SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito: 28.08.2001

Assinatura: _____

β -Conexão: uma família de objetos
tridimensionais reconstruídos a partir de seções
planares

Alex Jesús Cuadros Vargas

Orientador: *Prof. Dr. Luis Gustavo Nonato*

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.

USP – São Carlos
Agosto/2001

A Comissão Julgadora:

Prof. Dr. Luis Gustavo Nonato

Prof. Dr. Antonio Castelo Filho

Prof. Dr. Geovan Tavares dos Santos

*Ao Senhor Deus
por este trabalho e por tudo o que ele me deu.*

*Aos meus pais Ernesto e Yolanda, e aos meus irmãos Tito e Chepy,
por todo o carinho e apoio que deles nunca faltou para mim.*

*Aos meus avôs de longe, papai Lino e mamãe Zoila,
papai Prude e mamai Chenchita,
vidas exemplares para mim.*

Agradecimentos

- *Ao meu orientador Gustavo, por toda a sua ajuda e apoio em todo momento, muito obrigado.*
- *Á professora Rosane por ter me aceitado no ICMC, ao professor Castelo, ao professor João, à professora Cristina. A todos eles, obrigado, por estar sempre dispostos a ajudar.*
- *Á Govy, Milu, Pastor, Raul, Jorge, Yorch, Mantequilla, Tio, Fresita, Miau, Guef, Juan, Richard, Titi, Waldo e Paty, obrigado pela sua amizade e todos os momentos felizes.*
- *Ao pessoal da Computação Gráfica: Tedy, Billy, Luis Fernando, Rogerio, Igor, Veridiana, Valerio, Helton, Bruno, Danielly e Marcelo, muito obrigado.*
- *Ao pessoal da Matemática: Valdemir, Fernando, Marcelo, Ricardo, Fabricio, Luciane, muito obrigado.*
- *E finalmente, também agradeço à CNPq pelo apoio financeiro da bolsa.*

Índice

Capítulo 1 Introdução.....	1
Capítulo 2 O problema da Reconstrução Tridimensional	4
2.1 Processo de obtenção de contornos.....	4
2.2 Reconstrução tridimensional.....	6
2.3 Formalizando o problema	7
2.4 Trabalhos prévios	8
Capítulo 3 Diagrama de Voronoi e Triangulação de Delaunay.....	13
3.1 Diagrama de Voronoi.....	13
3.2 Triangulação de Delaunay.....	14
3.3 Algoritmos para construção da Triangulação de Delaunay	15
3.3.1 Algoritmo Incremental.....	15
3.3.2 Outros algoritmos e comparação de tempos de execução.....	17
3.3.3 Primitivas geométricas para a Triangulação de Delaunay	19
3.3.4 Algoritmo de Boissonnat	22
3.3.5 Algoritmo <i>Quickhull</i>	25
3.3.5.1 Fecho convexo	25
3.3.5.2 O algoritmo <i>Quickhull</i>	27
Capítulo 4 Pipeline de Reconstrução tridimensional.....	29
4.1 Visão geral do pipeline	29
4.2 Pré-Processo de contornos	31
4.2.1 Leitura de contornos.....	32
4.2.2 Orientação de contornos.....	32
4.2.3 Verificação de arestas Delaunay	33
4.3 Cálculo de deslocamento	34
4.4 Reconstrução tridimensional - β -Conexão.....	34
4.4.1 Triangulação.....	36
4.4.2 Classificação de arestas.....	37
4.4.3 Classificação de tetraedros.....	39
4.4.4 Cálculo de componentes conexas	41
4.4.5 Desconexão de componentes conexas	45

4.4.6	Subdivisão de tetraedros	46
4.5	União de Sub-Sólidos.....	50
Capítulo 5	Resultados e Conclusões	51
5.1	Resultados	51
5.2	Contribuições	55
5.2.1	Utilização do <i>pipeline</i> de reconstrução	55
5.2.2	Ferramenta de visualização de dados.....	56
5.3	Conclusões e trabalhos futuros	57
Referências Bibliográficas.....		58
Apêndice A	Estrutura Half-Face	62
As classes da estrutura Half-Face		62
A classe sólido		63
A classe célula.....		64
A classe vértice		64
A classe face de bordo		64
A classe semi-face.....		65
A classe semi-aresta.		65
A classe Estrela do vértice		66
A classe vértices singulares.....		66
A classe arestas singulares		67
Armazenamento de informações.....		68
Representação de singularidades		68
Apêndice B	Definições básicas de Topologia Algébrica	69

Lista de Figuras

Figura 2.1 – Processo de obtenção de contornos.	5
Figura 2.2 – Problema da correspondência.	6
Figura 2.3 – Conjuntos de contornos com topologias iguais.	7
Figura 3.1 – Diagrama de Voronoi e triangulação de Delaunay.	14
Figura 3.2 – Inserção fora da triangulação.	16
Figura 3.3 – Inserção dentro da triangulação.	17
Figura 3.4 – Triângulo abc localmente Delaunay.	17
Figura 3.5 – Operação <i>flipping</i>	18
Figura 3.6 – Desempenho em segundos de três algoritmos que constróem a triangulação de Delaunay.	19
Figura 3.7 – Teste de visibilidade de uma aresta a partir de um ponto.	20
Figura 3.8 – Teste para ver se um ponto esta fora ou dentro da circunsfera do triângulo.	21
Figura 3.9 – Triangulação de Delaunay e diagrama de Voronoi de pontos em planos paralelos.	22
Figura 3.10 – Projeção do Diagrama de Voronoi de P1 em P2.	23
Figura 3.11 – Tetraedros associados aos pontos dos diagramas de Voronoi.	23
Figura 3.12 – Tetraedros associados a um vértice que surgiu por uma interseção.	24
Figura 3.13 – Triangulação final produzida pelo algoritmo de Boissonnat.	24
Figura 3.14 – Exemplos de fechos convexos.	25
Figura 3.15 – Cálculo da triangulação de Delaunay a partir do fecho convexo.	26
Figura 3.16 – Inserção de um novo ponto com o algoritmo incremental para fechos convexos.	27
Figura 3.17 – Desempenho das estratégias de inserção: seleção aleatória do algoritmo incremental e seleção do ponto mais distante do algoritmo <i>Quickhull</i>	28
Figura 4.1 – Dados de entrada e saída do <i>pipeline</i> de reconstrução.	30
Figura 4.2 – Sub-processos principais do <i>pipeline</i> de reconstrução.	31
Figura 4.3 – Pré-Processo de contornos.	31
Figura 4.4 – Sub-processos principais do <i>pipeline</i> de reconstrução.	31
Figura 4.5 – Conjunto de contornos em um nível de entrada.	32
Figura 4.6 – Conjunto de contornos coerentes.	33
Figura 4.7 – Arestas Delaunay.	34

Figura 4.8 – Esquema de entradas e saídas do algoritmo β -Conexão.....	35
Figura 4.9 – Etapas do algoritmo de reconstrução β -Conexão.	36
Figura 4.10 – Triangulação de Delaunay 3D.	37
Figura 4.11 – Classificação de arestas.	37
Figura 4.12 – Esqueletos internos e externos diagrama de Voronoi.....	38
Figura 4.13 – Classificação de tetraedros.	40
Figura 4.14 – Origem dos tetraedros reversos.	41
Figura 4.15 – Grafo lógico associado a uma triangulação de Delaunay 3D com pontos distribuídos em seções planares.	42
Figura 4.18 – Exemplo de desconexão de componentes com singularidades	45
Figura 4.19 – Caso especial com $\beta = 0$, e tetraedros reversos.	46
Figura 4.20 – Aparição de singularidades e sua eliminação com subdivisão de tetraedros. ...	47
(a) Exemplo de tetraedro reverso externo,	47
(b) Eliminação do tetraedro causando uma singularidade em aresta, (c) Subdivisão de tetraedro, (d) Traslado dos novos pontos da subdivisão.	47
Figura 4.21 – Subdivisão de tetraedros do tipo 1.....	48
Figura 4.22 – Subdivisão de tetraedros do tipo 2.....	48
Figura 4.23 – Exemplos de sub-sólidos.	49
Figura 4.24 – Exemplos de sólidos finais.	50
Figura 5.1 – Diferentes objetos a partir de um mesmo conjunto de seções planares.....	51
Figura 5.2 – Fato interessante acontecido durante o cálculo de componentes conexas.....	52
Figura 5.3 – Comportamento do algoritmo β -Conexão com o mesmo conjunto de seções planares e diferentes valores para o parâmetro β	53
Figura 5.4 – Modelos finais gerados a partir de um mesmo conjunto de seções planares paralelas e diferentes valores de β entre seções planares.....	54
Figura 5.5 – Visualizações em tempo de depuração. (a) Contornos, (b) Arestas classificadas, (c) Triangulação classificada, (d) Tetraedros encolhidos, (e) Wire frame, (f) Grafo associado, (g) Componentes conexas e (h) Singularidades.	56
Figura A.1 – Visão geral da estrutura Half-Face	63
Figura A.2 – Relações da classe Sólido.	63
Figura A.3 – Relações da classe Célula.	64
Figura A.4 – Relações do nó Vértice.	64
Figura A.5 – Relações do nó Faces de Bordo.	64

Figura A.6 – Relações da classe Semi-Face.	65
Figura A.7 – Relações da classe Semi-Aresta.....	66
Figura A.8 – Relações da classe Estrela do Vértice.....	66
Figura A.9 – Relações da classe Vértice Singular.	67
Figura A.10 – Relações da classe Arestas Singulares.....	67
Figura A.11 – Exemplos de: (a) Vértice singular e (b) Aresta singular.....	68
Figura B.1 – Fechos convexos de: (a) 2 pontos, (b) 3 pontos e (c) 4 pontos.....	70
Figura B2 – Estrela de: (a) um ponto, (b) uma aresta.	71
Figura B3 – Link de: (a) um ponto, (b) uma aresta.....	72
Figura B4 – (a) 3-Variedade LP com Bordo e (b) Bordo (sem tampas) do objeto da figura a.72	
Figura B5 – Conjunto de 2-simplexos com orientação coerente.	73

Lista de Tabelas

Tabela 3.1 Desempenho em segundos de três algoritmos que constróem a triangulação de Delaunay.	19
---	----

Resumo

Neste trabalho, uma nova técnica de reconstrução tridimensional é apresentada. Esta técnica oferece uma família de modelos, governada por um parâmetro chamado β , e que pode ser obtida a partir de um mesmo conjunto de seções planares. Este fato faz possível múltiplas opções para o problema da correspondência, tarefa difícil de ser realizada por outros métodos de reconstrução. Além da flexibilidade na conexão entre regiões de níveis consecutivos, o algoritmo produz LP-variedades e respeita a condição de re-exemplificação.

Abstract

In this work a new three-dimensional reconstruction technique is presented. This technique offers a family of models, governed by a parameter denoted β , that can be obtained from a same set of cross sections. This fact makes possible multiples choices for the correspondence problem, a difficult task to be performed by other reconstruction methods. Besides the flexibility in the connection among regions of the adjacent slices, the algorithm produces PL-manifolds and respects the re-sampling condition.

Introdução

A crescente disponibilidade de recursos computacionais para o cálculo, simulação e aquisição de dados permite que cientistas e engenheiros produzam enormes conjuntos de dados bi ou tridimensionais (2D ou 3D), em geral multivariados. A aplicação de técnicas de Computação Gráfica em dados científicos com o objetivo de ganhar compreensão desses dados, testar hipóteses e esclarecer fenômenos e efeitos em geral, compreende o objeto de estudo da área denominada Visualização Científica (ViSC - *Visualization in Scientific Computing*) [CG87].

A visualização científica tem sido utilizada em muitas áreas da ciência, especialmente na área médica, onde desempenha um papel importante para diagnósticos, cirurgias e terapias. Dentre as técnicas de visualização, a reconstrução tridimensional ganhou ampla aceitação já que permite visualizar objetos tridimensionais a partir de dados bidimensionais ou seções planares paralelas (imagens 2D). Em medicina, dados bidimensionais podem ser obtidos por dispositivos de aquisição de imagens como Tomografia por Ressonância Magnética (*MRI - Magnetic Resonance Imager*), Tomografia Computarizada (*CT - Computerized Tomography*) ou Microscopia Eletrônica por Laser. Estes resultados são apresentados como uma lista de imagens 2D com resolução e clareza variáveis.

Através dos anos, muitos algoritmos têm sido desenvolvidos para reconstruir modelos tridimensionais de um objeto real a partir de um conjunto de imagens 2D. Para reconstruir um modelo 3D, estes algoritmos têm que lidar com três problemas intrínsecos à reconstrução, a saber: a correspondência (*correspondence*), a geração de malha (*tiling*) e a ramificação (*branching*).

- O problema da correspondência aparece quando as regiões contidas em dois níveis consecutivos se dividem ou se juntam. Neste problema, é preciso decidir, entre as várias alternativas de conexão, a mais apropriada.
- Geração de malha refere-se ao aspecto da triangulação na faixa entre dois níveis consecutivos.

- O problema de ramificação está relacionado com a identificação de pontos de sela (singularidades) que aparecem quando o número de contornos nas seções adjacentes é diferente.

Embora os algoritmos de reconstrução descritos na literatura apresentem estratégias diferentes para resolver os problemas de correspondência, geração de malha e ramificação, eles têm uma característica em comum: a inflexibilidade na escolha da correspondência. A maioria das técnicas de reconstrução supõe a correspondência previamente definida ou assume um critério de proximidade para conectar as regiões contidas em seções planares adjacentes.

A importância da flexibilidade na escolha da correspondência é evidente, já que somente a informação contida nas seções planares não é suficiente para garantir uma correta conexão.

Este trabalho apresenta uma nova técnica de reconstrução a partir de seções planares. Esta técnica permite múltiplas escolhas na definição da correspondência entre regiões contidas em seções planares adjacentes. Baseado na triangulação de Delaunay, o algoritmo oferece uma família de modelos que podem ser obtidos a partir de um mesmo conjunto de seções planares, sendo assim, uma ferramenta interessante na reconstrução de objetos com alto grau de ramificação ou união. A família de modelos é governada por um parâmetro, β , que assume valores no conjunto dos números naturais $\{0,1,2,\dots\}$. Para $\beta = 0$, não existe correspondência entre as regiões e quando β cresce a correspondência cresce também. Além da flexibilidade oferecida na correspondência, o algoritmo garante que o modelo final é uma variedade tridimensional linear por partes (*PL-manifold*), o que é desejado em simulações numéricas. Outra importante característica deste algoritmo é que ele respeita o critério de re-exemplificação (*re-sampling*). Este critério significa que a interseção do modelo com os planos originais de corte deve resultar nas regiões originais.

A idéia de usar parâmetros para criar uma família de modelos tridimensionais tem sido utilizada na reconstrução a partir de nuvens de pontos. Um exemplo clássico é o *Alpha-shapes* de Edelsbrunner *et al.* [Ede94]. Embora seja interessante a estratégia adotada pelo *Alpha-shapes*, ela não pode ser diretamente aplicada para resolver o problema da correspondência, e garantir que o modelo resultante seja uma variedade pode ser difícil.

