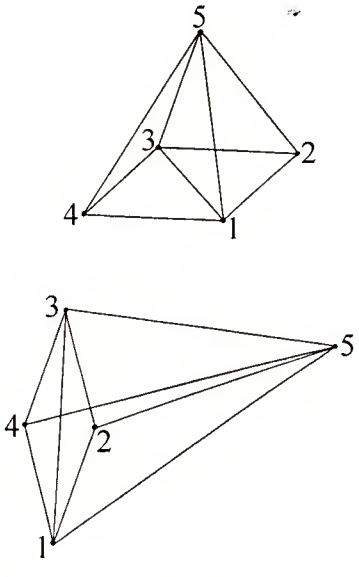

Figura 1.4.10: Poliedro cuja fronteira é mal descrita

Seja F uma face da fronteira de Ω e sejam E_1, E_2 os dois semi-espaços abertos cuja fronteira é o hiperplano determinado por F. Sejam X o conjunto dos pontos $P_1, \ldots, P_n$, dois conjuntos $X \cap E_1$ ou $X \cap E_2$, com pelo menos um deles não vazio já que os pontos de X não pertencem a um mesmo hiperplano. Suponha-se $X \cap E_1$ não vazio, seja P o ponto (ou um ponto) de $X \cap E_1$, tal que a esfera B circunscrita no simplexo PF não contém nenhum ponto de $X \cap E_1$.

Definição 1.4.4 Dir-se-á que a face F é singular se o conjunto $\bar{ß} \cap X \cap E_2$ for não vazio (onde $\bar{ß}$ denota a esfera fechada associada a B). Por definição, a fronteira de Ω será dita regular se nenhuma de suas faces for singular.

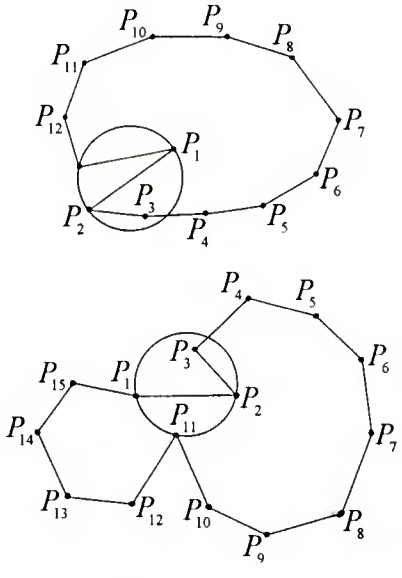

Figura 1.4.11: Polígonos cuja fronteira não é regular

Pode-se observar na figura 1.4.11 dois exemplos de polígonos cuja fronteira não é regular, a face $F = P_1P_2$ sendo singular.

Se a fronteira de Ω é bem descrita e regular, demonstra-se que a triangulação de Delaunay ζ respeita a fronteira de Ω [9]. Na prática existem poucos poliedros cuja fronteira não é regular. Se tivermos uma fronteira não regular, é preciso:

- Determinar todas as faces singulares;
- Acrescentar novos vértices (os extremos, por exemplo) de maneira a dividir essas faces singulares em várias novas faces;
- Recomeçar a primeira etapa se ainda existirem faces singulares.

A experiência mostra que este algoritmo converge, isto é, que as novas faces criadas tornam a fronteira regular.

1.4.8 Determinação dos elementos exteriores

Supõe-se então que Ω é um poliedro qualquer cuja fronteira é "bem descrita" e "regular". Trata-se de determinar os elementos da triangulação de Delaunay que não estão contidos em Ω.

No caso de poliedro quase convexo, lembre-se que $\Omega = C(\Omega) \cap \left(\bigcap_{1 \leq i \leq w} \complement_E \Omega_i \right)$ onde $\Omega_1, \ldots, \Omega_n$ são poliedros convexos tais que, para todo i, j com, $1 \leq i \neq j \leq n$, $\Omega_i \cap \Omega_j$ esteja contido num subespaço de dimensão $d-2$.

Um simplexo $K \in \zeta$ não está contido em Ω se todos os seus vértices pertencem à fronteira de algum dos poliedros $\Omega_1, \ldots, \Omega_n$. Logo, para cada simplexo $K \in \zeta$ e cada poliedro $\Omega_j, 1 \leq j \leq n$, basta fazer a soma s dos valores atribuídos aos vértices de K (lembre-se que esses valores são 0 ou 1, [5, 6, 9]).

Se essa soma s for igual a $d+1$, o elemento K não está contido em Ω. Para fixar melhor esse processo pode-se observar a figura 1.4.12.

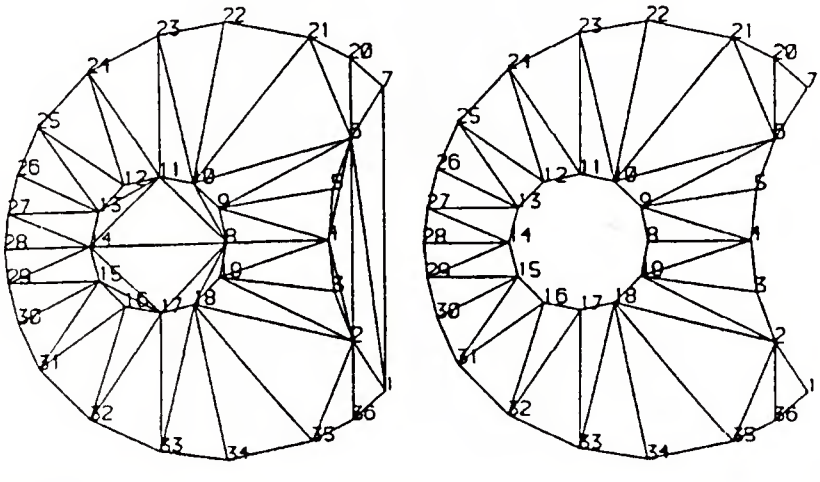

Figura 1.4.12: Localização de um simplexo em um poliédro quase convexo

No caso de poliedro qualquer, para saber se um simplexo $K \in \zeta$ está contido em Ω é preciso construir uma seqüência de simplexos $K_1, \ldots, K_n \in \zeta$ tal que:

- $K_1 = K$.

- Dois elementos consecutivos da seqüência tenham uma face comum que não está contida na fronteira de Ω.

- K_n seja o primeiro elemento que possui pelo menos uma face G contida em uma face F da fronteira de Ω, de tal modo que todos os vértices de G sejam também os vértices de F.

Suponha-se que $K_n = P_1 \dots P_{k-1} P P_{k+1} \dots P_{d+1}$ onde P é o vértice de K_n oposto à face G e sejam $Q_1, \dots, Q_d$ os d primeiros vértices da face F. Denota-se K', o simplexo $K' = Q_1 \dots Q_k P Q_{k+1} \dots Q_d$.

Uma vez que a face F é tal que sua normal unitária é dirigida até o exterior de Ω, o simplexo K_n está contido em Ω se e somente se $\det(K') > 0$. Pode-se mostrar que os elementos da seqüência $K_1, \dots, K_n$ estão contidos em Ω ou todos contidos em $\complement_E \Omega$, conseqüentemente $K_1 = K$ está incluso em Ω se e somente se $\det(K') > 0$. Uma ilustração desse processo é dado por meio da figura 1.4.13.

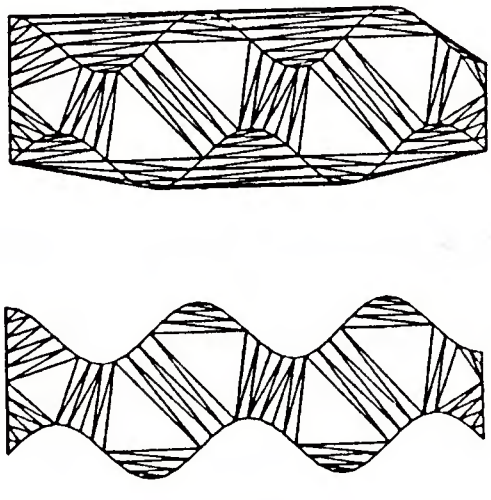

Figura 1.4.13: Localização de um simplexo em um poliedro qualquer

1.4.9 Criação eventual e tratamento dos pontos internos

De posse de uma triangulação fronteiriça é preciso, "refinar" essa triangulação, isto é, incluir no algoritmo da seção 1.3, um conjunto de pontos

internos $Q_1, \ldots, Q_t$ convenientemente distribuídos em Ω, de maneira a obter uma triangulação que se apoia nestes pontos e naqueles da fronteira e que verifica as condições 3 e 4 da seção 1.1.

Normalmente, os pontos internos $Q_1, \ldots, Q_t$ não são conhecidos e devem ser gerados por algoritmos que serão dados nesta seção.

Definição 1.4.5 Dir-se-á que um ponto Q é compat vel com a fronteira de Ω se Q não pertencer a nenhuma esfera fechada circunscrita a um elemento exterior de Ω. Um exemplo de ponto não compatível com a fronteira de um polígono Ω é dado por meio da figura 1.4.14.

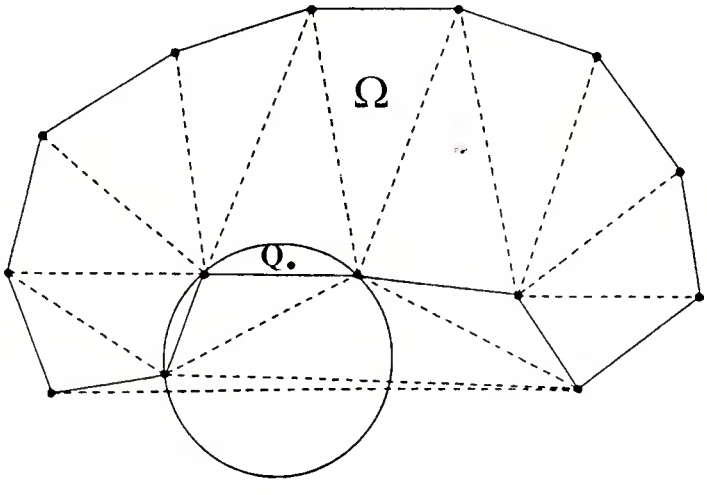

Figura 1.4.14: Exemplo de um ponto não compatível com a fronteira de Ω

Se os pontos $Q_1, \ldots, Q_t$ são compatíveis com a fronteira de Ω, é demonstrável que a triangulação de Delaunay associada aos vértices situados na fronteira e a esses pontos $Q_1, \ldots, Q_t$ "respeita" a fronteira de Ω [9].

Na prática não é restritivo exigir que os pontos $Q_1, \ldots, Q_t$ sejam "compatíveis" com a fronteira de Ω. De fato, se um ponto Q não for compatível, isso significa que a fronteira de Ω é mal descrita (poucos vértices dados para descrever a curvatura dessa fronteira) ou que Q é muito próximo da fronteira.

Suponha-se num caso particular que todos os vértices internos $Q_1, \ldots, Q_t$ da futura triangulação sejam fornecidos. O algoritmo da seção

1.3 gera uma triangulação de Delaunay ζ associada aos vértices situados na fronteira e aos pontos $Q_1, \ldots, Q_t$. Basta então eliminar os elementos de ζ exteriores à Ω para obter uma triangulação de Ω.

