UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Sidney Ferreira de Lima

Estudos de métodos de esqueletização de formas 3D

Uberlândia, Brasil

2017

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Sidney Ferreira de Lima

Estudos de métodos de esqueletização de formas 3D

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Ciência da Computação.

Orientador: André Ricardo Backes

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2017

Sidney Ferreira de Lima

Estudos de métodos de esqueletização de formas 3D

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Ciência da Computação.

Trabalho aprovado. Uberlândia, Brasil, 02 de março de 2017:

André Ricardo Backes
Orientador

Professor

Professor

Uberlândia, Brasil
2017

Dedico este trabalho à toda a minha família, em especial para as minhas sobrinhas Júlia, Ana Luíza e Maria Isadora.

Agradecimentos

Agradeço primeiramente ao Criador por principalmente ter concedido o dom da vida, toda a realidade e todo o conhecimento, condições e oportunidades para que este trabalho possa se tornar possível, graças à Sua vontade. Agradeço a toda a minha família pelo apoio, conselhos e por estarem juntos em cada etapa desta caminhada, em especial à minha mãe, Maria Lúcia, por todo seu amor, dedicação e por ser o alicerce de minha estrutura familiar. Agradeço as minhas sobrinhas Júlia, Maria Isadora e Ana Luíza por me trazerem alegria em momentos difíceis.

Agradeço aos meus grandes amigos Eduardo, Michael e Johnatan pela motivação e pelo período de estudos para o processo seletivo da universidade, pois sem eles não estaria no curso de graduação. Agradeço ao professor orientador André Backes pelo tema, a oportunidade de trabalhar em equipe para o desenvolvimento deste projeto e por me trazer esperança em diversos momentos no decorrer do desenvolvimento do projeto.

"Se não puder voar, corra. Se não puder correr, ande. Se não puder andar, rasteje, mas continue em frente de qualquer jeito."

Martin Luther King Jr

Resumo

Este projeto apresenta um estudo sobre métodos de esqueletização de objetos tridimensionais. O processo de esqueletização busca extrair de um objeto 3D o seu esqueleto, que pode ser definido como uma estrutura curva semelhante a um grafo e capaz de descrever a topologia deste objeto de maneira simples e compacta. Esta estrutura possui diversas aplicações como animação na área de computação gráfica, comparação entre objetos com o objetivo de medir similaridade, segmentação de objetos e até mesmo reconstituição da superfície do objeto 3D.

Palavras-chave: Objetos 3D, esqueletização, Descritor de Formas, Malhas, Nuvem de pontos.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

2D	Bidimensional ou duas dimensões
3D	Tridimensional ou três dimensões
CAD	*Computer-Aided Design*
CAE	*Computer-Aided Engineering*
CAM	*Computer-Aided Manufacturing*

Sumário

1 Introdução

1.1 Visão Geral

A modelagem e processamento de malhas e superfícies tridimensionais possui diversas aplicações graças à sua capacidade de modelar fielmente objetos do mundo real. Juntamente com a potencial capacidade computacional presente em computadores atuais e com ferramentas de modelagem mais avançadas, modelos 3D podem ser facilmente criados e manipulados sem a necessidade de equipamentos avançados. Muitos modelos profissionais relacionados a várias áreas (como na Biologia, Medicina, Química ou Robótica) estão disponíveis em volume crescente através da Internet (YANG; LIN; ZHANG, 2007). Google 3D Warehouse [1] e TurboSquid [2] são exemplos de sites contendo modelos profissionais de vários objetos (OVSJANIKOV et al., 2011).

Várias são as aplicações em que objetos 3D podem ser utilizados de modo geral. Aplicações CAD, CAE e CAM tem auxiliado profissionais como engenheiros, arquitetos, designers na modelagem 2D e 3D de produtos nos diversos setores, dentre eles na indústria automotiva, arquitetura, construção naval (MODERNA, 2016). Com o uso de impressoras 3D é possível através de modelos tridimensionais produzir de modelos de brinquedos a protótipos complexos e próteses médicas (TECMUNDO, 2013). O processo de digitalização 3D (o uso de digitalizadores para obter modelos tridimensionais através de objetos do mundo real) tem demonstrado eficácia no desenvolvimento de produtos industriais, tendo como exemplo de aplicação a etapa de inspeção 3D no final do desenvolvimento, na qual consegue-se comparar o modelo desenvolvido em CAD com o modelo digitalizado do produto final (ENGENHARIA, 2012). O trabalho de Levoy et al. (2000) descreve um sistema para digitalização no qual foi utilizado para escanear dez estátuas de Michelangelo.