Este trabalho está organizado como segue:

- No capítulo 2 será descrito o problema da reconstrução tridimensional a partir de seções planares. Este capítulo começa descrevendo o processo de obtenção de contornos. Depois os problemas comuns de um processo de reconstrução são revisados. Em seguida, o problema da reconstrução será formalizado e as condições que um objeto reconstruído deve satisfazer serão apresentadas. Finalmente alguns trabalhos prévios de reconstrução tridimensional serão apresentados.

- No capítulo 3 serão apresentados os conceitos fundamentais de diagrama de Voronoi e triangulação de Delaunay que serão necessários para os próximos capítulos. Neste capítulo também serão descritos três algoritmos para a triangulação de Delaunay em $\mathbb{R}^2$ com comparações entre eles: os algoritmos Incremental, *Flipping* e *Divide-and-Conquer*. Depois, será explicado o algoritmo de Boissonnat, que é um caso específico para pontos distribuídos em planos paralelos em $\mathbb{R}^3$. Finalmente o algoritmo *Quickhull* em $\mathbb{R}^n$ para fechos convexos é explicado juntamente com sua relação com a triangulação de Delaunay.

- No capítulo 4 será detalhado o *pipeline* de reconstrução tridimensional a partir de seções planares, onde é utilizado o novo algoritmo de reconstrução β -Conexão, proposto e desenvolvido neste trabalho.

- No capítulo 5 serão apresentados os resultados e conclusões.

- No apêndice A será apresentada a estrutura de dados *Half-face* que foi utilizada no desenvolvimento do algoritmo. Esta estrutura representa volumes, enfatizando a velocidade na obtenção das relações de incidências e adjacências e também os custos de armazenamento, que são fatores importantes para o processo de reconstrução.

- Finalmente, no apêndice B serão apresentadas algumas definições básicas sobre topologia algébrica que servirão como base para os conceitos apresentados.

O problema da Reconstrução Tridimensional

A aparição de dispositivos de medição não invasivos como Tomografia por Ressonância Magnética (*MRI - Magnetic Resonance Imager*), Tomografia Computarizada (*CT - Computerized Tomography*) ou Microscopia Eletrônica por Laser, tornou possível a visualização de uma sequência de seções planares de objetos tridimensionais. Este fato motivou o desenvolvimento de várias técnicas de reconstrução tridimensional. O objetivo destas técnicas é reconstruir, a partir de um conjunto de seções planares, um modelo 3D que se aproxime do original e que possa ser utilizado em aplicações como simulação de procedimentos médicos, simulações numéricas, educação, etc.

Antes do processo de reconstrução tridimensional, muitos algoritmos precisam de uma etapa de pré-processamento nas imagens originais para extrair contornos (é o caso do algoritmo proposto neste trabalho).

O capítulo começa descrevendo o processo de obtenção de contornos (seção 2.1). Depois os problemas comuns de um processo de reconstrução são revisados (seção 2.2). Em seguida o problema da reconstrução será formalizado e as condições que um objeto reconstruído deve satisfazer serão apresentadas (seção 2.3). Finalmente algumas técnicas de reconstrução tridimensional são apresentadas (seção 2.4).

2.1 Processo de obtenção de contornos

Com o objetivo de organizar informações e obter mais precisão na hora da reconstrução tridimensional, muitos algoritmos utilizam como entrada de dados um conjunto de contornos representando o bordo das regiões contidas nas seções planares. Estes contornos podem ser obtidos de imagens bidimensionais provenientes de algum aparelho de medição não invasivo.

O processo de obtenção de contornos começa intersectando um objeto 3D com um conjunto de planos paralelos $\pi_1, \dots, \pi_n$ como mostra a figura 2.1a. Quanto menor a separação entre os planos paralelos maior a precisão do modelo reconstruído. Cada uma destas interseções dá origem a uma imagem bidimensional ou seção planar como na figura 2.1b. Em cada uma

destas imagens bidimensionais, o bordo das áreas mais representativas podem ser obtidas em forma de contornos, como pode ser apreciado na figura 2.1c e d. O processo de extrair regiões de interesse a partir de imagens é chamado de segmentação. Existe uma variada e enorme quantidade de métodos para extrair contornos de imagens 2D. Muitos desses métodos não são totalmente automatizados, devido à complexidade das imagens, necessitando muitas vezes de interação humana para decidir quais as áreas importantes e fazer uma segmentação de forma correta. Por exemplo, no caso da figura 2.1c, informações dos olhos não deveriam ser misturadas com informações da região do cérebro. Fazer com que um algoritmo de segmentação decida o que é uma área representativa depende muito do tipo da imagem e não é uma tarefa trivial.

Figura 2.1 Processo de obtenção de contornos.

(a) Objeto 3D intersectado pelos planos de corte, (b) Imagem correspondente a um plano de corte, (c) Processo de Segmentação na imagem bidimensional e (d) Conjunto de contornos em um plano de corte.

A seguir, alguns problemas clássicos que acontecem em uma reconstrução tridimensional a partir de seções planares serão apresentados.

2.2 Reconstrução tridimensional

O processo de aquisição das seções planares de um objeto 3D origina uma perda de informação do objeto original. Por causa desta perda de informação os algoritmos de reconstrução tridimensional têm que decidir qual a forma do objeto entre os planos.

Existem três problemas que aparecem nesta fase de decisão: a correspondência (*correspondence*), a geração de malha (*tiling*) e a ramificação (*branching*).

- O **problema da correspondência** aparece quando, em dois níveis consecutivos, os contornos se dividem ou se juntam. Neste problema é preciso decidir, entre várias alternativas de conexão, a mais apropriada.
- O **problema de geração de malha** refere-se ao aspecto da triangulação na faixa entre dois níveis consecutivos.
- O **problema de ramificação** está relacionado à identificação de pontos de sela (singularidades) que aparecem quando o número de contornos nas seções adjacentes é diferente.

Um dos principais problemas em reconstrução 3D a partir de seções planares paralelas é a correspondência, ou seja, como encontrar quais são os contornos que devem ser conectados entre níveis consecutivos. Ainda que os contornos a serem relacionados estejam bem definidos, estas informações não são suficientes para decidir como eles devem ser conectados. Por exemplo, a figura 2.2a apresenta um conjunto de contornos onde é impossível saber, utilizando apenas as informações planares, qual modelo de conexão é o mais adequado. As figuras 2.2b, 2.2c, 2.2d apresentam algumas possíveis ligações para este conjunto de contornos. Utilizar um critério de proximidade para realizar as ligações, como é feito por muitos algoritmos de reconstrução, pode ser uma boa opção, embora definir o que é “próximo” não seja uma tarefa fácil.

Figura 2.2 – Problema da correspondência.

(a) Conjunto de regiões em planos paralelos. (b, c, d) Conexões possíveis a partir de um mesmo conjunto de regiões.

Mesmo que a topologia entre dois níveis consecutivos seja a mesma, pode ser complicado fazer as ligações como se mostra na figura 2.3.

Figura 2.3 – Conjuntos de contornos com topologias iguais.

(a) Conexão relativamente evidente, (b) Conexão de difícil realização.

2.3 Formalizando o problema

Quando um objeto 3D, denominado O , é intersectado por um conjunto de planos paralelos $\pi_1, \dots, \pi_n$, um conjunto de curvas ou contornos contidos nestes planos pode ser obtido. Devido aos problemas que existem na hora de fazer as ligações entre as seções planares, descritos anteriormente, não é possível garantir que o objeto reconstruído O' seja igual ao objeto original, ou que pelo menos possuam a mesma topologia. Para que O' seja uma aproximação razoável de O impõe-se as **hipóteses de reconstrução** que são:

1. A interseção de O' com os planos π_i , $i = 1, \dots, n$ é idêntica à interseção de O com estes planos.
2. Cada objeto reconstruído é uma 3-Variedade LP orientável com bordo.

A primeira hipótese de reconstrução é o mínimo que se deve esperar da aproximação obtida do objeto O . A segunda hipótese de reconstrução procura tornar o modelo gerado mais próximo do mundo real. Outro motivo para se reconstruir objetos que são variedades LP é que as simulações numéricas, as operações booleanas, etc., utilizam relações de vizinhança em variedades.

O esquema de reconstrução proposto no capítulo 4 garante que as duas hipóteses de reconstrução são satisfeitas entre cada um dos níveis consecutivos, estendendo-se diretamente para o objeto O' .

2.4 Trabalhos prévios

Desde a aparição dos dispositivos de medição não invasivos (*MRI, CT, etc*) capazes de representar um objeto real a partir de uma sequência de imagens, muitas técnicas de reconstrução tridimensional têm sido desenvolvidas. Algumas destas técnicas constroem uma superfície que representa o bordo do objeto e outras geram modelos com representações volumétricas. Entre as principais técnicas de reconstrução podemos citar: as técnicas implícitas, os modelos baseados em *voxels*, os métodos de otimização, os modelos deformáveis, os modelos baseados em nuvens de pontos e as técnicas heurísticas.

Embora os algoritmos de reconstrução descritos na literatura apresentem estratégias diferentes para resolver os problemas de correspondência, geração de malha e ramificação, todos eles têm uma característica em comum, a inflexibilidade na escolha da correspondência. A maioria dos algoritmos de reconstrução supõe a correspondência definida previamente ou assume um critério de proximidade para conectar as regiões contidas em seções planares adjacentes.

A seguir algumas destas estratégias de reconstrução são analisadas dando ênfase na forma em que elas resolvem o problema da correspondência.

Para lidar com o problema de criação de modelos 3D a partir de dados 2D, as técnicas implícitas geram uma função implícita suave de forma que o bordo do objeto que se quer representar está no conjunto zero desta função. No trabalho feito por Jones *et al.* [Jon94], uma função é definida a partir de uma distância com sinal para cada um dos contornos. Esta função, juntamente com a técnica *Marching Cubes* [Lor87], é utilizada para extrair a superfície do modelo. Utilizando técnicas implícitas é possível garantir que a função criada contenha os contornos originais no seu conjunto de valores zero, mas é difícil garantir que os contornos nas superfícies geradas sejam os mesmos que os contornos originais. A correspondência é automaticamente definida pela função implícita, fazendo difícil uma outra opção de conexões.

Os modelos baseados em *voxels* [Kau98], [Soa00] visualizam o volume do objeto diretamente a partir das propriedades das unidades do volume de dados, chamados *voxels* (os correspondentes 3D dos pixels 2D). Esta visualização é feita sem construir uma superfície ou malha volumétrica. Embora este tipo de estratégia evite problemas como segmentação, correspondência, geração de malha e ramificação, é praticamente impossível especificar a correspondência entre as regiões, porque as interpolações usadas para gerar a imagem estipulam as conexões. Existem também dificuldades na interação com o modelo gerado,

dificultando sua utilização em algumas aplicações. No entanto, este tipo de modelo pode ser uma alternativa interessante para a visualização de informações volumétricas.

Tentativas de reconstrução mediante métodos de otimização utilizam usualmente teoria de grafos. Um dos primeiros trabalhos em reconstrução a partir de seções planares foi desenvolvido por Keppel [Kep75]. Neste trabalho a superfície que maximiza o volume do objeto que esta sendo reconstruído é obtida a partir de uma busca ótima em um grafo. Shinagawa *et al.* [Shi91] expandem um grafo toroidal discreto para formar um grafo contínuo, e uma homotopia é usada para reconstruir superfícies paramétricas e definir as correspondências. Meyers *et al.* [Mey92] aproximam os contornos dentro de cilindros e uma árvore geradora mínima determina a correspondência entre eles. Os métodos de otimização podem resolver o problema de geração de malha, mas alguma heurística é necessária para encontrar uma solução ao problema de ramificação.

Os modelos deformáveis combinam geometria, física e teoria da aproximação para gerar o modelo. A partir de um modelo inicial, deformações são executadas para aproximar o objeto original. Embora conexões diferentes possam ser obtidas usando distintos modelos iniciais, realizar tais inicializações não é uma tarefa trivial. Uma boa revisão de modelos deformáveis pode ser encontrada no trabalho feito por Singh *et al.* [Sin98]. A vantagem de utilizar modelos deformáveis é que a segmentação pode ser incorporada ao processo de reconstrução, embora objetos com topologias complexas sejam difíceis de serem tratados. Delingette [Del94], utilizando *simplex meshes*, apresenta uma alternativa para o problema da topologia.

Existem também técnicas que reconstróem o objeto original a partir de uma nuvem de pontos. A maioria destas técnicas assumem que a amostra de pontos de entrada satisfaz alguns critérios de densidade. Sendo assim, é possível garantir que a saída seja topologicamente correta e convergente à superfície original.. Nina Amenta *et al.* [Ame97], baseados no diagrama de Voronoi, e Dey *et al.* [Dey00], baseados na triangulação de Delaunay, criaram técnicas deste tipo.

Existe uma quantidade enorme de algoritmos que fazem uso de heurísticas para decidir a respeito das conexões entre contornos em planos consecutivos. Ekoule *et al.* [Eko91] mapeiam contornos planares dentro de seus fechos convexos e um critério baseado na aresta mais curta é usado para conectar os fechos convexos que estão em níveis adjacentes. O problema de ramificação é resolvido criando um contorno intermediário entre dois níveis consecutivos. Bajaj *et al.* [Baj96] definem alguns critérios que a superfície reconstruída deve satisfazer, e, a partir destes critérios, um algoritmo é construído. Nas regiões onde as condições não podem ser satisfeitas, a superfície é completada através de uma triangulação, levando em consideração o esqueleto de Voronoi. A correspondência entre regiões é implicitamente definida pelo critério de reconstrução, e um modelo volumétrico pode ser gerado usando um processo de tetraedralização [Baj99].

Boissonnat [Boi88] utiliza um grafo obtido a partir da projeção ortogonal dos diagramas de Voronoi 2D para gerar a triangulação de Delaunay 3D. Depois, utilizando este grafo e eliminação de tetraedros para cada dois níveis consecutivos, Boissonnat gera o modelo volumétrico do objeto. A correspondência é dada neste caso pela triangulação de Delaunay juntamente com a eliminação de tetraedros. Geiger [Gei93] utiliza a idéia de Boissonnat projetando os esqueletos de Voronoi bidimensionais em níveis consecutivos para manipular ramificações complexas e contornos desiguais. Nesta alternativa, não é tratado o problema de singularidades geradas no modelo durante a eliminação de tetraedros. Geiger criou também um pacote de reconstrução 3D de domínio público chamado NUAGES [Gei93b].

Nonato [Non98],[Non01], motivado pelas idéias de Boissonnat e de Geiger, utiliza a triangulação de Delaunay tridimensional para construir um modelo volumétrico do objeto. Para estabelecer a correspondência entre contornos, Nonato baseia-se num critério de proximidade que pode ser obtido verificando a existência de um tipo especial de tetraedro na triangulação de Delaunay, que ele chama de tetraedro reverso. Ele considera que quando este tipo de tetraedro aparece na triangulação de Delaunay gerada a partir dos pontos dos contornos, estes contornos estão geometricamente bem posicionados, e então são conectados. Este algoritmo pode também aceitar outras conexões se previamente definidas. Uma importante característica deste algoritmo é que o modelo gerado, além de tratar os problemas da reconstrução de forma bastante satisfatória, satisfaz também os critérios de re-amostragem e 3-variedade LP.