Observa-se que a "qualidade" dessa triangulação depende da escolha dos pontos $Q_1, \ldots, Q_t$.

Quando nenhum ponto interno é fornecido, é preciso criar um conjunto de pontos internos convenientemente distribuídos no interior de Ω. A idéia diretriz é a seguinte:

- Dada uma triangulação fronteiriça ζ de Ω, pode-se criar os primeiros pontos internos calculando os baricentros dos elementos de ζ cujo determinante é maior que os outros.

- Se eles existirem, eliminar os novos pontos que não são compatíveis com a fronteira de Ω.

- Aplicar o algoritmo da seção 1.3 para construir uma triangulação de Delaunay que inclua esses novos pontos.

- Recomeçar a primeira etapa para esta nova triangulação até a obtenção de um conjunto de pontos internos convenientemente distribuídos em Ω, e com isso, de uma triangulação de Delaunay satisfatória associada a estes pontos. Descreveremos de maneira mais detalhada estas diferentes etapas.

A cada vértice P da fronteira de Ω toma-se um valor $p(P)$ tal que $p(P)^d/d!$ corresponde ao valor desejável do volume dos simplexos que serão situados na vizinhança de P.

Na prática $p(P)$ foi escolhido como a média aritmética dos comprimentos das arestas fronteiriças originárias de P [3].

Seja ζ uma triangulação fronteiriça de Ω, para cada elemento $K = P_1 \ldots P_{d+1}$ de ζ associa-se o valor

$$VS(K) = \left(\prod_{1 \leq k \leq d+1} p(P_k) \right)^{1/d+1} \tag{1.5}$$

onde $(VS(K))^d/d!$ corresponde ao valor desejável do volume dos simplexos que serão situados na vizinhança de K.

Todos os elementos $K = P_1 \ldots P_{d+1}$ de ζ tais que:

$$\det(K) > (VS(K))^d \tag{1.6}$$

são então marcados com pontos de referência e para cada um destes elementos calcula-se o baricentro h dos pontos $P_1 \ldots P_{d+1}$ assossiados aos coeficientes,

$$\left(\sum_{1\leq k\neq j\leq d+1} p(P_k)\right) / d \left(\sum_{1\leq k\leq d+1} p(P_k)\right) \tag{1.7}$$

Se for compatível com a fronteira de Ω, este novo ponto (futuro ponto interno) é conservado, associado ao valor:

$$p(h) = \left(\prod_{1\leq j\leq d+1} p(P_j)\right)^{1/d+1}. \tag{1.8}$$

Obtém-se assim um conjunto de pontos internos compatíveis $Q_1 = \{Q_1, \ldots, Q_t\}$ tomados dos valores $p(Q_1), \ldots, p(Q_t)$.

O algoritmo da seção 1.3 é utilizado a fim de obter uma triangulação de Delaunay ζ_1, associada a esses pontos e aos vértices da fronteira.

O processo que acaba de ser descrito é em seguida aplicado à nova triangulação ζ_1.

Assim, obtém-se uma seqüência finita de conjuntos de pontos internos $Q_1, \ldots, Q_n$ e de triangulações de Delaunay $\zeta_1, \ldots, \zeta_n$ associadas aos vértices da fronteira e aos pontos dos conjuntos $Q_1, Q_1 \cup Q_2, \ldots, Q_1 \cup Q_2 \cup \ldots \cup Q_n$.

A triangulação ζ_n é a primeira tal que:

$$\forall K \in \zeta_n \det(K) \leq (VS(K))^d.$$

Na prática, o processo interativo que acaba de ser descrito permite construir um conjunto de pontos internos "convenientemente distribuídos" em Ω, de tal modo que a "densidade" desses pontos na vizinhança da fronteira corresponde à densidade dos pontos situados na fronteira.

Uma triangulação obtida por esse método pode ser vista por meio

da figura 1.4.15.

É possível também, fornecer certos pontos internos $Q_1, \ldots, Q_t$ e exigir a construção de uma triangulação conveniente, tendo $Q_1, \ldots, Q_t$ como vértices. O algoritmo precedente permite construir tais triangulações com a condição de atribuir aos pontos $Q_1, \ldots, Q_t$ os valores $p(Q_1), \ldots, p(Q_t)$ tais que $p(Q_j)^d / d!$ corresponde ao valor desejado do volume dos simplexos na vizinhança de Q_j.

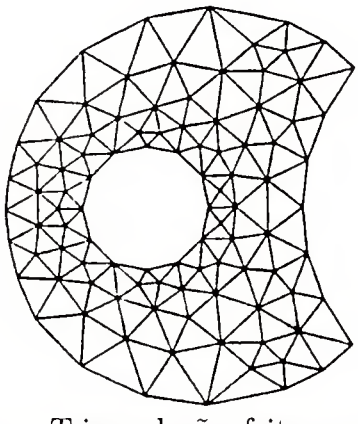

Figura 1.4.15: Triangulação feita quando nenhum ponto interno é fornecido

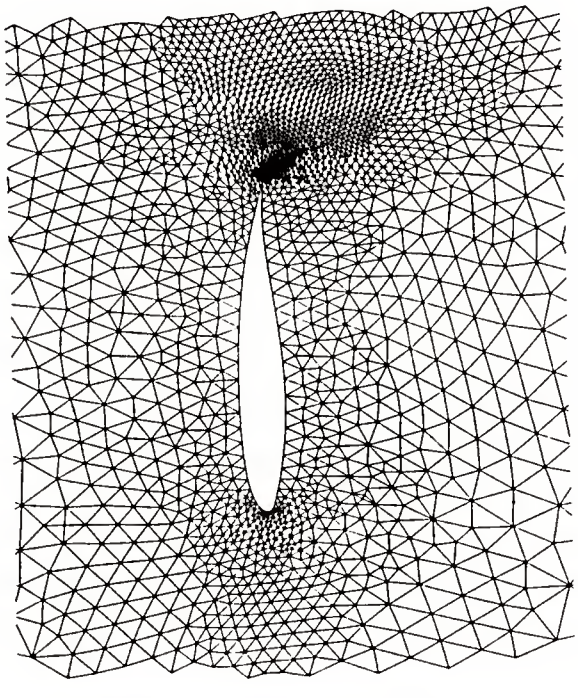

Figura 1.4.16: Triangulação para região de ar em simulação de aerofólio

A densidade dos pontos internos criados nas regiões do domínio que não estão situados na proximidade da fronteira pode ser alterada, conforme mostra a figura 1.4.16.

1.5 Implementações do algoritmo de Delaunay

De acordo com as definições encontradas nas seções 1.1 e 1.2, existe uma estreita correspondência entre polígonos, poliedros de Voronoi e a malha de simplexos ζ que se pretende gerar. Na figura 1.2.2 essa correspondência é perfeitamente ilustrada para o caso bi-dimensional: as linhas que ligam os P_i's, geram por sua vez a triangulação de Delaunay procurada.

Este resultado é perfeitamente generalizável para um espaço Euclidiano n-dimensional. Em geral, a triangulação acima gera simplexos n-dimensionais que gozam da propriedade de que a hiper-esfera circunscrita a eles não contem nenhum outro ponto do conjunto $P_1, \ldots, P_n$ que os $n+1$ que formam o simplexo n-dimensional. Essa propriedade é constantemente a chave para vários algoritmos que calculam a triangulação de Delaunay em três dimensões, pois a geração direta dos polígonos V_i segundo a definição 1.2.1 é impraticável do ponto de vista computacional. Dentre esses algoritmos, a chamada implementação de Watson [14] é a mais citada na literatura, como técnica de construção de uma malha de Delaunay. Explicita-se a seguir a implementação de Watson, através de operações matemáticas sobre as coordenadas cartesianas dos pontos do conjunto $P_1, \ldots, P_n$. No Apêndice A, encontra-se uma compararação das características desta implementação com as de outras técnicas de geração de malhas de Delaunay citadas na literatura.

1.5.1 Organizando a implementação de Watson em etapas

Até a seção anterior, estabeleceu-se as propriedades gerais da tetraedrização de Delaunay faltando agora explicitar a implementação de Wat-

son por meio de operações matemáticas sobre as coordenadas cartesianas dos pontos dados. Agora, nos restringiremos ao espaço tridimensional.

Etapas da implementação [2]**:**

1. São dados os n pontos $P_1, P_2, \ldots, P_n$ com suas coordenadas e índices de ordenação (posição em que os pontos aparecem na lista), esses pontos definem um sólido qualquer que se deseja decompor em células tetraédricas.

2. Determina-se um paralelepípedo englobante a partir do "extent" do sólido, calculando as coordenadas máximas $x_{\max}, y_{\max}, z_{\max}$ dos pontos $P_1, P_2, \ldots, P_n$ e majorando-as por um fator aleatoriamente escolhido, as arestas do paralelepípedo assumirão essas dimensões majoradas, para que os n pontos sejam contidos com folga dentro do paralelepípedo.

3. O paralelepípedo é decomposto em 6 tetraedros. Os n pontos estarão forçosamente espalhados por cada um deles, mas ainda precisam ser conectados de modo adequado.

4. O primeiro ponto da lista $P_1, P_2, \ldots, P_n$ é inserido na tetraedrização descrita na etapa 3 do seguinte modo: calcula-se o centro e o raio de todas as esferas que circunscrevem os tetraedros do passo 3, e verifica-se, se o ponto P_1 está incluído em alguma dessas esferas, verificando se sua distância ao centro de cada esfera é menor que o raio da esfera. Os tetraedros dentro das esferas que incluem o ponto P_1 são analisados, e suas faces comuns são eliminadas e novos tetraedros formados, unindo P_1 às faces restantes não comuns; os tetraedros dentro das esferas que não incluem P_1 são deixados como estão.

5. A nova decomposição obtida na etapa 4 é modificada recursivamente repetindo-se o processo para os demais pontos $P_2, \ldots, P_n$ até que todos estejam incluídos. Evidentemente, em cada inclusão vão aparecer novos tetraedros toda vez que faces comuns forem eliminadas.

6. Após a inclusão de todos os pontos da lista, restarão tetraedros com vértices que coincidem com vértices do paralelepípedo englobante "extent", ou seja, fora da região ocupada pelo sólido que esta sendo decomposto. Esses tetraedros são eliminados da lista de células da decomposição.

Estas são apenas as etapas fundamentais da implementação de Watson em domínios tridimensionais. Existem problemas adicionais que podem interferir na execução correta da decomposição. Os principais são o rompimento de ligações obrigatórias quando por exemplo, arestas essenciais à geometria inicial do sólido são destruídas, e uma interpretação confusa sobre a inclusão ou não de um ponto numa esfera (etapa 4) devido basicamente a erros de precisão nos cálculos. Além disso, tetraedros achatados ("slivers"), um outro problema de degenerescência da decomposição, podem chegar ao total de dez por cento das células geradas [15]. Discute-se esses problemas e sua influência na implementação computacional de Watson na seção seguinte.