Algumas aplicações utilizam apenas informações extraídas de um objeto 3D em uma representação simples e compacta, chamada de **descritor de formas**. Esta representação possibilita economia em armazenamento e maior desempenho computacional para os algoritmos que a utiliza. Também é desejado que esta informação não seja sensível a variações do modelo 3D original, tais como escala, rotação e translação (PAPADAKIS et al., 2008). O tipo de representação pode afetar a performance ou a precisão de um algoritmo. No caso de uma aplicação de recuperação de objetos 3D com base no conteúdo, a escolha de um descritor de formas afeta como será realizada a etapa de comparação entre dois objetos pelo fato desta etapa depender de como o descritor de formas está representado (YANG; LIN; ZHANG, 2007).

[1] Link para Google 3D Warehouse: https://3dwarehouse.sketchup.com/
[2] Link para TurboSquid: http://www.turbosquid.com/

O processo para extrair informações relevantes de um objeto 3D e representá-lo por meio de um descritor de formas é denominado **extração de características**. Os métodos para extração de características podem ser classificados em quatro categorias, sendo estes (YANG; LIN; ZHANG, 2007):

1. **Métodos baseados na análise da geometria global**: estes métodos analisam diretamente características da geometria global de um objeto 3D. Estes métodos necessitam previamente de algoritmos semelhantes à analise de componentes principais (PCA ou ACP) para normalizar a forma, além de possuírem alto custo computacional e de armazenamento. Conforme apresentado por Kolonias et al. (2005), a razão do aspecto (razão entre largura e altura da caixa de contorno do objeto 3D (FLORES, 2012)) e ângulos tridimensionais de vértices ou arestas são alguns dos exemplos de descritores mais simples e diretos, porém limitados ao caracterizar a forma. Outro exemplo é apresentado por Vranic (2004) ao propor uma representação geométrica baseada no raio, onde são definidos vetores de distância da origem até a superfície da malha. O descritor proposto por Suzuki, Kato e Otsu (2000) se baseia em "classes de equivalência". Nesta abordagem o objeto 3D é particionado em uma grade onde, para cada célula da grade, calcula-se o numero de vértices. Cada conjunto de células que contém características em comum (como número de vértices) define uma classe de equivalência. Depois constrói um vetor de características com propriedades da classe de equivalência (por exemplo o número de vértices) (Figura 1).

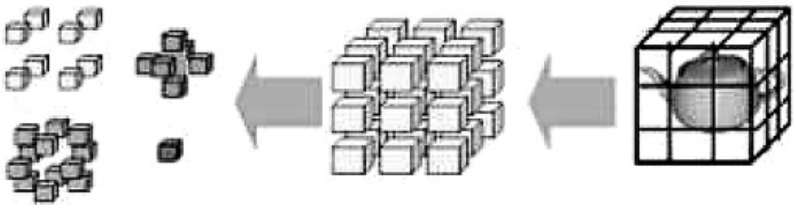

Figura 1 – Método baseado em "classes de equivalência"
Fonte: Suzuki, Yaginuma e Sugimoto (2003)

2. **Métodos baseados em funções de mapeamento**: estes métodos definem uma função de mapeamento entre o objeto 3D e um determinado domínio. Tal abordagem tem como vantagem menor complexidade computacional e em armazenamento, resultando em um descritor mais compacto. Todavia, resulta em muita perda de informação do objeto 3D original devido à restrição do processo de mapeamento. A forma do objeto 3D pode, por exemplo, ser mapeada em uma série de funções esféricas, não necessitando de normalização das coordenadas canônicas para gerar características invariantes. Métodos neste domínio supõe, de forma geral, que o modelo possua topologia válida (malhas) ou volume explícito (volumétricos), além de

ser um processo complexo e com alto custo computacional. Horn (1984) propõe um mapeamento que relaciona um ponto da superfície do objeto 3D com o ponto na esfera gaussiana por busca de pares de pontos que possuem mesmo vetor normal. Outra abordagem procura mapear projeções de um objeto 3D em diversas visões em planos 2D. Alguns exemplos são a proposta de (VRANIC, 2004), onde as características são extraídas por projetar o objeto 3D nos planos XY, XZ e YZ e calcular a densidade espectral de Fourier dos pontos igualmente espaçados em distância e ângulo do contorno para cada projeção.