- O critério de re-amostragem exige que a interseção do modelo gerado com os planos de corte resulte nos contornos originais.
- O critério de 3-variedade LP impõe que o modelo gerado seja sempre uma 3-variedade linear por partes.

Uma comparação entre o algoritmo de Nonato e o algoritmo utilizado em NUAGES por Geiger pode ser encontrada em [Non01]. Uma outra comparação entre o algoritmo de Nonato e uma abordagem que utiliza técnicas implícitas para gerar superfícies pode ser encontrada em [Non00b].

Dentre as técnicas de reconstrução baseadas em contornos, as que utilizam geometria computacional têm apresentado resultados robustos e eficientes. Uma das vantagens em se utilizar geometria computacional para construir os modelos, mais especificamente a triangulação de Delaunay, é que existem algoritmos eficientes para criar tais estruturas. Outra vantagem está no fato do modelo criado ser volumétrico, o que possibilita reconstruir as regiões internas dos objetos de maneira bastante eficiente. Outro motivo que justifica a utilização de geometria computacional em reconstrução 3D é a possibilidade de interação com o modelo.

Neste trabalho será apresentada o β -Conexão, uma nova técnica de reconstrução tridimensional baseada na triangulação de Delaunay e nas idéias de Nonato[Non98]. Desta forma, a estratégia de reconstrução proposta é volumétrica e baseada em contornos. Esta nova técnica oferece uma família de modelos que podem ser obtidos a partir de um mesmo conjunto de seções planares, tornando possível múltiplas opções para resolver o problema de correspondência, fato que é difícil ser obtido com outras técnicas de reconstrução. Para resolver o problema da correspondência, o algoritmo β -Conexão calcula o caminho mais curto em tetraedros entre cada dois contornos contidos nos níveis consecutivos. Depois, utiliza um valor inteiro positivo, chamado β -Parâmetro (estabelecido pelo usuário), que expressa uma distância mínima em tetraedros como condição para que os contornos sejam conectados. Isto é, para um β -Parâmetro = n , todos os contornos que estejam a uma distância em tetraedros menor ou igual a n ficarão conectados. Desta forma, a partir de um conjunto de contornos em dois níveis consecutivos, é possível gerar uma família de objetos reconstruídos dando diferentes valores ao β -Parâmetro. Este algoritmo, da mesma forma que o algoritmo de Nonato, satisfaz os critérios re-amostragem e 3-variedade LP.

No capítulo seguinte serão apresentados os conceitos fundamentais de diagrama de Voronoi e triangulação de Delaunay que serão necessários para os próximos capítulos. Serão descritos também alguns algoritmos que constroem a triangulação de Delaunay bidimensional e tridimensional fazendo comparações entre eles.

Diagrama de Voronoi e Triangulação de Delaunay

Neste capítulo serão apresentados os conceitos fundamentais de diagrama de Voronoi e triangulação de Delaunay que serão necessários no decorrer deste trabalho.

O capítulo começa com a definição de diagrama de Voronoi e triangulação de Delaunay com suas propriedades e relações. Três algoritmos para a triangulação de Delaunay serão apresentados: O algoritmo Incremental em R^2 será explicado em detalhe junto com uma breve descrição dos algoritmos *Flipping* e *Divide-and-Conquer* em R^2 . Uma comparação entre estes três algoritmos será também realizada. Em seguida será explicado o algoritmo de Boissonnat para pontos distribuídos em planos paralelos em R^3 . E finalmente o algoritmo *Quickhull* será detalhado junto com a definição de fecho convexo e uma explicação de como obter uma triangulação de Delaunay a partir de um fecho convexo.

3.1 Diagrama de Voronoi

Um conjunto finito não vazio $A = \{x_1, \dots, x_z\}$ em R^m está em **posição geral** se nenhum subespaço afim de R^m contém A , e se não há uma esfera A^{m-1} passando através de um subconjunto de A com $m+k$, $k>1$ pontos. No texto a seguir sempre vamos considerar conjuntos de pontos em posição geral.

O **Diagrama de Voronoi** V de A é uma decomposição de R^m em células m -dimensionais $\{V_1, \dots, V_n\}$ com as seguintes propriedades (Figura 3.1a):

- Cada V_i contém apenas um ponto $x_i \in A$
- Dado $x \in R^m$, $x \in V_i$ se e somente se $d(x, x_i) \leq d(x, x_j)$ para todo $i \neq j$, onde d é a distância euclidiana.

Cada V_i será chamado **poliedro de Voronoi** associado a x_i . Note que $R^m = \cup V_i$ e que cada poliedro de Voronoi é definido pela interseção de $n-1$ semi-espacos fechados. Como semi-espacos são convexos e a interseção de conjuntos convexos é convexa tem-se que cada poliedro de Voronoi é convexo.

Destas propriedades pode-se mostrar que a interseção de k poliedros do Diagrama de Voronoi, onde $2 \leq k \leq m+1$, é vazia ou é uma célula $(m-k+1)$ -dimensional contida no diagrama.

3.2 Triangulação de Delaunay

A **Triangulação de Delaunay DT** (Figura 3.1b) é o dual geométrico do Diagrama de Voronoi (VD), isto é, para cada célula p -dimensional do diagrama pode-se associar de forma natural um $(m-p)$ -simplexo da triangulação.

Por exemplo, no caso de 2D (Figura 3.1c) :

- Um triângulo (2-simplexo) de DT está associado a um vértice do Diagrama de Voronoi,
- Uma aresta (1-simplexo) de DT está associada a uma aresta do Diagrama de Voronoi,
- Um vértice (0-simplexo) de DT está associado a uma célula de dimensão 2 do Diagrama de Voronoi.

Figura 3.1 – Diagrama de Voronoi e triangulação de Delaunay.

(a) Diagrama de Voronoi, (b) Triangulação de Delaunay, (c) Triangulação de Delaunay como dual geométrico do diagrama de Voronoi e (d) Conseqüência da dualidade.

Uma conseqüência da dualidade e da posição geral dos pontos é que cada célula 0-dimensional do Diagrama de Voronoi é o centro da esfera que circunscreve um m -simplexo da Triangulação de Delaunay. Uma importante propriedade que caracteriza a

triangulação de Delaunay é que a esfera que circunscreve um m -simplexo não pode conter qualquer ponto de A em seu interior (Figura 3.1d).

3.3 Algoritmos para construção da Triangulação de Delaunay

Esta seção descreve alguns algoritmos para obter a triangulação de Delaunay. Embora não seja fácil implementar uma triangulação de Delaunay de forma robusta, vários algoritmos que utilizam técnicas distintas se dedicam a este propósito, sendo que a maioria deles constrói a triangulação de Delaunay assumindo os pontos em posição geral. Por várias razões, é mais conveniente expressar algoritmos em termos de triangulação de Delaunay que de diagrama de Voronoi. Uma vez obtida a triangulação de Delaunay, o diagrama de Voronoi pode ser construído em tempo linear [For94](pg.209).

Nesta seção será feita uma descrição detalhada do algoritmo *Incremental* em $\mathbb{R}^2$, que é considerado relativamente fácil de implementar, intuitivo e de fácil generalização para dimensões maiores. Depois, serão apresentados brevemente dois algoritmos em $\mathbb{R}^2$ chamados *Flipping* e *Divide-and-Conquer* e será feita uma comparação com o algoritmo *Incremental*. Em seguida, duas primitivas geométricas para a construção da triangulação de Delaunay serão detalhadas.

O algoritmo de Boissonnat [Boi88], que constrói a triangulação de Delaunay em $\mathbb{R}^3$ para pontos distribuídos em planos paralelos, será também apresentado. Finalmente o algoritmo Quick Hull será explicado.

3.3.1 Algoritmo Incremental

O algoritmo incremental é um dos mais populares. Isto é devido a sua conduta local e relativa simplicidade para ser estendido a $\mathbb{R}^m$. Em $\mathbb{R}^m$ este algoritmo constrói a triangulação de Delaunay de n pontos em posição geral em tempo $O(n^{\lceil (m+1)/2 \rceil})$, $m \geq 2$. Em [Zha96] e [Lu97] podem ser encontrados programas exemplo em *java* deste algoritmo.

O algoritmo incremental consiste em:

Considere $A = \{x_1, \dots, x_n\}$ em $\mathbb{R}^2$ um conjunto de pontos em posição geral.

- 1- Inicia-se a Triangulação de Delaunay DT com os 3 primeiros pontos de A que formam um triângulo, este triângulo possui circunsfera vazia em relação a ele mesmo.

2- Para cada x_i , $i = 4, \dots, n$, determina-se se x_i está fora ou dentro da triangulação já construída.

- Se o novo ponto está fora da triangulação (Figura 3.2a):

Encontra-se as arestas no bordo de DT que possam ser vistas a partir de x_i e denota-se por B o conjunto formado por estas arestas. Seja W o conjunto dos triângulos de DT cujas circunsferas possuem x_i em seu interior e $B' = \partial W$, onde ∂W é o bordo da união dos triângulos de W . Seja B'' o conjunto dado por:

$$B'' = (B \cup B') - (B \cap B') \text{ (Figura 3.2b).}$$

Elimina-se cada triângulo que está em W da triangulação DT e cria-se novos triângulos a partir de x_i e as arestas em B'' (Figura 3.2c).

Figura 3.2 – Inserção fora da triangulação.

(a) Ponto que fora da triangulação, (b) Arestas que serão afetadas com a inserção e (c) Novo ponto inserido.

- Se o novo ponto está dentro da triangulação (Figura 3.3a):

Seja W o conjunto dos triângulos de DT cujas circunsferas possuem x_i em seu interior. Seja $B = \partial W$ o conjunto das arestas do bordo de W (Figura 3.3b).

Elimina-se cada triângulo que está em W da triangulação DT e cria-se novos triângulos a partir de x_i e as arestas em B (Figura 3.3c). Pode ser demonstrado que B é um polígono estrelado com relação a x_i , isto é, todas as arestas de B podem ser vistas a partir de x_i [For94].

Figura 3.3 – Inserção dentro da triangulação.

(a) Ponto dentro da triangulação, (b) Arestas que serão afetadas com a inserção e (c) Novo ponto inserido.

A demonstração de que o algoritmo acima produz a triangulação de Delaunay pode ser encontrada em [For94] e se baseia no fato de que todos os triângulos criados são localmente Delaunay. Sejam abc e acd dois triângulos adjacentes e ac a aresta em comum, diz-se que abc é localmente Delaunay se a esfera passando por abc não contém d em seu interior como se mostra na figura 3.4.

Figura 3.4 – Triângulo abc localmente Delaunay.

Observação: Uma triangulação é Delaunay se, e somente se, todos os triângulos são localmente Delaunay. A prova deste fato pode ser encontrada em [For94](pg.200).

3.3.2 Outros algoritmos e comparação de tempos de execução

Nesta seção serão apresentados rapidamente dois algoritmos que geram a triangulação de Delaunay em R^2 chamados *Flipping* e *Divide-and-Conquer*. Uma descrição mais detalhada destes e outros algoritmos pode ser encontrada em [For94].

- **Flipping 2D.**- Este algoritmo é realizado em dois passos básicos (Um programa exemplo em *java* deste algoritmo pode ser encontrado em [Ung97]):
 - 1- Primeiro monta-se uma triangulação qualquer que não é necessariamente Delaunay a partir do conjunto de pontos em $\mathbb{R}^2$.
 - 2- Para cada par de triângulos opostos abc e acd , verifica-se se são localmente Delaunay em caso negativo realiza-se o *flipping*, ou seja, troca-se os triângulos abc e acd pelos triângulos dab e dbc . Por exemplo, na figura 3.5a os triângulos abc e acd não são localmente Delaunay, depois de realizado o *flipping*, como mostra a figura 3.5b, os triângulos dab e dbc são localmente Delaunay.
 - 3- Repete-se o passo 2 até que nenhum *flipping* seja realizado.

A complexidade deste algoritmo é de $O(n \log n + f)$, onde f é o numero de *flips*.

Figura 3.5 – Operação *flipping*.

(a) Triângulos opostos que não são localmente Delaunay abc e acd e (b) Novos triângulos depois do *Flipping* dab e dbc .

- **Divide-and-Conquer 2D.**- Este algoritmo consiste de dois passos básicos (Um programa exemplo em *java* deste algoritmo pode ser encontrado em [Luo97]):
 - Primeiro divide-se o conjunto de M pontos em dois subconjuntos de tamanhos aproximadamente iguais por uma linha vertical.
 - Recursivamente calcule-se o diagrama de Voronoi nas duas metades e depois junta-se ou mistura-se as triangulações realizando testes locais nos pontos de junção.

O tempo para juntar as duas triangulações no pior caso é linear. Assim o pior caso do algoritmo para n pontos é $O(n \log n)$.

A tabela 3.1 e a figura 3.6 apresentam uma comparação de tempos de execução, em segundos, entre os algoritmos *Incremental*, *Flipping* e *Divide-and-Conquer* em 2D. Esta comparação foi

extraída do trabalho feito por Fortuna [For94](pg. 227). Os algoritmos foram implementados na linguagem C e executados em uma máquina VAX 8550.

Número de pontos	Divide and conquer	Flipping	Incremental
500	2	2.1	2.2
1000	3.8	4.5	4.5
5000	21.5	26.2	24.2
10000	42.3	59.3	51
50000	222	316.9	285

Tabela 3.1 Desempenho em segundos de três algoritmos que constróem a triangulação de Delaunay.

Figura 3.6 – Desempenho em segundos de três algoritmos que constróem a triangulação de Delaunay.

3.3.3 Primitivas geométricas para a Triangulação de Delaunay

Se as arestas de uma triangulação estão orientadas de forma coerente, isto é, as arestas de todos os triângulos possuem uma mesma orientação positiva ou negativa, então pode-se verificar se uma aresta é vista de um ponto x_i , analisando o sinal do seguinte determinante:

$$M = \begin{vmatrix} P_{1x} & P_{1y} & 1 \\ P_{2x} & P_{2y} & 1 \\ x_{ix} & x_{iy} & 1 \end{vmatrix}$$

onde a aresta a ser verificada começa no ponto (p_{1x}, p_{1y}) e termina no ponto (p_{2x}, p_{2y}) e o ponto a ser verificado é (x_{ix}, x_{iy}) . Quando o valor do determinante é menor que zero a aresta

testada é visível a partir do ponto (x_{ix}, x_{iy}) , caso contrario não é (supondo orientação anti-horária).

Por exemplo, na figura 3.7 existem três arestas com orientações coerentes **a, b e c**.

Figura 3.7 – Teste de visibilidade de uma aresta a partir de um ponto.