1.5.2 Aspectos importantes na implementação de Watson

A base de dados mínima capaz de conter as informações necessárias ao funcionamento das etapas fundamentais da implementação são:

- Um conjunto de variáveis indexadas para guardar as coordenadas dos vértices dos tetraedros.
- Um conjunto de variáveis indexadas para guardar índices de referência dos vértices dos tetraedros.

Além dessas informações mínimas, a implementação é bastante acelerada pela adoção de c digos de vizinhan a, isto é, variáveis indexadas que indicam vizinhanças entre vértices (quem é vizinho de quem e sobre qual face).

Os testes de inclusão são feitos a partir das listas de vizinhos, não necessitando assim verificar todos os tetraedros para cada ponto inserido.

Como já vimos, pode acontecer que erros de precisão nos cálculos dos parâmetros das esferas perturbem a integridade da decomposição, quando isso ocorre tem-se tetraedros que se interpenetram, ou "overlapings" na malha.

Isso é evidenciado toda vez que a distância de um novo ponto

inserido a uma dada circunsfera é menor que η, onde η é o erro de truncamento acumulado pelo computador.

Uma cura para isso consiste em perturbar as coordenadas dos pontos para os quais a detecção de inclusão é afetada por uma incerteza da ordem de η (pontos sobre a fronteira da circunsfera, por exemplo), onde η agora é um erro de referência que arbitra-se a priori.

O teste de inclusão é repetido para as coordenadas perturbadas, e uma vez concluída a decomposição, todos os pontos que tiveram suas coordenadas alteradas são restaurados a sua posição original.

O outro problema de degenerescência de malha mencionado, é a ocorrência de tetraedros achatados ou mesmo com volume nulo, é tratado na literatura pela remoção do elemento afetado da lista da decomposição celular, reescrevendo as listas de coordenadas de modo adequado para não comprometer os tetraedros não afetados [15].

Em linhas bem gerais, estas seriam as curas mais simples para os problemas de consistência de malha mais graves.

O problema de rompimento de ligações obrigatórias pode ser tratado, numa primeira versão de um software interativo, através de uma ampliação e reordenação dos pontos de definição inicial dos volumes.

De fato, o rompimento de ligação obrigatória ocorre quando uma aresta da geometria inicial é incluída numa face de tetraedro que é depois eliminada na execução da implementação de Watson.

Por geometria inicial entende-se não apenas os limites externos de um objeto composto, mas também as interfaces entre sub-regiões de propriedades físicas diferentes. Em geral, não há qualquer razão para que a implementação de Watson respeite os limites dessas interfaces.

A solução para o problema entretanto é simples: se a introdução de um vértice implica rompimento de aresta obrigatória, introduz-se um novo vértice no ponto médio da aresta rompida, e aplica-se o algoritmo para inserir o novo vértice.

Tal técnica pode ser conduzida ou indiretamente, verificando em

tela, se a tetraedrização apresenta rompimentos que violam a geometria e inserindo os pontos médios necessários na coleção de pontos iniciais, ou inserindo um algoritmo de diagnóstico dentro do gerador de malha.

Tal algoritmo adicional deve simplesmente :

- Guardar uma lista das arestas que não podem ser rompidas.
- Comparar todas as arestas geradas numa dada decomposição em tetraedros com as arestas da lista acima.

Os rompimentos assim detectados são então tratados como descrito anteriormente.

Essa quebra das ligações obrigatórias é eventualmente mencionada na literatura como "efeito porco espinho".

Todos esses problemas de degenerescência são mais evidentes em poliedros regulares, e é essa a razão porque o cubo é freqüentemente usado para os testes de cura das degenerescências.

No presente trabalho, o cubo foi referência para todos os resultados obtidos.

Por meio dele, podemos tirar algumas conclusões de grande relevância .

Pudemos observar que boa parte dos casos de degenerescência podem ser evitados através de uma conveniente inserção dos pontos.

Veremos no Capítulo 3, critérios simples para ordens de inserção em algumas topologias de uso bem freqüente.

No caso do cubo,testado por nós, teremos sem nenhuma margem de erro uma malha degenerada se tomarmos os pontos de forma que ao menos 3 deles estejam em um mesmo plano. Isto nos dará uma malha em que encontramos tetraedros cuja intersecção é não vazia.

Os pontos neste caso, devem ser tomados diagonalmente, ou seja, de forma que divida cada plano em dois triângulos.

Para refinarmos esta malha, basta tomarmos os novos pontos nos baricentros de cada tetraedro da malha existente. Obviamente, sempre to-

mando o devido cuidado de não pegarmos em seqüência pontos no mesmo plano.

Embora trabalhar com a ordem conveniente de uma quantidade grande de pontos seja muitas vezes exaustivo, devemos superar essa deficiência a fim de conseguirmos uma malha mais fiel possível do sólido a ser discretizado.

Porém, nem sempre temos o mínimo de pontos necessários para se ter uma representação fiel do sólido em questão. No caso do cubo, o mínimo de pontos foi facilmente conseguido através dos seus vértices extremos. Já no que diz respeito a esfera e ao cilindro, não podemos tomar como pontos suficientes apenas a origem e pólos. Nestes casos, são necessários mais pontos a fim de obtermos um contorno perfeito dos respectivos sólidos.

Quanto à inserção dos pontos para estes dois exemplos devemos proceder da seguinte maneira :

- Na esfera deve-se inserir primeiro a origem e posteriormente os outros pontos em ordem crescente de raio. Como dito anteriormente, deve-se ter sempre cuidado para não tomar muitos pontos consecutivos em um mesmo plano.

- No cilindro deve-se inserir primeiro o centro e posteriormente os outros pontos de forma a não tomar muitos pontos consecutivos em um mesmo plano.

Também no cilindro, deve-se inserir o centro de cada circunferência. Desta forma, evitamos tetraedros degenerados nas extremidades pois cada vértice, será ligado com o centro.

1.6 Métodos de geração de malhas de Delaunay citadas na literatura

Método 1: (Shenton e Cendes) [16]

- Primeiro encontra-se o tetraedro que contém o novo ponto.
- Cria-se uma lista de faces usando as 4 faces do tetraedro encontrado e executando na lista de faces o seguinte teste:

Se a esfera circunscrita formada pela face e o novo ponto contém o último ponto do outro tetraedro associado com a face, adiciona-se as 3 outras faces deste tetraedro à lista de faces e coloca-se o novo tetraedro na lista de tetraedros deletados. Descarta-se a face testada.

- Continua-se até que a esfera circunscrita não inclua qualquer outro ponto.
- Posteriormente forma-se os novos tetraedros unindo cada face (dos poliedros formados pelas faces na lista de faces) com o ponto.

Método 2: [15]
Algoritmo de Watson's [16]

- Inicia-se com um tetraedro que contenha todos os pontos a serem adicionados; novos tetraedros internos são formados enquanto os pontos são introduzidos um a um.
- Num típico estágio do processo, um novo ponto é testado para determinar quais esferas circunscritas dos tetraedros existentes contém o ponto. Os tetraedros associados são removidos, restando uma região poliédrica contendo o novo ponto.
- Arestas unindo o novo ponto às faces triangulares da superfície de região poliédrica são criadas, definindo tetraedros que preenchem a região poliédrica. Combinando esses com os tetraedros externos da região poliédrica produz-se uma nova triangulação de Delaunay que contém o recente ponto adicionado.

Método 3:

- Encontra-se primeiro o tetraedro que contém o ponto.
- Se a esfera circunscrita formada pelos quatro vértices do tetraedro encontrado contém o quarto ponto do tetraedro vizinho, deleta-se esse tetraedro vizinho e descarta-se as faces comuns, restando uma região poliédrica que contém o novo ponto.
- Arestas unindo o novo ponto às faces triangulares da superfície de região poliédrica são criadas, definindo tetraedros que preenchem a região poliédrica. Combinando esses com os tetraedros externos da região poliédrica produz-se uma nova triangulação que contém o recente ponto adicionado.

Método 4:

- Encontra-se primeiro o tetraedro que contém o ponto.
- Encontra-se os vizinhos do tetraedro acima.
- Se as esfera circunscrita desses vizinhos contém o ponto então deleta-se esses vizinhos e o tetraedro encontrado.
- Deleta-se as faces comuns do tetraedro deletado, restando uma região poliédrica contendo o novo ponto.
- Arestas unindo o novo ponto às faces triangulares da superfície de região poliédrica são criadas, definindo tetraedros que preenchem a região poliédrica. Combinando esses com os tetraedros externos da região poliédrica produz-se uma nova triangulação que contém o recente ponto adicionado.

O desempenho computacional desses métodos pode ser visto, no Apêndice A.

1.6.1 Critério para comparação

A comparação dos métodos descritos na seção anterior, será feita através do seguinte critério:

- Medindo-se o tempo gasto para produzir as malhas tetraedricas para um dado conjunto de pontos. Isso será realizado considerando-se o número de tetraedros gerados.
- A qualidade das malhas. Mede-se a qualidade pela seguinte proporção:

  raio da esfera inscrita / raio da esfera circunscrita

  Se a proporção acima for menor do que EPSI então o particular tetraedro não é bom. Valor típico de EPSI é 0.01 [15].

1.6.2 Qualidade das malhas

A qualidade das malhas depende da distribuição dos pontos na região a ser discretizada. Para comparar os métodos vistos anteriormente usa-se o seguinte método.

Considera-se um cubo e gera-se os pontos dos lados do cubo usando o seguinte código:

```
N:= 0;
FOR k:= 0 TO kNN DO
FOR j:= 0 TO jNN DO
FOR i:= 0 TO iNN DO
IF (k = 0) OR (j = 0) OR (i = 0) OR (k = kNN)
OR (j = jNN) OR (i = iNN) THEN
BEGIN
N:= N+1;
X[N]:= i*lx;
Y[N]:= j*ly;
Z[N]:= k*lz;
```

```
END;
```

onde iNN, jNN e kNN são o número de divisões nas direções x, y e z respectivamente, e lx, ly e lz são os comprimentos dessas divisões. Então (i ×lx, j ×ly, k ×lz) são as coordenadas de um típico ponto do cubo.

Variando os valores iNN, jNN, kNN e lx, ly, lz pode-se gerar pontos nos lados do cubo. Com iNN = 1, jNN = 1 e kNN = 1 tem-se os oito pontos extremos do cubo. Usando esses oito pontos gera-se uma malha e então faz-se o seguinte:

- Toma-se um ponto no centro de um tetraedro e adiciona-se esse ponto na malha existente usando qualquer um dos métodos visto.

Por este processo pode-se adicionar qualquer número de pontos no interior do sólido. Depois de adicionar os pontos no interior do sólido calcula-se a proporção citada em 1.6.1 para todos os tetraedros da malha.

Se a proporção é menor que EPSI, então esse tetraedro não é bom (sliver [15]). A porcentagem do pior tetraedro nos dá uma medida da qualidade da malha [15].

Detalhes sobre a geometria analítica no espaço, essenciais para a implementação dos algoritmos vistos neste capítulo, são revistas no Apêndice B.