3. **Métodos baseados nas propriedades estatísticas**: estes métodos utilizam propriedades estatísticas dos objetos 3D. Dentre estas temos os momentos (CANTERAKIS, 1999) e os histogramas (ASHBROOK et al., 1995).

4. **Métodos baseados em análise topológica**: estes métodos fazem uso da representação topológica do objeto. Tal representação possui uma estrutura semelhante a um grafo, o qual contém informações relacionadas à topologia, isto é, como os vértices, arestas e superfícies estão interligadas (YANG; LIN; ZHANG, 2007). Exemplo de representação é o esqueleto do objeto 3D (Figura 2).

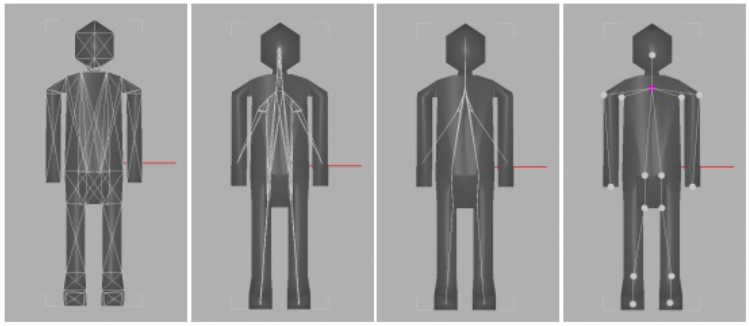

Figura 2 – Exemplo de representação pelo esqueleto de um objeto 3D
Fonte: Madaras (2013)

1.2 Objetivo

Este trabalho tem como objetivo apresentar um estudo sobre algoritmos de esqueletização de objetos tridimensionais, focando nos métodos utilizados para a extração do esqueleto de objetos 3D representados por **nuvem de pontos** ou **malhas triangulares**. Neste trabalho é realizada a análise das duas publicações mais referenciadas na literatura: a publicação de Tagliasacchi, Zhang e Cohen-Or (2009) para nuvem de pontos e a publicação de Au et al. (2008) para malhas triangulares. Além disto, este trabalho também apresenta o método de esqueletização proposto por Jiang et al. (2013) e um algoritmo para simplificação de malhas triangulares proposto por Valette e Chassery (2004). Em cada análise, este trabalho propõe mostrar quais as vantagens e desvantagens encontra-

das em cada método e buscar apresentar meios de solucionar alguma das desvantagens encontradas.

1.3 Justificativas

1.3.1 Motivos para utilizar o esqueleto como descritor de formas

O esqueleto de um objeto 3D é uma representação intuitiva e natural devido ao seu nível mais elevado de representação e proximidade a nossa visualização (YANG; LIN; ZHANG, 2007). Métodos que fazem uso do esqueleto do objeto 3D como descritor de formas possuem vantagens em relação a métodos baseados no contorno ou superfície do objeto devido ao esqueleto não ser tão sensível à variabilidades estruturais ou de articulação (YANG et al., 2007), isto é, alterações sutis no contorno ou superfície da forma de um objeto não impactuam diretamente na sua topologia.

1.3.2 Aplicações práticas

Devido à sua importância e variedade de aplicações para objetos 3D, existe uma quantidade crescente de modelos tridimensionais em base de dados disponíveis na Internet. Exemplos de repositórios são National Design Repository [3] para modelos CAD e Protein Data Bank [4] para modelos biológicos (TANGELDER; VELTKAMP, 2008). Com este cenário, surge a necessidade de motores de busca para objetos 3D, os quais ainda estão em sua fase inicial. A maioria dos sites permitem a busca apenas através de palavras-chave. Tais abordagens não possuem muita eficácia, podendo gerar ambiguidades. Além disto, é dificil representar características do modelo desejado através de uma consulta de texto. Para estas questões é considerável uma abordagem utilizando recuperação com base em conteúdo (YANG; LIN; ZHANG, 2007).