Para determinar se o ponto $x_i = (5,5)$ é visto a partir de cada aresta (neste caso a orientação é anti-horária) se procede como segue:

Para a aresta **a** o valor do determinante é negativo, isto significa que o ponto x_i é visível a partir da aresta **a**.

$$M_a = \begin{vmatrix} 5 & 2 & 1 \\ 2 & 5 & 1 \\ 5 & 5 & 1 \end{vmatrix} \quad \text{Det}(M_a) = -9$$

Para as arestas **b e c** o valor da determinante é positivo, isto significa que estas arestas não são visíveis desde o ponto x_i .

$$M_b = \begin{vmatrix} 2 & 5 & 1 \\ 1 & 1 & 1 \\ 5 & 5 & 1 \end{vmatrix} \quad \text{Det}(M_b) = 12 \quad M_c = \begin{vmatrix} 1 & 1 & 1 \\ 5 & 2 & 1 \\ 5 & 5 & 1 \end{vmatrix} \quad \text{Det}(M_c) = 12$$

O teste para saber se o ponto x_i está no interior da circunsfera de um triângulo orientado dado pelos pontos p_1, p_2, p_3 pode também ser feito através da análise do determinante da seguinte matriz:

$$M = \begin{vmatrix} p_{1x} & p_{1y} & p_{1x}^2 + p_{1y}^2 & 1 \\ p_{2x} & p_{2y} & p_{2x}^2 + p_{2y}^2 & 1 \\ p_{3x} & p_{3y} & p_{3x}^2 + p_{3y}^2 & 1 \\ x_{ix} & x_{iy} & x_{ix}^2 + x_{iy}^2 & 1 \end{vmatrix}$$

Onde o triângulo orientado a ser verificado é formado pelos pontos (p_{1x}, p_{1y}) , (p_{2x}, p_{2y}) e (p_{3x}, p_{3y}) e o ponto a ser verificado é (x_{ix}, x_{iy}) . No caso de orientação anti-horária quando o valor do determinante é positivo o ponto (x_{ix}, x_{iy}) está dentro da circunsfera do triângulo, caso contrário o ponto está fora.

Por exemplo, no triângulo apresentado na figura 3.8 (com orientação anti-horária) serão testados os pontos $X_1 = (5,5)$ e $X_2 = (2,2)$.

Figura 3.8 – Teste para ver se um ponto esta fora ou dentro da circunsfera do triângulo.

Para o ponto $X_1 = (5,5)$ o valor do determinante é negativo, isto significa que o ponto está fora da circunsfera.

$$M_{x1} = \begin{vmatrix} 5 & 2 & 29 & 1 \\ 2 & 5 & 29 & 1 \\ 1 & 1 & 2 & 1 \\ 5 & 5 & 50 & 1 \end{vmatrix} \quad \text{Det}(M_{x1}) = -72$$

Para o ponto $X_2 = (2,2)$ o valor do determinante é positivo, isto significa que o ponto está dentro da circunsfera.

$$M_{x2} = \begin{vmatrix} 5 & 1 & 26 & 1 \\ 2 & 5 & 29 & 1 \\ 1 & 1 & 2 & 1 \\ 2 & 2 & 8 & 1 \end{vmatrix} \quad \text{Det}(M_{x2}) = 72$$

3.3.4 Algoritmo de Boissonnat

Este algoritmo [Boi88] assume que o conjunto de pontos $A \subset \mathbb{R}^3$ está distribuído em planos paralelos P_1 e P_2 . Embora este algoritmo apresente instabilidade, é útil para compreender o comportamento de uma triangulação de Delaunay em $\mathbb{R}^3$ de pontos distribuídos em planos paralelos. Este algoritmo possui tempo de execução $O(n \log n + T)$, onde n é o número total de pontos e T é o número de tetraedros na triangulação.

Está dividido em três etapas:

- 1- Geração da triangulação de Delaunay (TD) e diagrama de Voronoi (DV) nos planos paralelos.

A figura 3.9 apresenta as triangulações de Delaunay T_1 , T_2 e os diagramas de Voronoi V_1 , V_2 de alguns pontos nos planos P_1 e P_2 , respectivamente. Estas triangulações de Delaunay e diagramas de Voronoi podem ser gerados utilizando qualquer algoritmo que faça a triangulação 2D, por exemplo, o algoritmo incremental citado anteriormente.

Figura 3.9 – Triangulação de Delaunay e diagrama de Voronoi de pontos em planos paralelos.

- 2 - Construção do grafo

A figura 3.10 mostra o grafo G gerado pela projeção do Diagrama de Voronoi do plano P_1 no plano P_2 .

Figura 3.10 – Projeção do Diagrama de Voronoi de P_1 em P_2 .

3 - Geração da triangulação 3D

A cada vértice v do grafo G está associado um tetraedro na triangulação de Delaunay 3D que será chamada DT. No grafo G existem dois tipos de vértices os que são de V_1 ou V_2 e os vértices que surgem das interseções entre V_1 e V_2 (Figura 3.10). A partir destes dois tipos de vértices pose-se identificar dois tipos de tetraedros:

- Se um vértice v pertence a V_1 , ele corresponde a um tetraedro construído com uma face em P_1 e um vértice em P_2 (Figura 3.11a). A face em P_1 é o triângulo de T_1 que é dual ao vértice v . O vértice em P_2 é o dual do poliedro de Voronoi em P_2 onde a projeção do vértice v está contida.
- Se um vértice v pertence a V_2 ele corresponde a um tetraedro construído com uma face em P_2 e um vértice em P_1 (Figura 3.11b). A face em P_2 é o triângulo de T_2 que é dual ao vértice v . O vértice em P_1 é o dual do poliedro de Voronoi em P_1 onde a projeção do vértice v pertence.

Figura 3.11 – Tetraedros associados aos pontos dos diagramas de Voronoi.

(a) Tetraedros associados a vértices de V_1 e (b) Tetraedros associados a vértices de V_2 .

- Se v foi produzido pela interseção de duas arestas dos diagramas V_1 e V_2 então o tetraedro de DT associado ao vértice v é construído a partir de uma aresta em T_1 e uma aresta em T_2 (Figura 3.12).

A aresta em T_1 é a aresta dual à aresta de V_1 que formou a interseção com a aresta de V_2 , e a aresta em T_2 é a aresta dual à aresta de V_2 que formou a interseção com a aresta de V_1 .

Figura 3.12 – Tetraedros associados a um vértice que surgiu por uma interseção.

A figura 3.13a apresenta a triangulação de Delaunay 3D obtida a partir das duas triangulações nos planos paralelos P_1 e P_2 e a figura 3.13b apresenta os tetraedros de DT separados.

Figura 3.13 – Triangulação final produzida pelo algoritmo de Boissonnat.

(a) Triangulação de Delaunay 3D obtida a partir de triangulações 2D e

(b) Tetraedros separados da triangulação em 3D.

O problema em utilizar o algoritmo de Boissonnat é que o cálculo de interseções é caro computacionalmente e instabilidades podem ocorrer quando um vértice é projetado próximo a uma aresta, neste caso perturbações devem ser feitas. Outro problema é que é difícil gerar um critério para realizar perturbações em um caso genérico, principalmente quando a quantidade de pontos nos planos aumenta muito.

3.3.5 Algoritmo Quickhull

Nesta seção, será apresentado o algoritmo *Quickhull* criado por Barber *et al.* [Bar96]. Este algoritmo calcula o fecho convexo de um conjunto de pontos A em $\mathbb{R}^m$, $m \geq 2$. Barber *et al.* [Bar99] disponibilizaram também uma implantação deste algoritmo. Esta implementação foi utilizada na parte de reconstrução tridimensional deste trabalho por ser considerada rápida e bem documentada em relação a outras implementações.

Esta seção começa com a definição de fecho convexo na seção 3.3.5.1, explicando como uma triangulação de Delaunay em $\mathbb{R}^m$ pode ser obtida do fecho convexo em $\mathbb{R}^{m+1}$. Depois na seção 3.3.5.2 será descrito o algoritmo *Quickhull* fazendo uma comparação com o algoritmo incremental para fechos convexos.

3.3.5.1 Fecho convexo

O **fecho convexo** (*convex hull*) de um conjunto de pontos A é o menor conjunto convexo que contém estes pontos. As figuras 3.13a e 3.13b apresentam fechos convexo em $\mathbb{R}^2$ e $\mathbb{R}^3$.

Figura 3.14 – Exemplos de fechos convexos.

(a) Fecho convexo de 50 pontos em $\mathbb{R}^2$ e (b) Fecho convexo de 50 pontos em $\mathbb{R}^3$.

Imagens geradas com o programa demonstrativo de Lambert [Lam98].

O fecho convexo é uma construção fundamental para matemática e geometria computacional. Algumas aplicações que utilizam esta estrutura são: geração de malha, detecção de colisão, planificação urbana, estatística, etc.

Vários problemas podem ser reduzidos ao problema do fecho convexo, como por exemplo, interseção de semi-espacos, triangulação de Delaunay e diagrama de Voronoi.

Um fecho convexo pode ser representado por um conjunto de faces e um conjunto de listas de adjacências dando acesso aos vizinhos e vértices para cada face. Os elementos de bordo de

uma face são chamados *ridges*. Cada *ridge* expressa a adjacência de duas faces. Por exemplo, em $\mathbb{R}^3$ e com pontos em posição geral, faces são triângulos e *ridges* são arestas.

A triangulação de Delaunay de um conjunto de pontos A em $\mathbb{R}^m$ pode ser obtida a partir do fecho convexo em $\mathbb{R}^{m+1}$. A prova deste fato pode ser encontrada em [For94](pg. 200-203).

O cálculo da triangulação de Delaunay do conjunto de pontos A em $\mathbb{R}^m$ a partir de um fecho convexo em $\mathbb{R}^{m+1}$ pode ser realizado em três passos:

- Em um primeiro passo os pontos do conjunto A são projetados em um parabolóide (figura 3.15a).
- Depois, a partir dos pontos projetados no parabolóide o fecho convexo em $\mathbb{R}^{m+1}$ é calculado (figura 3.15b).
- Finalmente, a triangulação de Delaunay em $\mathbb{R}^m$ é obtida projetando o conjunto de *ridges* da parte inferior do fecho convexo de $\mathbb{R}^{m+1}$ em $\mathbb{R}^m$ (figuras 3.14c, 3.14d).

Figura 3.15 – Cálculo da triangulação de Delaunay a partir do fecho convexo.

(a) Projeção de pontos em $\mathbb{R}^2$ num parabolóide, (b) Fecho convexo em $\mathbb{R}^3$, (c) Projeção do fecho convexo em $\mathbb{R}^3$ a $\mathbb{R}^2$ e (d) *Wire frame* do fecho convexo em $\mathbb{R}^3$.

Imagens geradas com ajuda do programa demonstrativo de Lambert [Lam98].

3.3.5.2 O algoritmo *Quickhull*

O algoritmo *Quickhull* é uma variação do algoritmo incremental para fechos convexos. No algoritmo incremental os pontos são processados um de cada vez em ordem aleatória. A proposta do algoritmo *Quickhull* é processar os pontos em uma forma mais eficiente. O resultado é um algoritmo rápido para distribuições com pontos interiores. Programas exemplo dos algoritmos incremental e *Quickhull* para fechos convexos podem ser encontrados em [Lam98].

O algoritmo incremental para fechos convexos insere repetidamente pontos ao fecho convexo dos pontos processados anteriormente. Um novo ponto p (figura 3.16a) no algoritmo incremental é processado em três passos:

- Primeiro localiza-se as faces visíveis desde p (figura 3.16b). O bordo destas faces visíveis é o conjunto de "ridges horizonte" para o ponto p .
- Segundo construi-se um cone de novas faces a partir do ponto p e suas "ridges horizonte" (figura 3.16c).
- Por último elimina-se as faces visíveis formando um fecho convexo entre o novo ponto p e os pontos processados anteriormente (figura 3.16d).

Figura 3.16 – Inserção de um novo ponto com o algoritmo incremental para fechos convexos.

(a) Fecho convexo anterior, (b) Faces visíveis desde o ponto p , (c) Cone entre o ponto p e os *ridges* horizonte e (d) Novo fecho convexo.

A variação que o algoritmo *Quickhull* propõe é processar o ponto mais distante em vez de um ponto aleatório. Assim, podemos comparar o algoritmo *Quickhull* com o algoritmo incremental mudando a etapa de seleção no *Quickhull*. Se o algoritmo *Quickhull* seleciona um ponto aleatório em vez do ponto mais distante, seria igual ao algoritmo incremental. Através de testes empíricos realizados por Barber *et al.* [Bar96], determinou-se que a execução do algoritmo *Quickhull* é mais rápida que o algoritmo incremental por ele processa menor quantidade de pontos. Também, *Quickhull* reutiliza a memória ocupada pelas faces anteriores.

As seguintes imagens são resultados de testes empíricos realizados por Barber em distribuições aleatórias de pontos em um cubo unitário. O objetivo dos testes e comparar o desempenho das estratégias de inserção: seleção aleatória do algoritmo incremental e seleção do ponto mais distante do algoritmo *Quickhull* para fechos convexos.

A figura 3.17a apresenta a quantidade de hiperplanos criados, a figura 3.17b mostra o número de testes de distâncias e a figura 3.17c mostra a quantidade de pontos processados pelos dos algoritmos. Barber mostrou também, empiricamente que o número de pontos processados por *Quickhull* é proporcional ao número de vértices de saída como também mostra a figura 3.17c.

Figura 3.17 – Desempenho das estratégias de inserção: seleção aleatória do algoritmo incremental e seleção do ponto mais distante do algoritmo *Quickhull*.

(a) Quantidade de hiperplanos criados, (b) Quantidade de testes de distância e (c) Relação entre pontos processados e pontos de saída do algoritmo *Quickhull*.

Pipeline de Reconstrução tridimensional

Neste capítulo será descrito o *pipeline* de reconstrução tridimensional desenvolvido neste trabalho. O algoritmo de reconstrução utilizado em um dos processos deste *pipeline* é uma nova proposta de reconstrução tridimensional chamado β -Conexão. Este novo algoritmo foi desenvolvido em C++ e está baseado em geometria computacional, mais especificamente na triangulação de Delaunay 3D, e nas idéias do algoritmo de Nonato [Non98, Non01].

Uma característica importante do algoritmo β -Conexão, é que da mesma forma que o algoritmo de Nonato, satisfaz os critérios de re-amostragem e 3-variedade LP.

- O critério de re-amostragem exige que a interseção do modelo gerado com os planos de corte resulte nos contornos originais.
- O critério de 3-variedade LP impõe que o modelo gerado seja sempre uma 3-variedade linear por partes.

Este capítulo descreve, na seção 4.1, os dados de entrada, dados de saída e a ordem com que os processos principais ocorrem no *pipeline* de reconstrução. As seções 4.2, 4.3, 4.4 e 4.5 descrevem cada um dos processos principais do *pipeline*.

4.1 Visão geral do pipeline

O esquema geral do *pipeline* de reconstrução pode ser apreciado na figura 4.1.

As entradas para o *pipeline* de reconstrução são:

- Um conjunto de arquivos $arq_1, \dots, arq_n$, $n \geq 2$, contendo o conjunto de seções planares $\pi_1 \dots \pi_n$, a partir das quais se quer reconstruir o objeto original. Cada seção planar π_i contém um conjunto de contornos $C_1, \dots, C_k$, $k \geq 2$, representando o bordo de regiões. Tais contornos podem ser obtidos através de um processo de segmentação de imagens.
- O parâmetro β , que especifica a "densidade" com que as regiões nos planos paralelos serão conectadas (os detalhes sobre este parâmetro serão explicados melhor na seção 4.4).

A saída do *pipeline* é um objeto tridimensional formado por um conjunto de tetraedros $T_1 \dots T_t$, $t \geq 3$.