Capítulo 2

A implementação de Watson para o algoritmo de Delaunay

Neste capítulo, trataremos do aspecto computacional da implementação de Watson, dando as principais funções codificadas no ambiente Mathematica for Windows para executar e visualizar as malhas de tetraedros.

2.1 O ambiente Mathematica

O pacote Mathematica é um software largamente empregado por engenheiros e cientistas como uma ferramenta para desenvolvimento de aplicações em simulação computacional.

Mathematica é dotado de vários recursos para cálculos algébricos, númericos e visualização gráfica. De fato, ele pode ser considerado como um ambiente de programação completo, onde tanto o estilo de programa funcional quanto o interpretativo podem ser usados para escrever longas funções e bibliotecas.

Quando algumas aplicações requerem grande tempo de CPU, as partes críticas de um código podem ser escritas como uma rotina em FOR-

TRAN ou C que trocam informações com o ambiente Mathematica.

O Mathematica produz gráficos em duas e três dimensões, bem como contornos e equipotenciais. Pode-se plotar tanto funções quanto listas de dados. Ele é dotado de muitas opções para controlar a saída de gráficos. Algumas versões do Mathematica também suportam animações gráficas [18].

Todos os gráficos produzidos no Mathematica estão no padrão PostScript, e podem ser transportados para uma ampla variedade de programas.

Por fim, o Mathematica pode ler dados em vários formatos, e pode gerar saídas para compiladores como C, FORTRAN e TEX. Maiores detalhes sobre os recursos de programação nesse ambiente podem ser encontrados em [18, 19, 20]

2.2 A implementação de Watson no ambiente Mathematica

Inicialmente, foram condificadas algumas funções avulsas, que utilizam equações paramétricas para gerar nuvens de pontos pertencentes a alguns tipos básicos de sólidos.

Esses pontos são armazenados em uma lista de triplas, onde cada tripla fornece as coordenadas espaciais de um ponto da nuvem, por exemplo:

$$\{\{1,1,1\},\{1,1,2\},\{1,2,1\},\{1,2,2\},\{2,1,1\},\{2,1,2\},\{2,2,1\},\{2,2,2\}\}.$$

Evidentemente, o nosso objetivo é testar a discretização em algumas topologias básicas de sólidos. Através de uma concatenação adequada dessas listas, figuras complexas podem ser montadas a partir de uns poucos sólidos simples [21].

A título de ilustração, damos a seguir uma dessas funções, empregada na geração de um toróide, e um gráfico da nuvem de pontos gerada por

ela já com o extent:

```
lcord:=Table[{N[4+(3+Cos[u])*Cos[t]],N[4+(3+Cos[u])*Sin[t]],
N[4+Sin[u]]},{u,0,2*Pi,0.5},{t,0,2*Pi,0.5}];
T=Flatten[lcord];
toro=Partition[T,3];Print["toro=",toro];
pts=Map[Point,toro];
H=Map[Graphics3D,pts];
Show[H,Boxed->False];
H
```

Código2.2.1

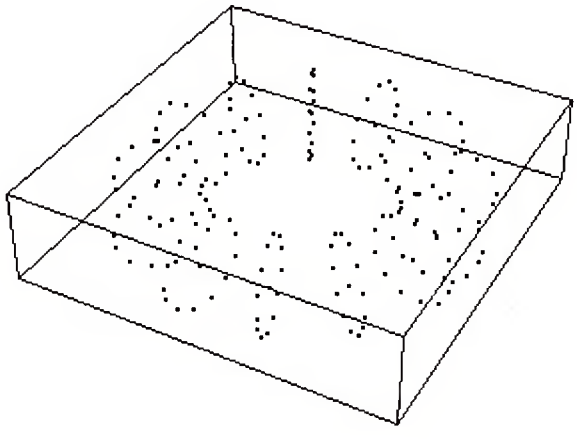

Figura 2.2.1: Nuvem de pontos para discretização de um toróide

Com base nas etapas fundamentais da implementação de Watson em domínios tridimensionais (seção 1.5.1), desenvolvemos uma função principal no ambiente Mathematica, que executa a implementação de Watson por meio de outras rotinas subordinadas.

Faremos agora um estudo detalhado de cada uma dessas rotinas subordinadas. Primeiramente apresentaremos a função principal que é dada

por:

```
Do[trip=lcord[[i]];
px=trip[[1]];
py=trip[[2]];
pz=trip[[3]];
   Do[lindex=tetra[[j]];
                Do[pi[[k]]=lindex[[k]];
                pts=lcord[[pi[[k]]]];
                xx[[k]]=pts[[1]];
                yy[[k]]=pts[[2]];
                zz[[k]]=pts[[3]],
                {k,1,4}
                    ];
   spher[xx,yy,zz];
   R2=(a^2+b^2+c^2-4d)/4.;
   R=(px+a/2)^2+(py+b/2)^2+(pz+c/2)^2;
   If[R<R2,fac=addfac[fac,lindex];T[[j]]=0];
   {j,1,Length[tetra]}
      ];
tetraold=deltetra[tetra,T];
delcomfac[fac];
If[tetra=={},tetra=t,tetra=Join[tetraold,t]];
T=Table[1,Length[tetra]];fac={},
{i,9,Length[lcord]}
]
```

Código 2.2.2

A função acima precisa ler somente como argumento a lista lcord que descreve a nuvem de pontos do sólido.

Os oito primeiros pontos da lista lcord correspondem aos vértices do extent e as triplas restantes são os pontos que serão inseridos na malha.

Assim, a lista lcord é da forma:

lcord={{_,_,_},{_,_,_},...,{_,_,_},{_,_,_},...}

A próxima lista que aparece na função principal é a lista tetra que é composta pelas quadruplas que formam a malha inicial citada na seção 1.5.1.

Logo, a lista tetra é inicializada com a seguinte forma:

tetra = {{1,2,3,6},{1,3,4,6},{1,4,5,6},
{3,4,6,7},{4,6,7,5},{4,7,8,5}}.

Cada inteiro que compõem as quadruplas corresponde à posição das triplas da lista lcord, isto é, cada quadrupla de tetra contém os números de ordem que correspondem aos vértices de um dado tetraedro da malha.

Por exemplo, se tomarmos da lista tetra a quadrupla 1,2,3,6 ela nos dará as triplas de posição 1,2,3 e 6 na lista lcord.

As funções subordinadas chamadas no corpo da função principal são: spher[xx,yy,zz], addfac[fac,lindex], deltetra [tetra,T] e delcomfac[fac].

A função spher[xx,yy,zz] lê todas as coordenadas dos pontos associados a uma quadrupla da lista tetra e os parâmetros a, b, c, d associados à equação da esfera que passa pelos 4 pontos. Há uma função pré existente no ambiente Mathematica que é chamada no corpo da spher[xx,yy,zz] (LinearSolve) e que resolve sistemas lineares.

```
spher[xx:{_,_,_},yy:{_,_,_},zz:{_,_,_}]:=
Module[{m,const},
m={{xx[[1]],yy[[1]],zz[[1]],1},
      {xx[[2]],yy[[2]],zz[[2]],1},
      {xx[[3]],yy[[3]],zz[[3]],1},
      {xx[[4]],yy[[4]],zz[[4]],1}};
const={-xx[[1]]^2-yy[[1]]^2-zz[[1]]^2,
        -xx[[2]]^2-yy[[2]]^2-zz[[2]]^2,
        -xx[[3]]^2-yy[[3]]^2-zz[[3]]^2,
        -xx[[4]]^2-yy[[4]]^2-zz[[4]]^2};
```

```
vars=LinearSolve[m,const];
sol=Flatten[vars];
a=sol[[1]];
b=sol[[2]];
c=sol[[3]];
d=sol[[4]]
]
```

Código 2.2.3

Se um dado ponto for detectado no interior de uma circunsfera a função addfac[fac,lindex] é chamada.

Essa função recebe como argumento uma lista de faces que no começo da discretização é inicializada como fac= 0,0,0 e lindex que é um elemento de tetra (lembremos que um elemento de tetra é da forma · ,· ,· ,·). O processamento realizado por addfac[fac,lindex] consiste em montar todas as combinações dos quatro inteiros que compõem lindex, três a três e adicioná-los um a um na lista fac. Deste modo, a lista fac agora contém as faces do novo tetraedro.

```
addfac[fac_,lindex_]:=Module[{aux1=fac},
imax=Length[lindex];
aux2=Table[Drop[lindex,{i}],{i,1,imax}];
Last[Map[AppendTo[aux1,#]&,aux2]]]
```

Código 2.2.4

Inicializamos também uma lista denotada por T como sendo T= 1,1,1,1,1,1 . Esta lista tem sempre o mesmo comprimento da lista tetra e armazena inteiros de valor 0 ou 1, dependendo do teste de inclusão dos pontos ser verdadeiro ou falso. Suponha por exemplo, que um dado ponto esteja no interior de uma circunsfera, então, a posição k associada ao k-ésimo tetraedro da lista T, passa de 1 para 0.

Após obtermos a nova lista fac, uma nova função é chamada para eliminar todas as faces comuns existentes nessa lista.

Essa função é a delcomfac[fac], ela recebe como argumento a lista fac, ordena as triplas contidas nessa lista e então remove todas as triplas iguais.

```
delcomfac[fac_List]:=
{
num=0;
ordfac=Map[Sort,fac];
n=Length[ordfac];
cnt=Table[aux[i],{i,1,n}];
Do[
elem=ordfac[[z]];
sed=Position[ordfac,elem];
cnt[[z]]=Length[sed],
{z,1,n}];
mut=Count[cnt,1];
cnt2=Table[aux[i],{i,1,mut}];
Do[
If[cnt[[z]]>=2,cnt[[z]]=cnt[[z]],
num=num+1;cnt2[[num]]=z],
{z,1,n}];
fac2=ordfac[[cnt2]];
t=Table[axx[m],{m,1,Length[fac2]}];
Do[t[[r]]=Append[fac2[[r]],i],
{r,1,Length[fac2]}];
}
```

Código 2.2.5

A função deltetra[tetra,T] que recebe como argumento a lista tetra

e a lista T deleta da lista tetra todos os tetraedros correspondentes à T[[j]]= 0 pois, suas faces já estão incluídas na lista fac.

Da função deltetra, obtemos uma lista chamada tetraold (fazendo tetraold= deltetra[tetra,T]) que contém os tetraedros para os quais o ponto testado não passou no teste de inclusão.

```
deltetra[tetra_List,T_List]:=Module[{aux},
aux={};
Table[If[T[[j]]==1,AppendTo[aux,tetra[[j]]]],
{j,1,Length[T]]};
Return[aux]]
```

Código 2.2.6

Os novos tetraedros são formados pela inserção do ponto testado em cada tripla da lista fac. Esse processo ocorre no corpo da função delcomfac[fac] por uma lista auxiliar t que retem esses novos tetraedros.

De posse dessas duas listas tetraold e t formados a nova lista tetra conforme o If correspondente na função principal.

Para essa nova lista tetra testamos o próximo ponto da lista lcord e assim, procederemos até que o último ponto da lista lcord seja inserido na malha.