Na implementação deste trabalho foi escolhido o formato da biblioteca VTK (*Visualization ToolKit*) [Sch98], [Kit01] para representar os arquivos de entrada e de saída. Esta biblioteca é de domínio público e de fácil utilização.

Figura 4.1 – Dados de entrada e saída do *pipeline* de reconstrução.

No *pipeline* de reconstrução existem quatro sub-processos principais (figura 4.2), que são:

- Pré-Processo de contornos
- Processo de cálculo de deslocamento
- Processo de reconstrução
- Processo de união de sub-sólidos

O *pipeline* começa executando a primeira etapa de pré-processamento (seção 4.2) para cada um dos n arquivos de entrada. Como resultado desta etapa, temos uma lista de n contornos orientados e que respeitam a triangulação final. Estes contornos irão passar por uma segunda etapa de cálculo de deslocamento (seção 4.3), com o objetivo de facilitar a criação do sólido final. A terceira etapa é o processo de reconstrução com o novo algoritmo β -Conexão (seção 4.4). Nesta etapa, a partir de cada dois níveis consecutivos de contornos e o parâmetro β , gera-se um sub-sólido. Finalmente na última etapa União de sub-sólidos (seção 4.5) cria-se o sólido final.

Figura 4.2 – Sub-processos principais do *pipeline* de reconstrução.

A seguir, cada uma destas etapas são explicadas.

4.2 Pré-Processo de contornos

Nesta etapa cada nível de dados de entrada é processado individualmente, com a finalidade de obter contornos que facilitem a etapa de reconstrução tridimensional (Figura 4.3).

Figura 4.3 – Pré-Processo de contornos

O pré-processamento está dividido em três sub-processos (Figura 4.4):

- Leitura de contornos
- Orientação de contornos
- Verificação de arestas Delaunay

A partir de um arquivo representando um nível de entrada, o pré-processamento começa lendo o conjunto de contornos. Depois disto, os processos de Orientação de Contornos e de Verificação de Arestas Delaunay são executados.

Figura 4.4 – Sub-processos principais do *pipeline* de reconstrução.

Estes sub-processos são detalhados a seguir.

4.2.1 Leitura de contornos

Neste primeiro passo do pré-processamento, um arquivo arq_i contendo os pontos dos contornos $C_1...C_k$, $k \geq 1$, de uma seção planar π_i é lido. O formato de arquivo adotado nesta implementação foi o formato do VTK. Esta etapa poderia ser modificada com facilidade para aceitar arquivos com outros formatos que especifiquem contornos.

4.2.2 Orientação de contornos

A partir de um nível de entrada contendo um conjunto de contornos num plano, como se mostra na figura 4.5, o algoritmo de reconstrução precisa identificar quais regiões são internas e quais são externas. O objetivo desta etapa é ajudar nesta identificação.

Figura 4.5 – Conjunto de contornos em um nível de entrada.

Para identificar regiões internas e externas num conjunto de contornos, três passos devem ser realizados:

- Em um primeiro passo, verifica-se a não existência de interseções entre os contornos. Este cálculo pode ser executado em tempo $O(n \log n)$ [Cor90](pg. 892-897), [Pre85](pg.279-287).
- O segundo passo é a classificação dos contornos como internos ou externos. Um contorno é externo quando nenhum outro contorno o contém, caso contrário é interno. Depois de ter certeza de que não existem interseções entre os contornos com passo anterior, o cálculo para saber se um polígono está dentro de outro pode ser realizado em tempo $O(n)$ [Pre85](pg.271-287). Logo, pode-se deduzir que o cálculo de contornos internos e externos pode ser executado no melhor caso em tempo $O(n)$ e no pior caso em tempo $O((n-1)!)$.
- Por último os contornos são orientados. Todos os contornos externos são orientados de forma anti-horária e a partir desta orientação qualquer contorno que esteja contido em um outro deverá estar com uma orientação contrária do contorno que o contém.

Um conjunto de contornos respeitando as orientações acima é chamado de **conjunto de contornos coerentes** (figura 4.6). Caminhando por um contorno com orientação coerente tudo o que está à esquerda é denominado **região interna**, e tudo o que permanece à direita será denominado **região externa**. Os contornos coerentes serão de muita utilidade para o algoritmo de reconstrução tridimensional da seção 4.4.

Figura 4.6 – Conjunto de contornos coerentes.

A etapa seguinte do pré-processamento explicada a seguir é a verificação de arestas Delaunay.

4.2.3 Verificação de arestas Delaunay

Na etapa de reconstrução tridimensional será necessário calcular a triangulação de Delaunay 3D a partir dos vértices dos contornos contidos em dois níveis consecutivos. Nesta triangulação, todas as arestas dos contornos devem estar presentes. O objetivo desta etapa é garantir este fato.

Com o algoritmo de Boissonnat, da seção 3.3.4, explicou-se o comportamento de uma triangulação de Delaunay 3D a partir de pontos distribuídos em seções planares. Segundo este comportamento, se conseguirmos garantir que as arestas dos contornos de um plano π_i estão na triangulação de Delaunay 2D, estaremos garantindo também que estas aparecerão na triangulação de Delaunay 3D dos pontos nos planos π_i e π_{i+1} .

Consideremos o exemplo de um nível π_i contendo apenas um contorno, como ilustra a figura 4.7a. A verificação de arestas Delaunay para este nível pode ser realizada em dois passos:

- O primeiro passo consiste em executar a triangulação de Delaunay 2D com pontos dos contornos do nível em questão (figura 4.7b).
- O segundo passo verifica se todas as arestas dos contornos do plano π_i estão presentes na triangulação. Se existem arestas que não aparecem na triangulação, estas são divididas pela metade e o processo se repete (Figuras 4.7c e d).

Estes dois passos são repetidos até que todas as arestas nos contornos apareçam na triangulação de Delaunay. A figura 4.7d apresenta a triangulação de Delaunay 2D depois da divisão da aresta.

Figura 4.7 – Arestas Delaunay.

O passo seguinte no processo do *pipeline* de reconstrução é o cálculo de deslocamento.

4.3 Cálculo de deslocamento

Os sub-sólidos gerados pelo algoritmo de reconstrução da etapa de reconstrução deverão ser unidos na última etapa para formar o sólido final. O objetivo do cálculo de deslocamento é facilitar esta união.

Para esta finalidade, em cada plano π_i devem ser calculados dois dados:

- O número de pontos no plano, ou seja, o somatório do número de pontos de todos seus contornos.
- O somatório do número de pontos dos níveis anteriores.

Estes dois dados facilitarão a criação do sólido final na última de etapa do *pipeline*.

A seguir apresentamos o novo algoritmo de reconstrução tridimensional β -Conexão.

4.4 Reconstrução tridimensional - β -Conexão

Nesta seção, apresentamos o β -Conexão, o novo algoritmo de reconstrução tridimensional a partir de seções planares proposto neste trabalho. Baseado na triangulação de Delaunay, e nas idéias do algoritmo de Nonato [Non98, Non01], criou-se uma nova técnica para resolver o problema da correspondência. Esta técnica oferece múltiplas escolhas de conexão entre regiões contidas em seções planares adjacentes.

O esquema geral de entradas e saída do algoritmo β -Conexão é apresentado na figura 4.8.

Este algoritmo recebe duas entradas:

- A primeira entrada são dois níveis ou seções planares consecutivas π_i e π_{i+1} , contendo um conjunto de contornos já pré-processados.
- A segunda entrada é o β -Parâmetro. Este parâmetro expressa a densidade com que os contornos serão conectados.

A saída deste algoritmo é o sub-sólido reconstruído.

Figura 4.8 – Esquema de entradas e saídas do algoritmo β -Conexão.

Este algoritmo está dividido em 6 etapas (figura 4.9):

- Triangulação de Delaunay 3D
- Classificação de arestas
- Classificação de tetraedros
- Cálculo de componentes conexas
- Desconexão de componentes
- Subdivisão de tetraedros

O β -Conexão começa executando a triangulação de Delaunay 3D com pontos contidos em duas seções planares adjacentes. Em seguida, as arestas desta triangulação, que estão contidas nas duas seções planares são classificadas. A partir desta classificação de arestas, os tetraedros da triangulação são classificados também. Baseado nesta última classificação, o β -Conexão determina as componentes conexas do modelo, ou seja, quais regiões ficarão conectadas entre si. Para isso, calcula-se o caminho mais curto em tetraedros entre cada par de regiões nos níveis consecutivos. Depois, utiliza-se um valor inteiro positivo chamado β (estabelecido pelo usuário) para expressar uma mínima distância em tetraedros. Esta distância mínima serve como condição para que as regiões sejam conectadas. Isto é, para um $\beta = n$, todas as regiões que estejam a uma distância em tetraedros menor ou igual a n ficarão conectadas. Desta forma, a partir de um conjunto de contornos em dois níveis consecutivos, é possível gerar uma família de objetos reconstruídos atribuindo diferentes valores a β . Após ter determinado as componentes conexas, ou grupos de regiões que ficarão conectadas no

modelo, estas devem ser isoladas no processo de desconexão de componentes. Finalmente, para refinar o modelo, um processo de subdivisão de tetraedros é realizado.

Figura 4.9 – Etapas do algoritmo de reconstrução β -Conexão.

O algoritmo β -Conexão, da mesma forma que o algoritmo de Nonato satisfaz os critérios de Re-amostragem e 3-variedade LP.

- O critério de re-amostragem exige que a interseção do modelo gerado com os planos de corte resulte nos contornos originais.
- O critério de 3-variedade LP impõe que o modelo gerado seja sempre uma 3-variedade linear por partes.

Em seguida, cada um dos processos deste algoritmo será detalhado.

4.4.1 Triangulação

A triangulação de Delaunay 3D denotada DT é o primeiro passo deste algoritmo de reconstrução. Esta triangulação é construída com pontos de duas seções planares consecutivas de entrada, como é mostrado na figura 4.10.

Na implementação do algoritmo β -Conexão, utilizou-se o algoritmo QuickHull [Bar96], [Bar99] para gerar a triangulação de Delaunay 3D. Embora o QuickHull tenha demonstrado muita rapidez durante os testes de reconstrução, alguma tetraedralização é necessária no caso de pontos co-circulares. Outras implementações interessantes de triangulações de Delaunay 3D podem ser encontradas em [CGA01] e [Müc98].

Figura 4.10 – Triangulação de Delaunay 3D.

(a) Regiões em seções planares paralelas, (b) triangulação de Delaunay 3D com pontos destas regiões.

4.4.2 Classificação de arestas

A partir da triangulação de Delaunay 3D DT feita no passo anterior e dos contornos orientados, as arestas que estão nos planos π_i e π_{i+1} são classificadas. Isto é, caminhando por um contorno com orientação coerente, as arestas de uma triangulação DT contidas nos planos π_i e π_{i+1} (figura 4.11 a) podem ser classificadas da seguinte forma.:

- As arestas que estão à esquerda do contorno serão chamadas **arestas internas**
- As arestas que permanecem à direita do contorno serão chamadas **arestas externas**
- As arestas dos planos π_i e π_{i+1} , que não são arestas internas nem externas serão chamadas **arestas de contorno**.

A figura 4.11b apresenta o resultado da classificação de arestas.

Figura 4.11 – Classificação de arestas.

(a) Exemplo de uma triangulação de Delaunay em *Wirre frame*,

(b) Arestas internas, Externas e de contorno.

Note que somente as arestas de DT que estão nos planos π_i e π_{i+1} são classificadas. As arestas que estão entre π_i e π_{i+1} não são classificadas.

Os **esqueletos internos** e **externos** do diagrama de Voronoi são definidos como os conjuntos de arestas do diagrama de Voronoi bidimensional em π_i e π_{i+1} que são duais respectivamente às arestas internas e externas de DT em π_i e π_{i+1} (Figura 4.12). Note que, de acordo com a definição acima, os esqueletos internos são sempre limitados, enquanto que, os esqueletos externos podem ser ilimitados, pois as arestas do diagrama de Voronoi que são duais às arestas externas que estão no bordo do fecho convexo são ilimitadas.

Figura 4.12 – Esqueletos internos e externos diagrama de Voronoi.

(a) Conjunto de contornos com arestas classificadas,
(b) Esqueletos internos, (c) Esqueletos externos.

Os esqueletos do diagrama de Voronoi desempenham um papel importante na definição das componentes conexas que será definida mais adiante.

Antes de definir o que são as componentes conexas do modelo, é preciso classificar os tetraedros de DT .

4.4.3 Classificação de tetraedros

Tendo classificadas as arestas nos planos π_i e π_{i+1} pelo passo anterior, os tetraedros da triangulação *DT* podem ser classificados também. Como já foi explicado no algoritmo de *Boissonat* na seção 2.3.3, em uma triangulação de Delaunay 3D *DT* a partir de pontos em planos paralelos existem dois tipos de tetraedros:

- O primeiro tipo de tetraedro tem 3 arestas em um dos dois planos paralelos π_i ou π_{i+1} , este tipo de tetraedro será chamado **tetraedro do tipo 1**.
- O segundo tipo de tetraedro possui uma aresta em cada um dos planos π_i e π_{i+1} , este segundo tipo de tetraedro será chamado **tetraedro do tipo 2**.

A partir destes dois tipos de tetraedros, tipo 1 e 2, e da classificação de arestas pode-se diferenciar cinco tipos de tetraedros em *DT* (figura 4.13a), da seguinte maneira:

- Se um tetraedro t é do tipo 1 então:
 - t possui uma ou mais arestas internas e não possui arestas externas, ou possui três arestas de contorno de um contorno com orientação anti-horária, t será chamado **tetraedro interno** (Figura 4.13b).
 - t possui uma ou mais externas e não possui arestas internas, ou possui três arestas de contorno de um contorno com orientação horária, t será chamado **tetraedro externo** (Figura 4.13c).
- Se um tetraedro t é do tipo 2 então:
 - t possui duas arestas internas, t será chamado **tetraedro reverso interno** (Figura 4.13d).
 - t possui uma aresta interna e uma externa, t será chamado **tetraedro reverso externo** (Figura 4.13e).
 - t possui duas arestas de contorno e não tem arestas internas nem externas, t será chamado **tetraedro redundante** (Figura 4.13f).
 - t possui duas arestas externas ou, uma aresta de contorno e uma aresta externa, t será chamado **tetraedro externo** (Figura 4.13c).
 - t possui uma aresta de contorno e uma aresta interna, t será chamado **tetraedro interno** (Figura 4.13b).

A figura 4.13g apresenta um exemplo do resultado da classificação de tetraedros.

Figura 4.13 – Classificação de tetraedros.

- (a) Exemplo de uma Triangulação de Delaunay com arestas classificadas,
 (b) Tetraedros internos, (c) Tetraedros externos, (d) Tetraedros reversos internos,
 (e) Tetraedros reversos externos, (f) Tetraedros redundantes, (g) Resultado da classificação de tetraedros.

A existência dos tetraedros reversos em DT origina-se por causa de uma interseção entre esqueletos internos e externos dos diagramas de Voronoi em π_i e π_{i+1} .

Isto é:

- Um tetraedro reverso interno em DT origina-se pela interseção dos esqueletos internos em π_i e π_{i+1} (Figuras 4.14a e b).
- Um tetraedro reverso externo em DT origina-se pela interseção de um esqueleto interno e um esqueleto externo em π_i e π_{i+1} (Figuras 4.14c e d).

A demonstração destes fatos pode ser encontrada no trabalho feito por Nonato [Non98, Non01].

Figura 4.14 – Origem dos tetraedros reversos.

- (a) Tetraedro reverso interno originado pela interseção de esqueletos internos do digrama de Voronoi,
- (b) Outra vista da figura a,
- (c) Tetraedro Externo originado pela interseção de esqueletos internos e externos do digrama de Voronoi,
- (d) Outra vista da figura c.

No algoritmo de Nonato, considera-se que dois contornos $c_1 \in \pi_i$ e $c_2 \in \pi_{i+1}$ estão geometricamente bem posicionados se e somente se na triangulação de Delaunay tridimensional com pontos de π_i e π_{i+1} existe um tetraedro reverso com vértices em c_1 e c_2 .

A heurística para resolver o problema da correspondência do algoritmo β -Conexão, considera também, entre outras coisas, a existência destes tetraedros reversos.

A etapa explicada a seguir é o cálculo de componentes conexas.

4.4.4 Cálculo de componentes conexas

Definir quais regiões devem ser conectadas é uma tarefa difícil, já que a informação destas não é suficiente para garantir uma correta correspondência. Nesta seção, será apresentada uma nova heurística baseada em uma distância mínima em tetraedros, representada por um valor inteiro positivo chamado β . Esta distância é utilizada como uma condição para que as regiões sejam conectadas. Isto é, para um $\beta = n$, todas as regiões que estejam a uma distância em tetraedros menor ou igual a n ficarão conectadas. Desta forma, a partir de um conjunto de regiões contidas em níveis consecutivos π_i e π_{i+1} , é possível gerar uma família de objetos reconstruídos atribuindo diferentes valores a β .

Sejam C_1 e C_2 dois conjuntos de contornos representando o bordo de dois conjuntos de regiões R_1 e R_2 contidos nas duas seções planares de entrada π_i e π_{i+1} , e seja DT a triangulação de Delaunay 3D gerada com pontos em C_1 e C_2 .

A triangulação DT (figura 4.15a) pode ser considerada, de forma lógica, como um grafo não dirigido G (Figura 4.15b) contendo dois tipos de nós. Tal grafo pode ser obtido da seguinte forma:

- Os nós de G
 - Cada região $r \in R_1 \cup R_2$, junto com todos seus tetraedros internos, é considerada como um nó chamado **nó região**.
 - Cada tetraedro $t \in DT$, do tipo redundante ou externo, é considerado como um nó chamado **nó externo**.
- As arestas de G
 - Se duas regiões r_1 e r_2 , possuem tetraedros internos com uma face em comum, ou se existe um tetraedro reverso interno com arestas em r_1 e r_2 , então, considera-se uma aresta em G conectando os nós região que representam r_1 e r_2 .
 - Se um tetraedro externo ou redundante, representado por um nó externo w , possui uma face em comum com um tetraedro interno de uma região r , então, considera-se uma aresta em G entre os nós w e r .
 - Considera-se também uma aresta entre dois nós do tipo externo, se os tetraedros que representam estes nós possuem uma face em comum.

Figura 4.15 – Grafo lógico associado a uma triangulação de Delaunay 3D com pontos distribuídos em seções planares.

(a) Exemplo de uma triangulação de Delaunay 3D, (b) Grafo gerado a partir da triangulação da figura a.

Por definição, as componentes conexas de um grafo não dirigido são os subconjuntos de nós sob uma relação "é alcançável desde" [Cor90](pg.88). No contexto do algoritmo β -Conexão, chamaremos **componentes conexas** aos subconjuntos de regiões em R_i e R_{i+1} que estão unidos sob a relação "distância mínima" especificada pelo parâmetro β .

A partir do grafo lógico G obtido de DT , as componentes conexas do modelo podem ser calculadas em dois passos:

- **Cálculo de distâncias em tetraedros entre regiões**

Para cada nó região r em G aplica-se o algoritmo de busca em largura (*Breadth-first search*) [Cor90](pg.469-477), tomando como nó inicial o próprio nó região r . Este algoritmo, a partir de um nó inicial, expande a fronteira entre os nós conhecidos e os desconhecidos de forma sistemática. Isto é, o algoritmo alcança todos os nós a uma distância k antes de descobrir qualquer outro nó a uma distância $k+1$. Este algoritmo produz também uma árvore (*Breadth-first tree*) contendo todos os nós alcançáveis desde o nó inicial. A execução deste algoritmo é feita em tempo linear, $O(V+E)$, onde V é o número de nós e E é número de arestas em G [Cor90](pg. 471).

Para qualquer nó região $s \in G$, $s \neq r$, o caminho na árvore desde s até r corresponde ao caminho mais curto entre s e r .

Depois de ter sido feito este processo para cada nó região r em G , origina-se um novo grafo J de distâncias entre regiões. A figura 4.16 apresenta o grafo de distâncias obtido a partir do grafo da figura 4.15b.

Figura 4.16 –Grafo de distâncias entre regiões.

- **Determinação de componentes conexas**

A partir do parâmetro β e do grafo de distâncias entre regiões J , é fácil determinar as componentes conexas do modelo. Isto é, para um $\beta = n$, todas as regiões que no grafo J estejam a uma distância menor ou igual a n , ficarão em uma mesma componente conexa.

Por último, cada tetraedro $t \in DT$, é atribuído à componente conexa da região que este está associado. Os tetraedros externos e redundantes que possuem vértices em duas componentes conexas distintas não são atribuídos a nenhuma componente conexa e serão eliminados na seguinte etapa.

Desta forma, segundo o grafo J , se tem:

- Com $\beta = 0$, cada região forma uma componente conexa (Figura 4.17a).
- Com $\beta = 1$, todas as regiões que possuam um tetraedro reverso interno entre elas ou tetraedros internos com faces em comum ficarão conectadas (Figura 4.17b).
- Com $\beta = d$, $d \geq 2$, todas as regiões que estejam a uma distância menor o igual a d ficarão conectadas (Figura 4.17c).

Figura 4.17 – Componentes conexas de um modelo

- (a) Componentes conexas com $\beta = 0$, (b) Componentes conexas com $\beta = 1$,
(c) Componentes conexas com $\beta = 3$.

4.4.5 Desconexão de componentes conexas

Após ter calculado as componentes conexas do modelo, estas devem ser isoladas. Com esta finalidade, os tetraedros externos e redundantes com vértices em duas componentes conexas distintas têm que ser eliminados. Após a eliminação de tetraedros, algumas componentes conexas poderiam não estar totalmente separadas, gerando vértices e arestas singulares. A figura 4.18 apresenta um modelo que, com $\beta = 2$, gera uma aresta singular e um vértice singular. Para eliminar estas singularidades, os vértices singulares devem ser duplicados e depois deslocados até uma posição apropriada.

Figura 4.18 – Exemplo de desconexão de componentes com singularidades

(a) Contornos do modelo exemplo, (b) Componente conexas,
 (c) Aresta Singular, (d) Vértice singular, (e) Componente Conexas separadas

Existem também casos especiais com $\beta = 0$ junto com tetraedros reversos internos no modelo. A existência de um tetraedro reverso indica uma forma sobreposição entre duas regiões. Um tetraedro deste tipo não pode ser separado somente com translação de vértices porque geraria singularidades (Figura 4.19c). Para evitar estas singularidades, estes tetraedros têm que ser duplicados (Figura 4.19d).

Figura 4.19 – Caso especial com $\beta = 0$, e tetraedros reversos.

(a) Singularidade gerada por arrancar um tetraedro reverso,

(b) Tetraedro Reverso duplicado para tampar as singularidades.

A última etapa do algoritmo β -Conexão é a subdivisão de tetraedros explicada a seguir.

A seguir será explicado como um processo de subdivisão de tetraedros pode ser usado para eliminar estas singularidades.

4.4.6 Subdivisão de tetraedros

Para satisfazer o critério de re-amostragem, isto é, que os contornos originais possam ser recuperados interceptando o modelo com os planos de corte originais, devemos eliminar os tetraedros externos (tetraedros dos tipos externo e reverso externo), o problema é que esta eliminação pode gerar singularidades. Por exemplo, a eliminação de um tetraedro reverso externo (com uma aresta externa) (Figura 4.20a), gera uma aresta singular no interior do contorno que contém este tetraedro (Figura 4.20b).

Por outro lado, a fim de garantir o segundo critério de reconstrução, ou seja, garantir que o modelo reconstruído seja sempre uma 3-variedade LP, os tetraedros cuja eliminação dá origem a singulares devem ser mantidos. Uma maneira de fazer com que a presença destes tetraedros não viole o primeiro critério de reconstrução, é introduzir um ponto na aresta externa, subdividir os tetraedros que incidem nela (Figura 4.20c) e transladar este ponto para uma posição entre os contornos (Figura 4.20d).

Figura 4.20 – Aparição de singularidades e sua eliminação com subdivisão de tetraedros.

- (a) Exemplo de tetraedro reverso externo,
 (b) Eliminação do tetraedro causando uma singularidade em aresta,
 (c) Subdivisão de tetraedro, (d) Traslado dos novos pontos da subdivisão.

A idéia principal do processo de subdivisão de tetraedros é:

1. Eliminar os tetraedros externos que não geram singularidades.
2. Subdividir aqueles tetraedros externos que não devem ser eliminados no caso 1.
3. Transladar os novos pontos inseridos na subdivisão até uma posição intermediária.

Estas subdivisões de tetraedros podem ser feitas da seguinte maneira:

Seja t é um tetraedro com uma face contida em um dos planos adjacentes (tetraedro do tipo 1):

- Se t possui apenas uma aresta externa, acrescenta-se um vértice no ponto médio desta aresta e divide-se t em dois novos tetraedros (Figura 4.21a).
- Se t possui duas arestas externas, acrescenta-se um vértice no ponto médio de cada uma delas e divide-se t em três novos tetraedros (Figura 4.21b).
- Se t possui três arestas externas, acrescenta-se um vértice no ponto médio de cada uma delas e divide-se t em quatro novos tetraedros (Figura 4.21c).

Figura 4.21 – Subdivisão de tetraedros do tipo 1.

(a) Com uma aresta externa, (b) Com duas arestas externas, (c) Com três arestas externas.

Seja t é um tetraedro com uma aresta em cada um dos planos adjacentes (tetraedro do tipo 2):

- Se t possui apenas uma aresta externa, acrescente um vértice no ponto médio desta aresta e divida t em dois novos tetraedros (Figura 4.22a).
- Se t possui duas arestas externas, acrescente um vértice no ponto médio de cada uma delas e divida t em quatro novos tetraedros (Figura 4.22b).

Figura 4.22 – Subdivisão de tetraedros do tipo 2.

(a) Com uma aresta externa, (b) Com duas arestas externas.

Note que a subdivisão de tetraedros respeita os contornos. Por exemplo, se um tetraedro a ser subdividido tem uma aresta de contorno, a face que contém esta aresta é mantida intacta, ou seja, somente as faces com arestas externas são subdivididas.

Desta forma, o modelo gerado pela aplicação do processo de subdivisão de tetraedros satisfaz os dois critérios de reconstrução: re-amostragem e 3-variedade LP. A figura 4.23a apresenta um conjunto exemplo de contornos em seções planares adjacentes e as figuras 4.23b, 4.23c e 4.23d apresentam os sub-sólidos, com β igual 0, 1, e 3 respectivamente, depois do processo de subdivisão de tetraedros.

Figura 4.23 – Exemplos de sub-sólidos.

(a) Conjunto de contornos de exemplo, (b) Sub-sólido com $\beta = 0$,

(c) Sub-sólido com $\beta = 1$, (d) Sub-sólido com $\beta = 3$.

Com a subdivisão de tetraedros, é finalizada a etapa do processo de reconstrução pelo algoritmo β -Conexão, dando origem a um sub-sólido.

Todos os sub-sólidos criados nesta etapa serão unidos pelo último processo, união dos sub-sólidos, do *pipeline* de reconstrução.

4.5 União de Sub-Sólidos

A união dos sub-sólidos é o último processo do *pipeline* de reconstrução. Neste processo, todos os sub-sólidos criados pelo algoritmo de reconstrução β -Conexão devem ser unidos para formar um sólido final. Já que o algoritmo de reconstrução é executado entre cada dois níveis consecutivos, dois sub-sólidos consecutivos possuem um conjunto de regiões em uma seção planar em comum. Por outro lado, tem-se conhecimento, pelo algoritmo de Boissonat (seção 3.3.4) para pontos distribuídos em seções planares, que as triangulações de Delaunay 2D dos planos estão contidas na triangulação de Delaunay 3D também destes planos. A partir destas regiões em comum com triangulações idênticas, os sub-sólidos criados pelo algoritmo β -Conexão podem ser unidos com facilidade.

A figura 4.24a apresenta 3 níveis consecutivos de seções planares. As figuras 4.24b, 4.24c e 4.24d, apresentam os sólidos finais com valores de β iguais a 0, 1 e 3 respectivamente, depois do processo de união de sub-sólidos.

Figura 4.24 – Exemplos de sólidos finais.

(a) Conjunto de seções planares em três seções planares,
 (b) Sólido final com $\beta = 0$, (c) Sólido final com $\beta = 1$, (d) Sólido final com $\beta = 3$.

O capítulo seguinte apresenta os resultados, conclusões e recomendações obtidas em este trabalho.

Resultados e Conclusões

Neste último capítulo apresentamos: os resultados, contribuições e conclusões obtidos com este trabalho. A seção 5.1, começa apresentando alguns resultados obtidos com o novo algoritmo de reconstrução β -Conexão. Em seguida, na seção 5.2, são descritas algumas contribuições relevantes. Finalmente, na seção 5.3, são apresentados algumas conclusões e trabalhos futuros.

5.1 Resultados

A figura 5.1a apresenta o resultado do algoritmo β -Conexão com regiões distribuídas em duas seções planares adjacentes. As figuras 5.1b, 5.1c, e 5.1d apresentam os modelos gerados com valores de β igual a 0, 1 e 3 respectivamente.

Figura 5.1 – Diferentes objetos a partir de um mesmo conjunto de seções planares

(a) Conjunto de seções planares adjacentes, (b) Reconstrução com $\beta = 0$,

(c) Reconstrução com $\beta = 1$, (d) Reconstrução com $\beta = 3$.

Durante os testes de reconstrução observou-se um fato interessante. Quando duas regiões em planos adjacentes estão bem posicionadas (figura 5.2a), aparecem na triangulação 3D um ou mais tetraedros reversos (figura 5.2b e 5.2c). Como consequência, a distância segundo o grafo associado à triangulação 3D é de 1 (figura 5.2d). Se nós afastamos ligeiramente estas duas regiões, aparece uma barreira de tetraedros do tipo redundante

(figura 5.2e), aumentando a distância entre elas, como pode ser observado a partir do grafo associado na figura 5.2f. Se nós afastamos ainda mais estas duas regiões, a barreira de tetraedros redundantes é maior (figura 5.2g), e como consequência a distância entre estas duas regiões aumenta ainda mais (veja grafo na figura 5.2h). Desta forma quanto maior a "distância" entre as regiões, maior deverá ser o valor do parâmetro β para que estas possam ser conectadas.

Figura 5.2 – Fato interessante acontecido durante o cálculo de componentes conexas.