A discretização estará concluída quando todos os tetraedros da lista tetra que tem algum vértice em comum com os vértices do extent forem deletados. Esse processo é realizado pela função elimina[tetra] que está codificada a seguir.

```
elimina[tetra_List]:=Module[{aux,aux1},
aux1={};
Table[
x=True;
aux=tetra[[v]];
For[u=1,u<5,u++,If[aux[[u]]>0&&aux[[u]]<9,x=False]];
```

```
If[x==True,AppendTo[aux1,aux],];
,{v,1,Length[tetra]}];
Return[aux1]]
```

Código 2.2.7

Para visualizarmos a malha devemos trabalhar com duas funções: Comb[tetra] e Plot4Edro[lcord,tetra].

A função Comb[tetra] forma as faces do tetraedro que serão plotadas por Plot4Edro[lcord,tetra].

Notemos que a lista tetra esperada como argumento em Plot4Edro[lcord,tetra] é a lista devolvida por elimina[tetra].

Essas duas funções estão codificadas a seguir.

```
Comb[tetra_List]:=
{
aux1={};
Do[aux=tetra[[i]];
aux2={{aux[[1]],aux[[2]],aux[[3]]},
                {aux[[1]],aux[[2]],aux[[4]]},
                {aux[[1]],aux[[3]],aux[[4]]},
                {aux[[2]],aux[[3]],aux[[4]]}};
Map[AppendTo[aux1,#]&,aux2],{i,1,Length[tetra]}];
h=Flatten[aux1];
H=Partition[h,3];
}
```

Código 2.2.8

```
Plot4Edro[lcord_,tetra_]:=
{
Comb[tetra];
H;
```

```
H=H/.{x_Integer->lcord[[x]]};
H=Map[Polygon,H];
H=Map[Graphics3D,H];
Show[H,Boxed->False];
H
}
```

Código 2.2.9

Estas funções evidentemente não são nem a única, nem a mais eficiente solução possível no ambiente Mathematica para implementar o algoritmo de Delaunay.

A rigor, elas combinam os dois estilos de programação (imperativo e funcional), que possuem desempenhos distintos no ambiente quando consideramos o tempo de CPU dispendido.

Contudo, uma vez verificada a sua correção, elas são um passo essencial na obtenção de funções mais rápidas e capazes de trabalhar com técnicas de armazenamento mais eficientes, permitindo a discretização de grandes malhas.

No capítulo seguinte, apresentaremos vários exemplos que ilustram o desempenho e a correção das funções aqui apresentadas.

Capítulo 3

Testes de validação

Até o presente momento, fizemos um estudo detalhado da matemática da implementação de Delaunay, comparamos a implementação de Watson com outros algoritmos e demos algumas noções dos problemas surgidos em domínios tridimensionais, bem como possíveis curas para esses problemas.

Neste capítulo, faremos a validação da implementação de Watson fazendo uso de alguns sólidos discretizados pelo algoritmo do Capítulo 2.

Faremos uma abordagem exemplificando cada problema mencionado no Capítulo 1 e maneiras de se evitar cada um deles. Discutiremos o tempo de processamento gasto em cada um dos exemplos discretizados por nós.

3.1 Validação em sólidos regulares

Como mencionado anteriormente (1.5.1), o algoritmo de Delaunay em domínios tridimensionais permite o surgimento do que chamamos degeneresc ncia. Isso ocorre freqüentemente em sólidos com geometria regular. Inicialmente consideraremos o caso do cubo.

Na figura 3.1.1, apresentamos um tipo de degenerescência conhe-

cida na literatura como *overlaping*, isto é, ocorrência de tetraedros que se interpenetram.

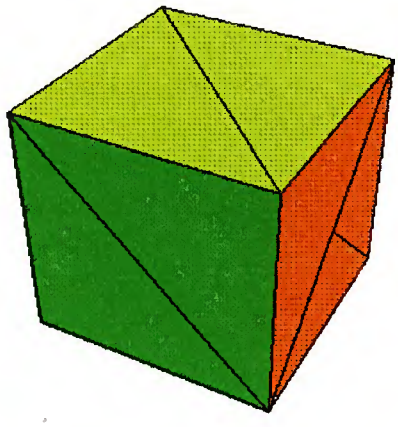

Figura 3.1.1: Cubo com superposição de elementos

Uma cura possível para essa degenerescência, embora não funcione para todos os casos, é a modificação na ordem de inserção dos pontos [4, 22]. Para o cubo da figura 3.1.1 os pontos a serem inseridos são:

$$\{1,1,1\},\{1,1,2\},\{1,2,1\},\{1,2,2\},\{2,1,1\},\{2,1,2\},\{2,2,1\},\{2,2,2\}.$$

Na orientação dada aos eixos pelo Mathemática esses pontos tem a seguinte distribuição:

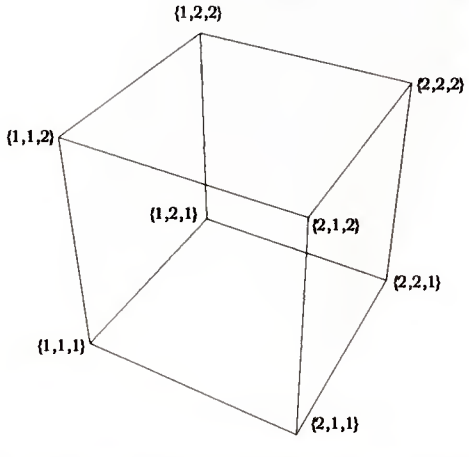

Figura 3.1.2: Coordenadas espaciais para o cubo da fig. 3.1.1

No cubo da figura 3.1.1 esses pontos foram inseridos na seguinte ordem:

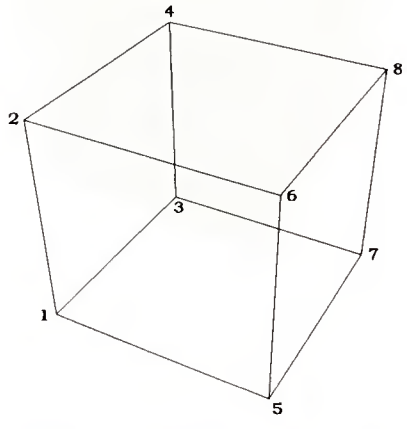

Figura 3.1.3: Ordem de inserção para o cubo da fig. 3.1.1

Note que os quatro primeiros pontos foram inseridos em um mesmo plano, o mesmo ocorreu para os pontos restantes.

Essa ordem de inserção gera a degenerescência apresentada na figura 3.1.1, podendo ser melhor observada por meio da figura 3.1.4.

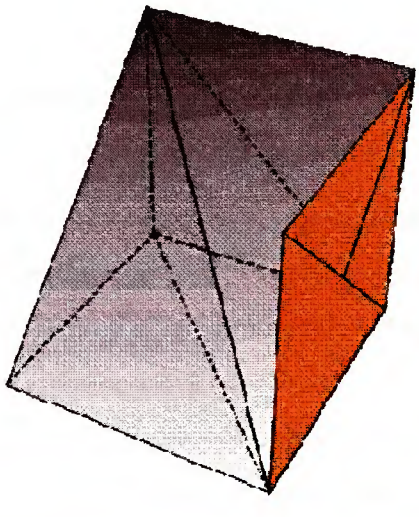

Figura 3.1.4: Visualização interna do cubo 3.1.1

A figura 3.1.5 apresentada a seguir ilustra o mesmo cubo da figura 3.1.1 porém, com a ordem de inserção modificada.

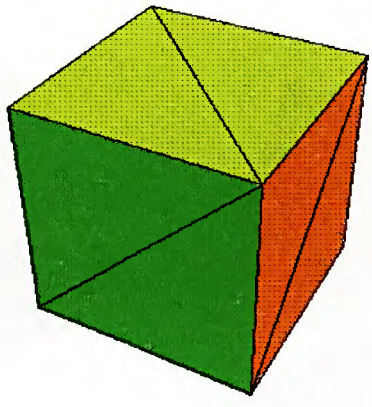

Figura 3.1.5: Cubo sem falhas na malha

Os pontos da figura 3.1.5 foram inseridos na seguinte ordem:

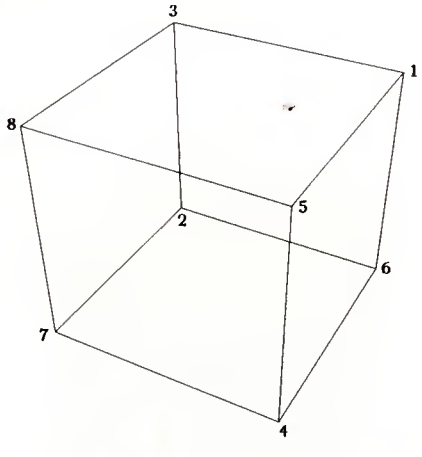

Figura 3.1.6: Ordem de inserção para o cubo da fig. 3.1.5

Observe porém, que inserimos três pontos em um mesmo plano mas isso não acarretou nenhum tipo de degenerescência. O critério adotado aqui foi sempre tomar os pontos sobre diagonais. Pode-se observar melhor o cubo perfeito por meio da figura 3.1.7.

De posse da malha da figura 3.1.5 (isto é, uma malha sem degenerescência) podemos inserir novos pontos e assim gerar novos tetraedros de modo a obter um refinamento da malha inicial (malha da figura 3.1.5).

Esses novos pontos a serem inseridos foram sempre tomados sobre diagonal como mencionado acima.

A figura 3.1.8 mostra um pequeno refinamento do cubo da figura

3.1.5.

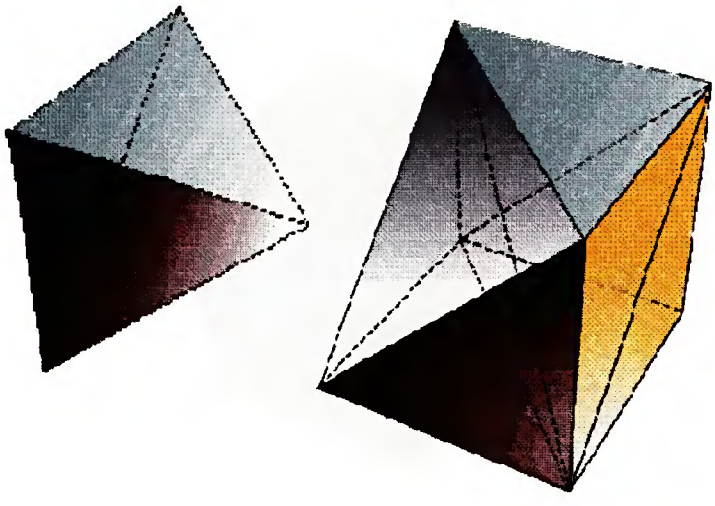

Figura 3.1.7: Visualização interna da figura 3.1.5

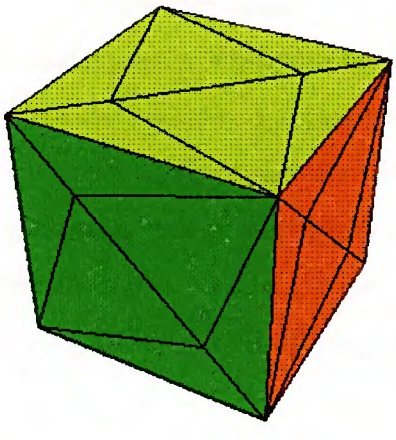

Figura 3.1.8: Refinamento na malha da fig. 3.1.5

Lembremos que o principal teste feito na função de inserção da seção 2.2 (código 2.2.2), utiliza os raio das esferas que circunscrevem os tetraedros. Portanto, se tomarmos os novos pontos por exemplo nos baricentros dos tetraedros da malha existente, isto é, distantes da fronteira da esfera que circunscreve o tetraedro, e seguir o modo de inserção dos pontos citados anteriormente, a malha obtida será uma malha sem degenerescência. Isso pode ser notado por meio da figura 3.1.8.

A figura 3.1.9 ilustra uma tentativa de refinamento do cubo da

figura 3.1.5 com os pontos sendo tomados de forma aleatória.

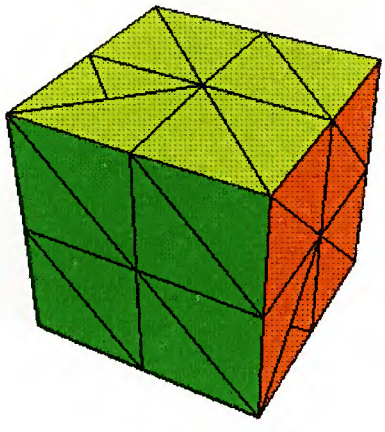

Figura 3.1.9: Refinamento com falhas de superposição

Notemos que os pontos não foram tomados nos baricentros dos tetraedros da malha pré-existente e sim no ponto médio de cada lado do cubo.

Através da ordem de inserção dos pontos e de como construímos os mesmos (no que diz respeito à dimensão das coordenadas dos pontos), podemos ter diferentes variantes para a topologia do cubo. Um exemplo disso pode ser visto através das figuras 3.1.10 e 3.1.11 que ilustram um mesmo paralelepípedo porém com os pontos inseridos em ordem diferente.

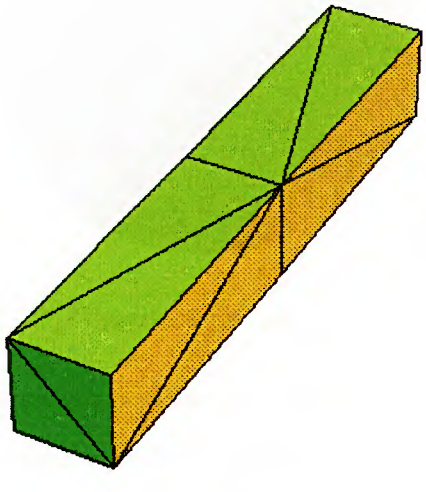

Figura 3.1.10: Malha para um paralelepípedo teste

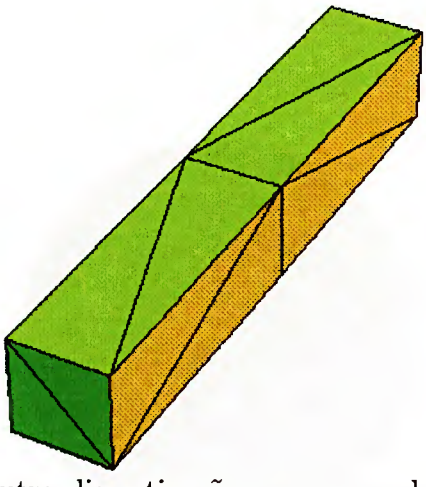

Figura 3.1.11: Outra discretização para o paralelepípedo da fig. 3.1.10

Ainda testando as topologias básicas, resolvemos trabalhar também com a esfera e o cilindro, por serem sólidos freqüentemente encontrados em modelagem geométrica de domínios.

Embora o número de pontos iniciais necessários, a serem inseridos em ambos os sólidos, seja bem maior que no cubo, eles são de mais fácil discretização que o mesmo.

A figura 3.1.12 ilustra uma esfera com 63 pontos.

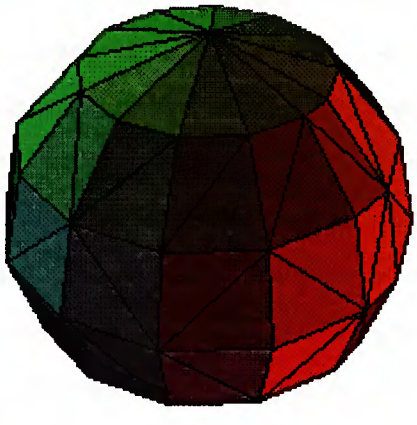

Figura 3.1.12: Discretização num domínio esférico

Podemos observar por meio da figura 3.1.12 que essa quantidade de pontos é insuficiente para obtermos fielmente uma esfera.

A mesma esfera agora com 115 pontos está ilustrada na figura

3.1.13. Por meio dela, podemos constatar a importância da quantidade de pontos.

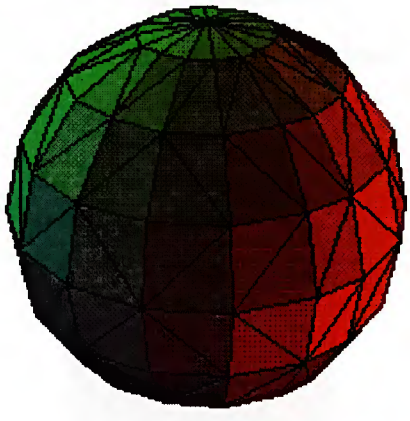

Figura 3.1.13: Discretização refinada para a fig. 3.1.12

Na esfera tivemos a ocorrência da degenerescência tratada na literatura como tetraedros *slivers*, isto é, tetraedros achatados (com volume nulo). Esse tipo de degenerescência eventualmente não afeta a qualidade da malha por se tratar de tetraedros com volume nulo e não de tetraedros que se interpenetram (*overlapings*).

Os tetraedros *slivers* podem ser retirados da lista de tetraedros (gerados pela função principal da seção 2.2 "código 2.2.2 ") sem prejuízo algum para a qualidade da malha [15].

A inserção dos pontos na esfera deve ser sempre em ordem crescente de raio e pontos consecutivos tomados em planos diferentes.

Uma maneira eficiente de se evitar tetraedros *slivers* e *overlapings* na malha é tomar sempre a origem da esfera e inserí-la primeiramente.

Após inserir a origem, devemos inserir os pontos do próximo raio e assim procederemos até os pontos do raio mais externo serem todos inseridos.

Nas figuras 3.1.12 e 3.1.13 as esferas foram geradas apenas pelo ponto origem e pontos cujo raio é 1.

Para o cilindro, o ponto origem de cada secção circular deve ser inserido primeiro, em seguida, são inseridos os pontos restantes sempre em planos distintos.

No cilindro além das exigências acima, às vezes tivemos que usar do artifício de perturbar as coordenadas de alguns pontos, para evitar degenerescências e manter a geometria do sólido [23].

Na figura 3.1.14 apresentamos um cilindro com superposição de tetraedros (*overlaping*).

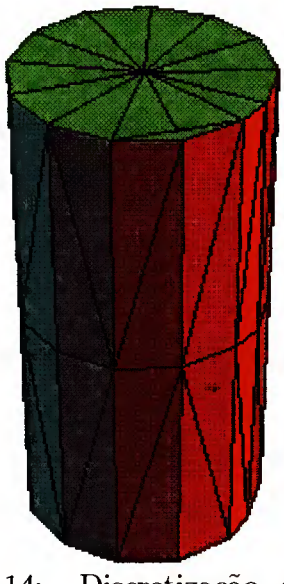

Figura 3.1.14: Discretização com superposição de elementos

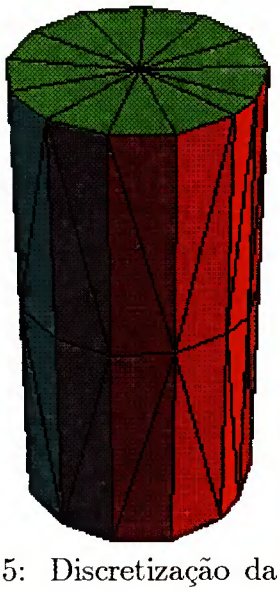

Figura 3.1.15: Discretização da fig. 3.1.14 corrigida

Como mencionado anteriormente, basta perturbar ligeiramente as coordenadas do ponto problema para obtermos um cilindro discretizado perfeitamente. O ponto problema neste caso é $\{2.70867, 1.29446, 6\}$. Se perturbarmos ligeiramente este ponto para $\{2.70867, 1.29447, 6\}$, teremos o cilindro visto na figura 3.1.15.

Por meio da figura 3.1.15 podemos observar a existência de três níveis de z. Neste cilindro z assume os valores 2, 4 e 6. Para refiná-lo, basta ir inserindo pontos em outros níveis de z por exemplo, 5 e 3.

Devemos lembrar que esses pontos não foram inseridos aleatoriamente e sim nos padrões ditos anteriormente.

3.2 Validação com pontos randômicos

Já introduzimos no Capítulo 1 a noção de que o algoritmo de Delaunay mesmo em domínios tridimensionais, tem excelente performance em sólidos não regulares.

De um modo especial, os pontos randômicos possuem essa propriedade, ou seja, quando aplicamos a eles o algoritmo de Delaunay, obtemos uma malha perfeita, isto é, sem nenhum tipo de degenerescência.

Fizemos testes com 100, 150 e 200 pontos distriuídos aleatoriamente no espaço. As malhas dos respectivos pontos estão ilustradas nas figuras 3.2.16, 3.2.17 e 3.2.18 respectivamente.

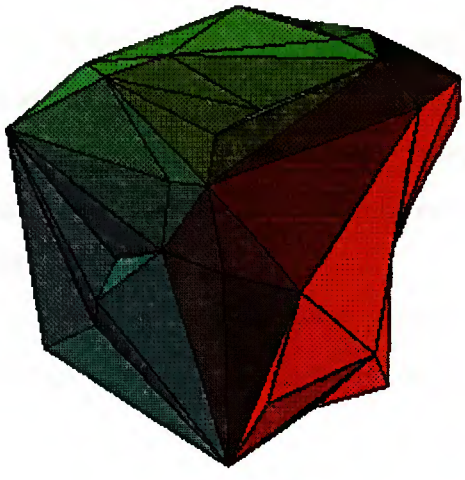

Figura 3.2.16: Discretização de 100 aleatórios

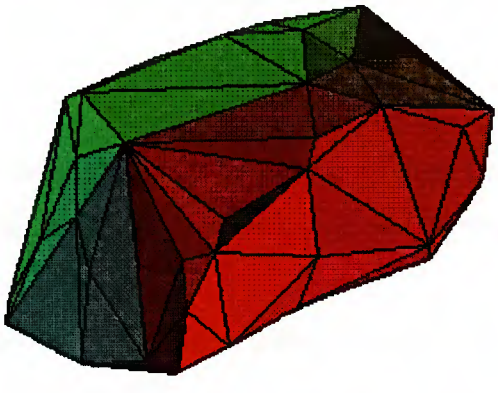

Figura 3.2.17: Discretização de 150 pontos aleatórios

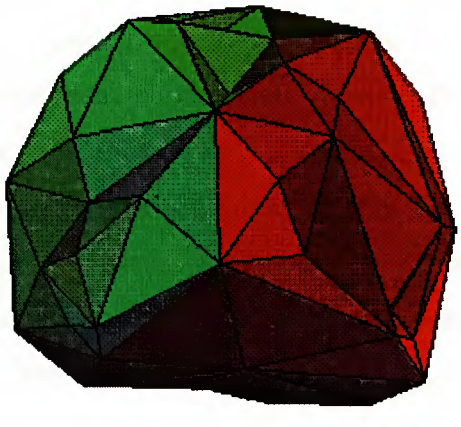

Figura 3.2.18: Discretização de 200 pontos aleatórios

Analisando as figuras podemos observar que por mais estranhas

que sejam as formas geradas pelos pontos (concavidades, curvas, bicos), o algoritmo da seção 2.2 nos fornece uma malha sem nenhuma degenerescência.