- (a) Contornos bem posicionados, (b) Triangulação 3D,
 (c) Aparição de tetraedro reverso em contornos bem posicionados, (d) Grafo associado à figura a,
 (e) Aparição de uma barreira de tetraedros redundantes em contornos ligeiramente afastados,
 (f) Grafo associado à figura e, (g) Barreira maior de tetraedros reversos em contornos mais afastados,
 (h) Grafo associado à figura g.

O exemplo da figura 5.3 apresenta o comportamento do pipeline de reconstrução com um mesmo conjunto de seções planares e diferentes valores para o parâmetro β . A figura 5.3a apresenta um conjunto de 6 seções planares. As figuras 5.3b, 5.3c, 5.3d, 5.3e e 5.3f apresentam sólidos finais com valores de β iguais a 0, 1, 2, 3, e 4 respectivamente.

Figura 5.3 – Comportamento do algoritmo β -Conexão com o mesmo conjunto de seções planares e diferentes valores para o parâmetro β .

(a) Conjunto de seções planares de exemplo, (b) Sólido final com $\beta = 0$, (c) Sólido final com $\beta = 1$,
(d) Sólido final com $\beta = 2$, (e) Sólido final com $\beta = 3$, (f) Sólido final com $\beta = 4$,

A figura 5.4 apresenta a importância da flexibilidade na escolha das componentes conexas. A figura 5.4a apresenta um conjunto de regiões contidas em uma sequência de seções paralelas. Pelo fato de que o processo de reconstrução é restrito a duas seções planares adjacentes, é possível especificar diferentes valores para β entre cada duas seções planares adjacentes. As figuras 5.4b, 5.4c e 5.4d apresentam diferentes modelos que podem ser obtidos escolhendo distintos valores para β entre cada par de seções planares. Note que a topologia dos modelos finais é completamente diferente em cada caso. Este fato faz o algoritmo β -Conexão uma poderosa ferramenta para modelar sólidos.

Figura 5.4 – Modelos finais gerados a partir de um mesmo conjunto de seções planares paralelas e diferentes valores de β entre seções planares.

(a) Seções planares de exemplo, (b, c, d) Figuras geradas com diferentes valores de β entre cada par de seções planares.

5.2 Contribuições

Além da contribuição do novo algoritmo β -Conexão em si, nesta seção apresentamos duas contribuições importantes: a descrição de como o pipeline de reconstrução pode ser utilizado (seção 5.2.1), e a descrição de uma ferramenta de visualização de estruturas de dados criada para ajudar na implementação deste trabalho.

5.2.1 Utilização do *pipeline* de reconstrução

O *pipeline* de reconstrução foi implementado em C++ *standard*. Desta forma pode ser utilizado sem muito problema em qualquer sistema operacional com um compilador de C++. Esta implementação também foi pensada para ser de fácil utilização. O exemplo de programa em C++ abaixo apresenta a forma de utilizar a implementação do *pipeline* de reconstrução.

```
#include "..\Rvw\GlobalBetaConnection.h"

int main(int argc, char *argv[])
{
    int nBetaParam = 1;
    CReconstructionPipeline* pRP = new CReconstructionPipeline(stdout);

    pRP->Init(
        "C:\\Temp\\Test1\\",           //Diretorio de entrada
        "x",                          //Nome do arquivo de entrada
        0, 2,                         //Faixa de niveis que se quer reconstruir
        "C:\\Temp\\Test1\\out");      //Diretorio de saida

    pRP->Execute(nBetaParam);         //Reconstrução com beta = 1

    delete pRP;
    return 0;
}
```

Os passos principais para usar o algoritmo são:

- Incluir o arquivo de definições de classes da reconstrução.
- Declarar e alocar um objeto da classe CReconstructionPipeline
- Inicializar o objeto, chamando o método **Init**.

Esta inicialização recebe:

- O diretório de entrada, onde estão contidos os arquivos que representam as seções planares do objeto que se quer reconstruir. Por exemplo, "x0.vtk", "x1.vtk", ..., "xn.vtk".
- O nome do arquivo. Por exemplo, se cada um dos arquivos de entrada tem como nome "xi.vtk" o nome do arquivo para este parâmetro é "x".

- A faixa de níveis que se pretende reconstruir.
- O diretório de saída onde será criado o sólido final.
- Executar o *pipeline* de reconstrução, chamando o método **Execute**
- Desalocar o objeto

5.2.2 Ferramenta de visualização de dados

Durante o desenvolvimento do algoritmo β -Conexão, houve muita dificuldade para imaginar e entender todos os processos que estavam sendo executados sob a malha de tetraedros na reconstrução. Por causa desta dificuldade, a depuração de erros chegou a ser demasiado complicada e lenta. Então nós resolvemos criar uma ferramenta que ajude a visualizar tudo o que estava acontecendo dentro da malha de tetraedros durante o processo de reconstrução. Esta ferramenta foi criada para ser incluída no meio do código e ser utilizada para fazer visualizações durante o tempo de depuração de erros.

Esta ferramenta foi desenvolvida paralelamente ao algoritmo β -Conexão conforme aos requerimentos de visualização para depuração, e demonstrou ser de muita utilidade facilitando e acelerando o desenvolvimento.

Com esta ferramenta é possível visualizar, entre outras coisas, grupos coloridos de objetos, objetos com transparência, tetraedros encolhidos, singularidades em vértice e em aresta, etc. A figura 5.5 apresenta várias imagens geradas em tempo de depuração.

Figura 5.5 – Visualizações em tempo de depuração.

- (a) Contornos, (b) Arestas classificadas, (c) Triangulação classificada, (d) Tetraedros encolhidos, (e) Wire frame, (f) Grafo associado, (g) Componentes conexas e (h) Singularidades.

5.3 Conclusões e trabalhos futuros

Este trabalho apresentou todo um processo de reconstrução tridimensional a partir de seções planares. O algoritmo de reconstrução utilizado neste processo é um algoritmo novo denominado β -Conexão. A nova heurística utilizada pelo algoritmo β -Conexão está baseada em um valor inteiro positivo β , que especifica uma distância mínima em tetraedros como condição para definir as conexões entre um conjunto de seções planares. Esta técnica oferece mais flexibilidade na escolha das componentes conexas, gerando uma família de objetos a partir de um mesmo conjunto de seções planares, fato dificilmente obtido por outros algoritmos da literatura. Esta técnica também resolve os problemas de ramificação e geração de malha de forma satisfatória.

O algoritmo foi implementado na linguagem C++ fazendo uso do VTK (Visualization Toolkit) [Sch98], [Kit01] para a visualização.

Este processo de reconstrução tridimensional começa com a entrada de um conjunto de contornos distribuídos em uma sequência de seções planares. Em geral, muitos algoritmos de reconstrução, citados na literatura, estão limitados por esta etapa prévia de segmentação para obter os contornos. Seria muito interessante, em um trabalho futuro, incluir o processo de segmentação dentro do processo da reconstrução. Desta forma a reconstrução poderia ser realizada diretamente a partir de uma sequência de imagens obtidas, por exemplo, por um aparelho de ressonância magnética.

É muito importante para processos de simulações numéricas de fenômenos físicos que o sólido reconstruído por esta nova técnica seja uma malha livre de singularidades. No entanto, em um trabalho futuro, seria válido acrescentar um processo para melhorar a qualidade da malha pelo nosso algoritmo.

O casamento da visualização volumétrica com a reconstrução 3D tem sido fonte de interesse em muitos grupos de pesquisa, sendo essa junção, mais um trabalho futuro apontado por nós.

Referências Bibliográficas

- [Ame97] Amneta, N.; Bern, M.; Kamvysselis, M. – “A new Voronoi-Based Surface Reconstruction Algorithm”, artigo disponível em: <http://web.mit.edu/manoli/crust/www/crust.html>, visitado em agosto de 2001.
- [Baj96] Bajaj, C.L.; Coyle, E.J.; Lin, K. – “Arbitrary topology shape reconstruction from planar cross sections”, *Graphical Models and Image Processing*, 58 (6), pp. 524-543 (1996).
- [Baj99] Bajaj, C.L.; Coyle, E.J.; Lin, K. – “Tetrahedral meshes from planar cross-sections”, *Comput. Methods Appl. Mech Engrg.*, 179, 31-52 (1999).
- [Bar96] C. B., Barber; D.P., Dobkin; H., Huhdanpaa – “The Quickhull Algorithm for Convex Hulls”, *ACM Transactions On Mathematical Software*, Vol.22, No. 4, pp. 469-483 (1996).
- [Bar99] Barber, C. B.; Huhdanpaa, H. – “qhull, rbox - Convex hull, Delaunay triangulation, Voronoi diagrams, Halfspace intersection”, *The Geometry Center University of Minnesota*, disponível em: <http://www.geom.umn.edu/software/qhull/>, visitado em agosto de 2001.
- [Bis00] Biscaro, H.H. – “Estrutura Half-Face”, *Dissertação de mestrado, Instituto de Ciências Matemáticas e de Computação - Universidade de São Paulo*, (2000).
- [Boi88] Boissonnat, J-D. – “Shape Reconstruction from Planar Cross Sections”, *Computer Vision, Graphics and Image Processing*, (44), 1-29 (1988).
- [CG87] ACM SIGGRAPH Computer Graphics, “Special Issue on Visualization in Scientific Computing”, vol. 21, no.6, (1987).
- [CGA01] “CGAL - Computational Geometry Algorithms Library”, disponível em: <http://www.cgal.org>, (2001).
- [Cor90] Cormen, T.H.; Leiserson, C.E.; Rivest, R.L. – “Introduction to Algorithms”, *The MIT Electrical Engineering and computer Science Series*, McGraw-Hill Edition (1990).
- [Del94] Delingette, H. – “Simplex Meshes: A General Representation for 3D Shape Reconstruction”, *Technical Report 2214, INRIA* (1994).

- [Dey00] Dey, T. K.; Giesen, J.; Hudson, J.; – "Delaunay based shape reconstruction ", Ohio State University, Columbus USA, (2000).
- [Ede94] H, Edelsbrunner; E.P., Mucke – "3-Dimensional Alpha-Shapes", ACM Transactions On Graphics, 13(1) 43-72 (1994).
- [Eko91] Ekoule, A.B.; Peyrin, F.C.; Odet, C. L. – "A Triangulation Algorithm from Arbitrary Shaped Multiple Planar Contours", ACM Trans. On Graph., 10(2), pp. 182-199 (1991).
- [For94] Fortune, S. – "Voronoi Diagrams and Delaunay Triangulation", in D.-Z. Du, F K. Hwang (eds.), Computing in Euclidean Geometry, World Scientific Pub. Co., 193-233 (1994).
- [Gei93] Geiger, B. – "Three Dimensional Modeling of Human Organs and its Application to Diagnosis and Surgical Planning", Technical Report 2105, INRIA Sophia-Antipolis France (1993).
- [Gei93b] Geiger, B. – "Nuages, a tool for 3D reconstruction from parallel cross-sections", INRIA Sophia-Antipolis France,(1993), disponível em: <http://www-sop.inria.fr/prisme/logiciel/nuages.html>, visitado em agosto de 2001.
- [Jon94] Jones, M. W.; Chen, M. – "A New Approach to the Construction of Surfaces from Contour Data", Proc. EUROGRAPHICS'94, 13(3), pp. 75-84 (1994).
- [Kau98] Kaufman, A. – "Advances in Volume Visualization", SIGGRAPH'98, Course Notes no. 24, Orlando (1998).
- [Kep75] Keppel, E. – "Approximating complex surface by triangulation of contour lines", IBM. J. Res. Dev. 19, pp. 2-11 (1975).
- [Kit01] Kitware, "The Visualization Toolkit", Object-Oriented Approach to 3D Graphics, disponível em: <http://public.kitware.com>, visitado em agosto de 2001.
- [Lam98] T. Lambert – "Convex Hull Algorithms", School of Computer Science and Engineering University of New South Wales, Sydney AUSTRALIA, (1998), disponível em: <http://www.cse.unsw.edu.au/~lambert/java/3d/hull.html>, visitado em agosto de 2001.
- [Lor87] W.E, Lorensen; H.E. Cline – "Marching cubes: A High Resolution 3D Surface Construction Algorithms", ACM SIG. Comp.Graph., 21,163-169 (1987).

- [Lu97] Lu, C. – “Robust Incremental Construction of Voronoi Diagrams (incremental algorithm)”, Computational Geometry and Graph Drawing, disponível em: <http://www.cs.brown.edu/courses/cs252/proj/src/Spr96-97/cl/delaunay.html>, visitado em setembro de 2000.
- [Luo97] Luo, C. – “Delaunay Triangulation and Voronoi Diagram (divide-and-conquer algorithm)”, Computational Geometry and Graph Drawing, disponível em: <http://www.cs.brown.edu/courses/cs252/proj/src/Spr96-97/chl/DelTri.html>, visitado em agosto de 2001.
- [Mey92] Meyers, D.; Skinner, S.; Sloan, K. – “Surface from Contours”, ACM Trans. On Graphics, 11(3), pp. 228-258 (1992).
- [Müc98] Mücke, E. – “Robust 3D Delaunay triangulations”, The Geometry Center-University of Minnesota, disponível em: <http://www.geom.umn.edu/software/cglist/GeomDir>, visitado em agosto de 2001.
- [Non00b] Nonato, L.G.; Minghim, R.; Shimabukuro, M. H. – “Qualitative Analysis of Two Reconstruction Techniques and their Application in Denstistry”, Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo, (2000).
- [Non01] Nonato, L.G.; Minghim, R.; Oliveira, M.C.F.; Tavares, G. – “A Novel Approach for Delaunay 3D Reconstruction with a Comparative Analysis in the Light of Applications”, Computer Graphics Forum, 20(2), 161-174, (2001).
- [Non98] Nonato, L.G. – “Reconstrução Volumétrica de Objetos definidos por seções Bidimensionais”, Tese de Doutorado, Departamento de Matemática, Pontifícia Universidade Católica, Rio de Janeiro, Brasil, (1998).
- [Pre85] Preparata F.P.; Shamos, M.I.; “Computational Geometry an Introduction”, Texts and Monographs in Computer Science, , 2nd Edition, (1985)
- [Sch98] Schröder, W.J.; Martin, K.; Lorensen, B. – “The Visualization Toolkit”, Object-Oriented Approach to 3D Graphics, 2nd Edition; Prentice-Hall, (1998).
- [Shi91] Shinagawa, Y.; Kunii, T.L. – “The homotopy model: a generalized model for smooth surface generation from cross sectional data”, The Visual Computer, 7, pp. 72-86 (1991).

- [Sin98] Singh, A.; Goldgof, D.; Terzopoulos, D. (ed.) – “Deformable Models in Medical Image Analysis”, IEEE CS Press (1998).
- [Soa00] Soares, I.P. – “Visualização Volumétrica em Odontologia”, Dissertação de mestrado, Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo, (2000).
- [Ung97] Ungor – “Delaunay Triangulation in Java (Edge Flip Algorithm)”, Department of Computer Science University of Illinois UC., disponível em: <http://www.cse.uiuc.edu/~ungor/delaunay/Delaunay.html>, visitado em agosto de 2001.
- [Zha96] Zhao, Z. – “Delaunay Triangulation & Voronoi Diagram (incremental algorithm)”, Center for Mapping, Ohio State University, disponível em: http://java.wiwi.uni-frankfurt.de:8080/java/owa/jr_qurres?vappletno=3144, visitado em agosto de 2001.