Com pontos randômicos, é menor a probabilidade de termos problemas com inserção de pontos, isto é, qualquer que seja a ordem dos pontos o algoritmo será executado com sucesso (nos referimos a sucesso quando obtemos uma malha perfeita).

De fato, a literatura confirma que domínios formados por pontos randômicos são menos sujeitos a esse tipo de ocorrência (pontos com degenerescência) [24].

3.3 Validação em espaços multiplamente conexos

Também consideramos a validação do algorítmo de Delaunay para espaços multiplamente conexos, cuja definição pode ser encontrada no Apêndice C.

Fizemos teste de validação com o toro para mostrar que mesmo se tratando desse tipo de geometria, o algoritmo de Delaunay é viável.

Um dos grandes problemas enfrentados com o algoritmo de Delaunay é a quebra da geometria.

Em sólidos como o toro, o algoritmo de Delaunay, quando executado, não consegue respeitar a geometria do mesmo. Ele não consegue tratar, os buracos ou quebras na geometria.

Podemos observar esse tipo de problema por meio da figura 3.3.19.

A figura 3.3.19, mostra claramente que o código da seção 2.2 discretiza todo o sólido sem levar em consideração a convexidade do toro. Para curar esse tipo de problema procedemos da seguinte maneira:

Primeiro discretizamos todos os pontos sem nos preocupar com a geometria do sólido.

Se houver algum tipo de degenerescência, precisamos curá-las (essas curas já foram discutidas nas validações anteriores).

Após obtermos uma malha perfeita no sentido de não termos nenhuma degenerescência, precisamos detectar todos os tetraedros cujos vértices estão na fronteira da região mais interna do toro e deletá-los.

Devemos observar a importância de inserir o ponto médio em cada uma das circunferências formadoras do toro com o objetivo de evitar tetraedros *slivers*. Logo, todos os tetraedros que possuem um desses pontos (os pontos médios) como vértice e pontos da fronteira mais externa do toro, isto é, pontos que não fazem parte da concavidade interna do toro devem permanecer. Somente serão deletados os tetraedros cujos vértices são todos da parte convexa do toro.

Assim procedendo, teremos uma malha respeitando a geometria do toro. Essa malha pode ser vista na figura 3.3.20.

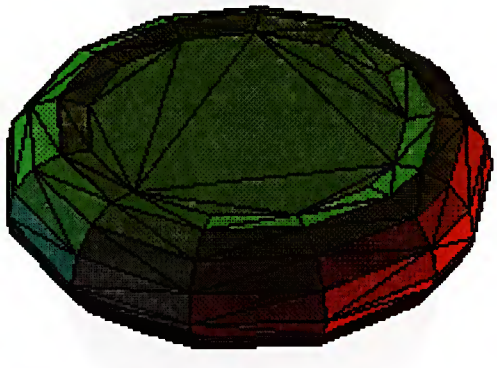

Figura 3.3.19: Discretização com ruptura da geometria inicial de um toróide

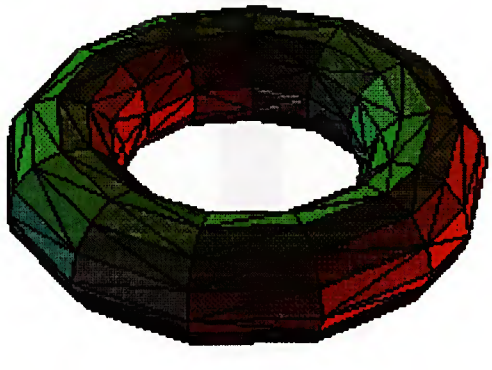

Figura 3.3.20: Discretização da fig. 3.3.19 corrigida

Um outro exemplo de geometria complexa em que ocorre a necessidade de um tratamento posterior da malha, com vistas à reconstituição da geometria original pode ser visto por meio da figura 3.3.21. Após esse tratamento, obteremos o sólido em estudo que pode ser visto por meio da figura 3.3.22.

Figura 3.3.21: Discretização com quebra da geometria de uma figura composta

Figura 3.3.22: Discretização de uma figura composta

O domínio da figura 3.3.22 foi gerado por meio da superposição de sólidos básicos (esfera e cilíndro), procurando ilustrar como casos mais complexos podem ser eventualmente tratados a partir de primitivas geométricas mais simples.

3.4 Desempenho computacional

Para cada um dos sólidos citados anteriormente avaliamos o tempo de processamento (que denotaremos por T.P.) e o número de tetraedros gerados para cada número de pontos inseridos. Essas avaliações estão ilustradas na tabela abaixo.

sólido	n° de pontos	n° de tetraedros	T.P.
cubo perfeito	8	6	1.16 s
cubo degenerado	8	8	1.7 s
esfera menor	63	165	63.27 s
esfera maior	115	297	186.14 s
cilindro perfeito	42	98	28.45 s
cilindro degenerado	42	101	28.72 s
pontos randômicos	100	476	175.6 s
pontos randômicos	150	753	398.6 s
pontos randômicos	200	1056	678.05 s
toro limpo	182	561	
toro	182	769	567.87 s
taça limpa	174	508	
taça	174	850	545.3 s

As avaliações por nós realizadas servem apenas para mostrar que é possível a implementação de Watson no ambiente Mathematica para domínios tridimensionais. Essas avaliações foram realizadas em um Pentium 200 Mhz, 64 Mb.

Reflexões e perspectivas sobre o desempenho computacional serão dadas no próximo capítulo.

Capítulo 4

Conclusões

Os resultados apresentados neste trabalho ilustraram para vários sólidos geométricos, alguns de discretização não trivial (como o toroide), que é possível empregar o pacote Mathematica na implementação do algoritmo de Delaunay em domínios tridimensionais.

Contudo, aplicações realísticas requerem que o sistema seja capaz de tratar malhas bastante grandes em um tempo de processamento razoável. Isto significa que as funções aqui apresentadas deveriam sofrer ainda um processo posterior de otimização, para tratar esses casos.

Os resultados obtidos até agora são bastante encorajadores, principalmente se considerarmos que nenhuma das técnicas de otimização de tempo de processamento e armazenamento possíveis para o ambiente Mathematica foi usada.

Tal abordagem foi intencional: o objetivo do trabalho era conseguir uma implementação correta do algoritmo com os recursos mais comuns à disposição de um usuário padrão, e não uma versão otimizada. Contudo, essa versão pode ser agora programada muito mais facilmente, uma vez que existe um conjunto completo de funções cuja correção foi amplamente verificada.

Várias possibilidades de aperfeiçoamento do pacote, aqui apresen-

tadas, são potencialmente exploráveis.

Para listar as mais importantes:

- Pseudo-compilação de todas as funções em uma versão mais recente do pacote Mathematica [20];
- Adoção de outros estilos de programação para as partes mais críticas do código (programação funcional pura, regras de inferência, etc);
- Adoção de rotinas externas compiladas em C para as partes mais críticas do programa [20];
- Migração para uma plataforma de Hardware de maior capacidade de memória e processamento;
- Desenvolvimento de versões para outros ambientes, como Maple e Matlab.

Esperamos que os resultados obtidos possam ser de alguma utilidade a todos aqueles que, trabalhando com simulação computacional em qualquer campo de atuação, tenham eventualmente que utilizar malhas de elementos tetraédricos.

Apêndice A

Desempenho computacional dos métodos da seção 1.6.

Os testes são feitos com um cubo discretizado em seis bons tetraedros (isto é, tetraedros com a proporção > 0.01). Os pontos são adicionados gradualmente na malha.

Comparando os gráficos a seguir (pg.65), observa-se que para os métodos 1, 3 e 4 a porcentagem de *slivers* aumenta com o refinamento da malha. Para o método 2, a porcentagem de *slivers* decresce com o refinamento da malha. Os métodos 1 e 4 geram aproximadamente números iguais de tetraedros e *slivers* e por isso o desempenho de ambos é quase o mesmo, mas segundo [23] o método 1 é considerado um pouco melhor do que o método 4. Já o desempenho do método 3 é melhor com um menor número de pontos (< 50), mas à medida que os pontos são acrescentados, a porcentagem de *slivers* decresce. De todos os métodos, o método 3 gera o menor número de tetraedros. Por isso ele é o método mais rápido dos quatro.

A implementação de Watson (método 2), envolve a possível supressão de todos os tetraedros vizinhos do ponto a ser deletado. Experimentos conduzidos sobre esta implementação, mostram que o número de tetraedros gerados a cada estágio é $c(n)n$, onde n é o número de pontos na malha no estágio particular e $c(n) = c \cdot \log_2(n)$. Portanto o número total de buscas necessárias é:

$$\sum_{i=1}^{n} \{c \cdot i \cdot \log_2(i)\} = \bigcirc \left(n^2 \cdot (\log_2(n))\right)$$

Este é o desempenho da implementação de Watson no pior caso.

Outras implementações envolvem a possível supressão dos vizinhos dos tetraedros que contém o ponto a ser adicionado. Encontrar os vizinhos envolve uma busca em todos os tetraedros existentes em *TN[ ]* para localizar as faces comuns. Portanto o processo inteiro envolve cinco interações no vetor *TN[ ]*. Quando um ponto é adicionado na malha existente, no máximo 16 tetraedros serão adicionados à malha. Portanto o número total de buscas é menor do que:

$$5 \times 16 \times (1 + 2 + 3 \cdots n - 1)$$

$$5 \times 16 \times n\,(n-1)\,/2$$

$$40 \times n\,(n-1)$$

$$\bigcirc\,(n^2)$$

onde n é o número de pontos a ser inserido.

Embora a dependência temporal é um pouco mais elevada do que os outros métodos, o método 2 gera uma melhor qualidade de malha que os outros métodos. Os gráficos da pg.65 nos dão, de forma resumida, as conclusões apresentadas anteriormente. Esses gráficos podem ser encontrados na íntegra em [23]

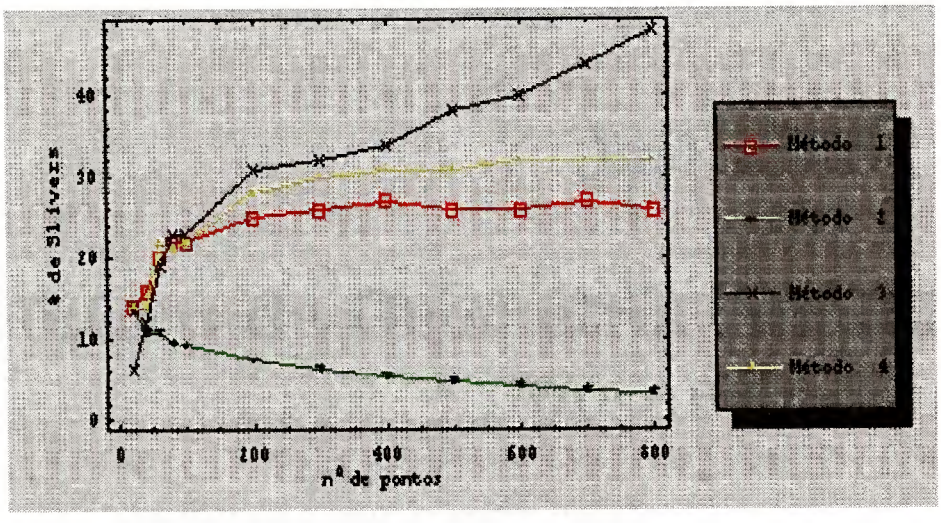

Figura A.1: % de slivers

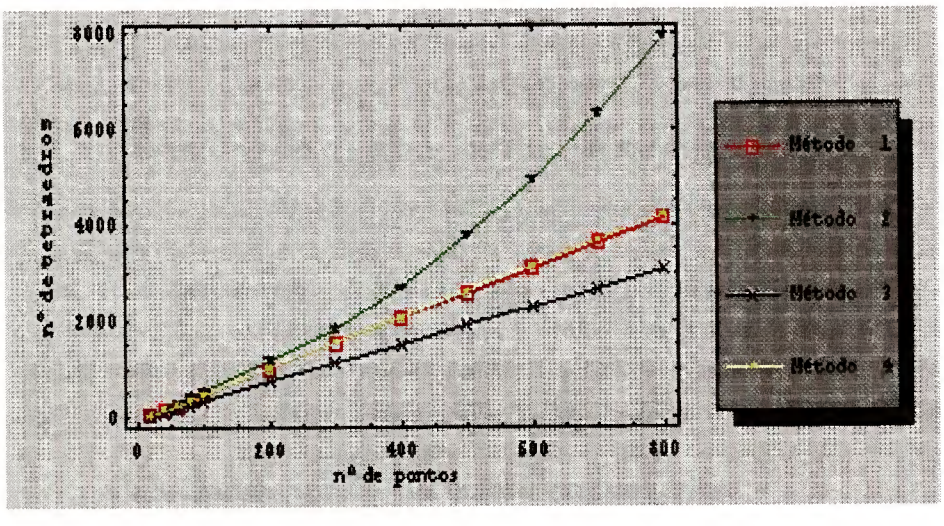

Figura A.2: Número de tetraedros

Apêndice B

Tópicos de Geometria Analítica no espaço

Calculando o raio da esfera inscrita

Seja H_1, H_2, H_3 e H_4 os comprimentos das normais, tomados a partir dos vértices dos tetraedros e apontando para as faces opostas. Se (s_1, s_2, s_3, s_4) são as coordenadas homogêneas do centro da esfera inscrita então:

$$s_1 + s_2 + s_3 + s_4 = 1$$

Por isso, o raio da esfera inscrita r é dado por:

$$r/H_1 + r/H_2 + r/H_3 + r/H_4 = 1,$$

$$(s_1 = r/H_1, s_2 = r/H_2, s_3 = r/H_3, s_4 = r/H_4)$$

que nos dá:

$$r = 1/\left(\frac{1}{H_1} + \frac{1}{H2} + \frac{1}{H_3} + \frac{1}{H_4}\right).$$

Calculando o raio da esfera circunscrita

Se (x_1, y_1, z_1) ..., etc são as coordenadas dos vértices dos tetraedros, então calculando as seguintes equações lineares:

$$x_1^2 + y_1^2 + z_1^2 + f \times x_1 + g \times y_1 + h \times z_1 + c = 0,$$

.....

etc.

podemos obter as quantidades f, g, h e c. Logo, o raio da esfera circunscrita R é dado por:

$$R^2 = \left(f^2 + g^2 + h^2\right)/4 - c.$$

Apêndice C

Espaços multiplamente conexos

Neste apêndice consideraremos X um espaço topológico e x_0, x_1 dois pontos de X.

Definição 4.0.1 Um caminho em X com ponto inicial x_0 e ponto final x_1 é uma *função contínua* α do intervalo fechado $I = [0,1]$ em X tal que $\alpha(0) = x_0$ e $\alpha(1) = x_1$. Se $x_0 = x_1$ dizemos que α é um *caminho fechado* em x_0. Geometricamente dois caminhos α, β com $\alpha(0) = \beta(0) = x_0$ e $\alpha(1) = \beta(1) = x_1$, são *homotópicos* se um deles pode ser deformado continuamente no outro.

Definição 4.0.2 Dois caminhos fechados α, $\beta : I \to X$ são *homotópicos* se existe uma função contínua $H : I \times I \to X$ tal que

$$H(t,0) = \alpha(t); H(t,1) = \beta(t), \forall t \in I$$
$$H(s,0) = H(s,1), \forall s \in I$$

Definição 4.0.3 Um espaço métrico (M,d) se diz *desconexo* quando existem dois conjuntos abertos G e H, ambos não vazios, de maneira que $G \cap H = \emptyset$ e $G \cup H = M$. Neste caso dizemos que o par de abertos G e H forma uma desconexão de M e indicamos tal fato por $M = G/H$. Um espaço *conexo* é um espaço que não é desconexo. Portanto, dizer que M é *conexo* significa dizer que não existe nenhuma desconexão de M [26].

Definição 4.0.4 Um espaço topológico X é *simplesmente conexo* se for conexo e todo caminho fechado em X é homotópico a um caminho constante. Um espaço conexo que não é simplesmente conexo é usualmente chamado de *multiplamente conexo*. Uma superfície esférica é *simplesmente conexa* e o toro é *multiplamente conexo*.

Um estudo mais detalhado dessas definições pode ser encontrado em [25].

Referências Bibliográficas

[1] C. A. R. MARRETTO & J. M. MACHADO, A compact library for automatic generation ot tetrahedra meshes using the Mathematica system, *Proc. of* 8^{th} *International IGTE Symposium on Numerical Field Calculation in Electrical Engineering & European TEAM Workshop*, Graz, Austria, Part I, (1998), 116-120.

[2] J. M. MACHADO, LMAG-3D: Um software CAD/CAE para eletromagnetismo baseado no método dos elementos finitos em 3D, Tese apresentada a Escola Politécnica da USP para obtenção do título de Doutor em Engenharia, São Paulo, (1993).

[3] F. HERMELINE, *Triangulation automatique d 'un polyèdre en dimension N*, R.A.I.R.O., Analyse numérique, Numerical Analysis, Vol. 16, n°3, pp. 211-242, (1982).

[4] P. L. GEORGE AND F. HERMELINE, Delaunay's mesh of a convex polyhedron in dimension d. Application to arbitrary polyhedra, *Int. J. for Num. Meth. in Engng.*, Vol. 33, pp. 975-995, (1992).

[5] Y. BABUSKA ET A. K. AZIZ, On the angle condition in the finite element method, Siam J. Num. Anal., Vol. 13, n°2, (1976).

[6] P. G. CIARLET, The finite element method for elliptic problems. North-Holland (1978).

[7] W. C. THACKER, A brief review of techniques for generating irregular computational grids, *Int. J. Num. Met. Engng.*, Vol. 15, pp. 1335-1341, (1980).

[8] H. S. M. COXETER, L. FEW ET C. A. ROGERS, *Covering space with equal spheres*, Mathematika 6, pp. 147-157, (1959).

[9] F. HERMELINE, Une méthode automatique de maillage en dimension n, *Thèse Paris* (1980).

[10] B. DELAUNAY, Sur la sphère vide, Bul. Acad. Sci. URSS Class. Sci. Nat., pp. 793-800, (1934).

[11] C. S. PESKIN, A Lagrangian method for the Navier-Stokes equations, Communication non publiée.

[12] M. BERGER, Géométrie tome 3: convexes et polytopes, polyèdres réguliers, aires et volumes, Fernand Nathan Paris (1978).

[13] J. C. CAVENDISH, Automatic triangulation of arbitrary planar domains for the finite element method, *Int. J. Num. Meth. Engng.*, 8, pp. 679-696, (1974).

[14] W. J. SCHROEDER AND M. S. SHEPHARD, Geometry-based fully automatic mesh generation and the Delaunay triangulation, *Int. J. Numer. Methods Eng.*, Vol. 26, pp. 2503-2515, (1988).

[15] J. C. CAVENDISH, D. A. FIED, AND W. H. FREY, An approach to automatic three-dimensional finite element generation, *Int. J. Numer. Methods Eng.*, Vol. 21, pp. 329-347, (1985).

[16] D. N. SHENTON and Z. T. Cendes, Three-dimensional finite element mesh generation using Delaunay tessellation, *IEEE Trans. Magn.*, Vol. MAG-21, n°6, pp. 2535-2538, (1985).

[17] D. F. WATSON, Computing the n-dimensional Delaunay tesselation with application to Voronoi polytopes, *Comput. J.*, Vol. 24, n°2, (1981).

[18] S. WOLFRAM, *Mathematica: A system for doing mathematics by computer*, Addison-Wesley Publishing Company, Inc., second edition, (1991).

[19] T. W. JONES, *Mathematica graphics techniques & applications*, TELOS, Inc., (1994).

[20] T. B. BAHDER, Mathematica for Scientists and Engineers, Wesley Publishing Company, *Inc*, (1995).

[21] M. MÄNTYLÄ, *An introduction solid modeling*, Computer Sciense Press, Inc, (1988).

[22] W. J. SCHROEDER AND M. S. SHEPHARD, A combined octree Delaunay method for fully automatic 3-D mesh generation, *International journal for numerical methods in engineering*, Vol. 29, pp. 37-55, (1990).

[23] S. KANAGANATHAN AND N. B. GOLDSTEIN, Comparison of four-point adding algorithms for Delaunay-type three-dimensional mesh generators, *IEEE Transactions on magnetics*, Vol. 27, n°3, May (1991).

[24] A. BOWYER, Computing Dirichlet tessellations, *The computer journal*, Vol. 24, n°2, pp. 162-166, (1981).

[25] E. L. C. FANTI E M. G. C. ANDRADE, Grupo fundamental- Uma visão geométrica, *Notas de seminários*, n^0 9, Departamento de matemática, IBILCE, UNESP, (1996).

[26] H. H. DOMINGUES, *Espaços Métricos e Introdução à Topologia*, Atual, São Paulo, (1982).