Apêndice A

Estrutura Half-Face

Este apêndice apresenta a estrutura de dados *Half-Face*, utilizada no processo de reconstrução tridimensional deste trabalho. Esta estrutura foi desenvolvida na linguagem C++ no projeto de mestrado de Biscaro [Bis00] do laboratório LCAD. A *Half-Face* representa volumes, enfatizando a velocidade na obtenção das relações de incidências e adjacências e também os custos de armazenamento. Esta estrutura registra também as singularidades do volume de forma explícita fazendo-a apropriada para o processo de reconstrução.

O apêndice começa descrevendo cada uma das classes da *Half-Face*, seguido por uma visão geral da mesma. Depois será falado sobre o armazenamento de informações, e por último, da representação de singularidades.

As classes da estrutura Half-Face

Seja V um objeto de dimensão 3 com bordo. A estrutura *Half-Face*, denotada por $\underline{HF}$, é uma estrutura formada por nós das seguintes classes:

- A classe Sólido, denotada por S .
- A classe Célula, denotada por C .
- A classe Vértice, denotada por V .
- A classe Face de bordo, denotada por ∂F .
- A classe Aresta singular, denotada por SE .
- A classe Vértices singulares, denotada por SV .
- A classe Semi-face, denotada por HF .
- A classe Semi-aresta, denotada por HE .
- A classe Estrela do vértice, denotada por $StarV$.

A figura A.1 mostra uma visão global das relações que existem entre estas classes. As classes Vértice e Célula foram criadas para formar parte de *arrays*. A classe Semi-aresta foi criada para ser parte de uma lista circular duplamente encadeada, pois este elemento é o responsável pela orientação do objeto. As classes restantes foram criadas para formar parte de listas lineares duplamente encadeadas. As classes que formam parte de listas possuem dois ponteiros a mais: *next* e *previous*. A seguir serão descritas as relações entre todas estas classes, omitindo estes ponteiros.

Figura A.1 – Visão geral da estrutura Half-Face .

A classe sólido

São cinco as relações que um nó desta classe possui (Figura A.2):

- S_V que liga o nó S ao nó $v_i \in V(\underline{HF})$
- $S_∂F$ que liga o nó S ao nó $∂f_i \in ∂F(\underline{HF})$
- S_C que liga o nó S ao nó $c_i \in C(\underline{HF})$
- S_SE que liga o nó S ao nó $se_i \in SE(\underline{HF})$
- S_SV que liga o nó S ao nó $sv_i \in SV(\underline{HF})$

Figura A.2 – Relações da classe Sólido.

A classe célula

Um nó da classe célula possui duas relações (Figura A.3).

- C_HF que liga o nó C ao nó $hf_l \in HF(\underline{HF})$
- C_S que liga o nó C ao nó $s_l \in S(\underline{HF})$

Figura A.3 – Relações da classe Célula.

A classe vértice

Também, são duas as relações de um nó da classe vértice (Figura A.4):

- V_StarV que liga o nó V ao nó $starv_l \in StarV(\underline{HF})$,
- V_S que liga o nó V ao nó $s_l \in S(\underline{HF})$

Figura A.4 – Relações do nó Vértice.

A classe face de bordo

Duas relações existem a partir de um nó desta classe (Figura A.5):

- ∂F_HF que liga o nó ∂F a $hf_l \in HF(\underline{HF})$;
- ∂F_S que liga o nó ∂F a $s_l \in S(\underline{HF})$;

Figura A.5 – Relações do nó Faces de Bordo.

A classe semi-face

São três as relações que possui um nó da classe semi-face (Figura A.6):

- HF_Mate que liga o nó HF ao $mate \in HF(\underline{HF})$, isto é a semi-face que está “colada” na semi-face em questão. Note que se a face for de bordo, este nó é nulo
- HF_HE que liga o nó HF ao nó $he_i \in HE(\underline{HF})$
- HF_C que liga o nó HF ao nó $c \in C(\underline{HF})$

Figura A.6 – Relações da classe Semi-Face.

Mate de uma semi-face nada mais é do que a semi-face que está “colada” à semi-face. Como já foi comentado, se a face em questão for uma face de bordo, não existe um *mate* para sua semi-face. É assim, pois, que esta estrutura identifica quando uma face é de bordo ou não.

A classe semi-aresta.

Quatro relações existem a partir de um nó da classe semi-aresta (Figura A.7), sendo que este é um nó extremamente importante, pois é ele o responsável pela orientação do objeto representado.

- HE_V que liga o nó HE ao nó $v \in V(\underline{HF})$.
- HE_Mate-aresta que liga o nó HE a *Mate-aresta* $\in HE(\underline{HF})$ na mesma célula.
- HE_Mate-célula que liga o nó HE a *Mate-face* $\in HE(\underline{HF})$ na célula adjacente, ou ao nulo no caso do bordo.
- HE_HF que liga o nó HE ao nó $hf_i \in HF(\underline{HF})$.

Figura A.7 – Relações da classe Semi-Aresta.

Temos aqui dois elementos que necessitam de maiores esclarecimentos, são eles o mate-aresta e o mate-face. Na verdade, são bastante simples de entender. O mate-célula é a semi-aresta que compartilha a mesma aresta na mesma célula, e o mate-aresta, refere-se à semi-aresta que compartilha a mesma aresta, mas em uma célula vizinha.

A classe Estrela do vértice

Um nó da classe estrela do vértice possui apenas uma relação partindo dele e ligando-o ao conjunto de semi-arestas que compõe a estrela de um vértice (Figura A.8). É através deste nó que se faz a identificação dos vértices singulares.

- A linha StarV_HE que liga o nó *StarV* ao nó $he \in HE(\underline{HF})$

Figura A.8 – Relações da classe Estrela do Vértice.

A classe vértices singulares

Elemento-chave nesta estrutura de dados, um nó da classe vértices singulares possui duas linhas dirigidas conectadas a ele (Figura A.9).

- A linha SV_V que liga o nó *SV* ao nó $v \in V(\underline{HF})$
- A linha SV_S que liga o nó *SV* ao nó $s \in S(\underline{HF})$

Figura A.9 – Relações da classe Vértice Singular.

A classe arestas singulares

Outro elemento chave nesta estrutura de dados, possui também duas relações (Figura A.10).

- SE_HE que liga o nó *SE* ao nó *he* $\in$ HE(HF);
- SE_S que liga o nó *SE* ao nó *s* $\in$ S(HF);

Figura A.10 – Relações da classe Arestas Singulares.

Observe que nesta estrutura não é representada explicitamente nem faces nem arestas, com o objetivo de reduzir os custos de armazenamento. No entanto, os elementos vértices singulares e arestas singulares foram incluídos apenas por motivos de velocidade de acesso, pois estas informações poderiam ser obtidas percorrendo as listas de vértices e semi-arestas.

Armazenamento de informações

É extremamente importante para o processo de reconstrução, que seja possível armazenar informações variadas na estrutura. A estrutura Half-face, que foi desenvolvida em C++, possui em cada classe um campo do tipo *void **. Este campo permite armazenar qualquer tipo de informação tais como: Variáveis de orientação, ponteiros, Temperatura, Energia, etc. Este fato aumenta bastante a versatilidade da estrutura.

No desenvolvimento do algoritmo reconstrução tridimensional deste trabalho foi necessário armazenar, para vários processos, diferentes tipos de dados na maioria das classes.

Representação de singularidades

A principal característica desta estrutura de dados é a representação de vértices e arestas singulares. Quando vértices ou arestas singulares aparecerem no volume, estes são registrados por nós das classes Vértice Singular e Aresta Singular, respectivamente. A figura A.11 ilustra exemplos de vértices e arestas singulares.

Esta característica foi também de muita utilidade no processo de reconstrução. De fato, verificando que não existam singularidades na estrutura Half-Face, pode-se garantir a segunda hipótese de reconstrução (que o objeto reconstruído seja uma variedade).

Figura A.11 – Exemplos de: (a) Vértice singular e (b) Aresta singular.

Apêndice B

Definições básicas de Topologia Algébrica

Neste apêndice serão apresentadas algumas definições básicas sobre topologia algébrica que servirão como base para os conceitos apresentados neste trabalho.

Começaremos comentando sobre o fecho convexo de um conjunto de pontos, seguido pela definição de um conjunto de pontos linearmente independentes. Depois serão apresentadas as definições de simplexos, complexos, estrela, link, variedades LP e orientação coerente em complexos.

Nonato [Non98] (pg. 7-17) apresenta também algumas definições de topologia algébrica e Biscaro [Bis00](pg. 36) faz uma descrição de singularidades.

Complexos Simpliciais

Sejam P_0 e P_1 dois pontos distintos. O fecho convexo de P_0 e P_1 é o conjunto dos pontos definido por:

$$\langle P_0, P_1 \rangle = \{ \lambda P_0 + (1-\lambda)P_1 \mid 0 \leq \lambda \leq 1 \}$$

Isto é, no conjunto $\langle P_0, P_1 \rangle$ com $\lambda = 0$ tem-se o ponto P_1 , com $\lambda = 1$ se tem o ponto P_0 e com λ entre 0 e 1 se tem o conjunto de pontos da reta que liga aos pontos P_0 e P_1 (Figura B1a).

O fecho convexo de três pontos P_0, P_1, P_2 (Figura 2.1b) é o conjunto dado por:

$$\langle P_0, P_1, P_2 \rangle = \{ (\lambda P_0 + (1-\lambda)P_1)M + (1-M)P_2 \mid 0 \leq \lambda, M \leq 1 \}$$

Se substituirmos $(\lambda P_0 + (1-\lambda)P_1)$ por q tem-se:

$$\langle P_0, P_1, P_2 \rangle = \{ Mq + (1-M)P_2 \mid q \in \langle P_0, P_1 \rangle, 0 \leq M \leq 1 \}$$

Isto é, no conjunto $\langle P_0, P_1, P_2 \rangle$ com $M = 0$ tem-se o ponto P_2 , com $M = 1$ tem-se o conjunto de pontos da reta q e para um M entre 0 e 1 tem-se o conjunto de pontos das retas que ligam P_2 à reta q (Figura B1b).

No caso geral temos que o **fecho convexo** para $d + 1$ pontos é dado por:

$$\langle P_0, \dots, P_d \rangle = \{ Mq + (1-M)P_d \mid q \in \langle P_0, \dots, P_{d-1} \rangle, 0 \leq M \leq 1 \} \quad (1)$$

Isto é, no conjunto $\langle P_0, \dots, P_d \rangle$ com $M = 0$ tem-se o ponto P_d , com $M = 1$ se tem o conjunto de pontos do conjunto q e para um M entre 0 e 1 tem-se o conjunto de pontos que ligam P_d ao conjunto q . A figura B.1c mostra o fecho convexo para quatro pontos.

Figura B.1 – Fechos convexas de: (a) 2 pontos, (b) 3 pontos e (c) 4 pontos.

Dizemos que $d+1$ pontos em $\mathbb{R}^d$ são **linearmente independentes** se nenhum dos pontos está contido no fecho convexo dos outros. Se a equação (1) for escrita como:

$$\langle P_0, \dots, P_d \rangle = \left\{ \sum_{k=0}^d \lambda_k P_k \mid \lambda_k \geq 0 \text{ e } \sum \lambda_k = 1 \right\}$$

os λ_k são chamados coordenadas baricentricas do ponto.

Qualquer ponto dentro do fecho convexo pode ser expresso como:

$$p = \sum_{k=0}^d \lambda_k P_k$$

se os pontos P_k são linearmente independentes então pode-se mostrar que as coordenadas baricênticas dos pontos em $\langle P_0, \dots, P_d \rangle$ são únicas.

Um **d-simplexo** é o fecho convexo de $d+1$ pontos linearmente independentes e d é a dimensão do simplexo. Por exemplo:

- Um 0-simplexo é um ponto
- Um 1-simplexo é um aresta
- Um 2-simplexo é um triângulo
- Um 3-simplexo é um tetraedro

O **bordo de um d-simplexo** S consiste de todos os $(d-k)$ -simplexo contidos em S , onde $0 < k \leq d$ e é denotado por ∂S .

Todo simplexo no bordo de S é chamado face de S . Um k -simplexo no bordo é denominado k -face.

O número de faces que contém um d -simplexo é: $\binom{d+1}{k+1}$

Observação: Quaisquer dois k -simplexos são homeomorfos.

Um **complexo simplicial** C é um conjunto finito de simplexos satisfazendo as seguintes propriedades:

1. Se S um simplexo e S' uma das faces de S , então S' está contida em C .
2. Se S_1 e S_2 dois simplexos de C , então a interseção de S_1 com S_2 ou é vazia ou é uma face comum de S_1 e S_2 .

A dimensão de um complexo simplicial C é a dimensão máxima de seus simplexos.

A **Estrela** de um simplexo S ($\text{star}(S)$) contido em um complexo simplicial C é o conjunto formado pela união de todos os simplexos de C que contém S , ou seja:

$$\text{Star}(S) = \{ \sigma \in C \mid S \text{ é face de } \sigma \}$$

A figura B2 mostra exemplos de estrelas para um vértice e para uma aresta.

Figura B2 – Estrela de: (a) um ponto, (b) uma aresta.

O **Link** de um simplexo S ($\text{link}(S)$) é a união de todos os simplexos que estão na estrela de S e que não intersectam S .

A figura B3 mostra exemplos de links para um vértice e para uma aresta.

Figura B3 – Link de: (a) um ponto, (b) uma aresta.

Uma **Variedade linear por partes (Variedade LP)** de dimensão 2 sem bordo é um complexo simplicial C de dimensão 2 tal que:

1. Todo 1-simplexo de C é incidente a exactamente dois 2-simplexos
2. O link de todo 0-simplexo em C é homeomorfo a um círculo

Uma **3-variedade LP com bordo** é um complexo simplicial de dimensão 3 tal que:

1. Todo 2-simplexo é adjacente a um ou dois 3-simplexos.
2. O link de todo 0-simplexo é homeomorfo a um disco ou a uma esfera.

Observação 1: O link dos vértices que estão no bordo é homeomorfo a um disco e o link de vértices interiores é homeomorfo a uma esfera.

Observação 2: O bordo de uma variedade LP de dimensão 3 com bordo é uma variedade LP de dimensão 2 sem bordo.

A figura B4 mostra um exemplo de um objeto que é uma 3-variedade LP com bordo (na figura B4b as tampas do objeto foram tiradas para uma melhor visualização).

Figura B4 – (a) 3-Variedade LP com Bordo e (b) Bordo (sem tampas) do objeto da figura a.

Observação: Uma orientação para um 2-simplexo induz uma orientação para os 1-simplexos.

Sejam S e S' dois 2-simplexos adjacentes a uma aresta em um complexo C . S e S' possuem **orientação coerente** se a aresta adjacente possui orientação oposta quando percorrida em S e S' . A figura B5 apresenta um 2-complexo com orientação coerente.

Figura B5 – Conjunto de 2-simplexos com orientação coerente.

A orientação no interior de uma 3-variedade é induzida pela orientação no bordo.