Recuperação de objetos 3D com base no conteúdo é uma abordagem que baseia-se nos atributos do objeto 3D (como a forma, cor, textura) para recuperação, reconhecimento e combinação de modelos tridimensionais. Estes conceitos não estão limitados a motores de busca, podendo ser aplicados em computação visual, computação gráfica, modelagem geométrica, reconhecimento de padrões, biologia molecular e assim por diante (YANG; LIN; ZHANG, 2007).

Em todo processo de recuperação baseada em conteúdo encontra-se os passos de extração de características e comparação. A extração de características consiste em obter atributos descritivos do objeto de consulta, tais como forma, cor, textura e outros;

[3] Link para Natural Design Repository: http://edge.cs.drexel.edu/repository/
[4] Link para Protein Data Bank: http://www.rcsb.org

enquanto a comparação é dada por obter atributos extraídos de dois objetos distintos e medir sua semelhança (YANG; LIN; ZHANG, 2007).

Yang, Lin e Zhang (2007) apresentam um estudo sobre recuperação com base em conteúdo de objetos tridimensionais (Figura 3). Segundo os autores, existem quatro etapas principais que geralmente serão encontradas neste tipo de sistema:

1. **Normalização**: consiste em normalizar um objeto 3D quanto à variações de escala, translação, rotação e reflexão. O objetivo é assegurar que as variações nos objetos 3D não interfiram nas etapas de extração de características e comparação, atribuindo um padrão destas variações a todos os objetos.

2. **Extração de características**: processo no qual são utilizados métodos para extrair informações necessárias para a etapa de comparação entre objetos. Tais métodos retornam uma representação do objeto capaz de descrevê-lo de forma mais simples e compacta. Tal representação pode ser determinada como **descritor de formas**, segundo Papadakis et al. (2008).

3. **Comparação**: tem como objetivo medir a similariade entre objetos tridimensionais utilizando a representação resultante do processo de extração de características, retornando os resultados em ordem de acordo com o grau de similaridade com cada modelo comparado.

4. **Representação das consultas e interface com o usuário** mecanismo no qual o usuário poderá realizar buscas por modelos tridimensionais com base no conteúdo.

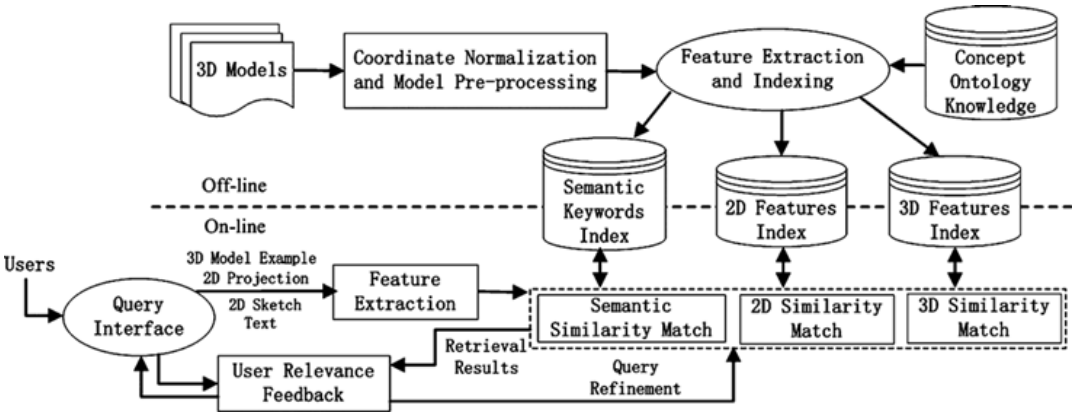

Figura 3 – Arquitetura de um sistema de recuperação de objetos 3D baseado em conteúdo
Fonte: Yang, Lin e Zhang (2007)

2 Metodologia

Este trabalho tem como objetivo apresentar um estudo sobre métodos de esqueletização de objetos 3D. Apesar de existirem diversas formas de representar um objeto 3D, este projeto possui foco em abordagens que utilizam **nuvem de pontos** e **malhas triangulares** como representação do objeto tridimensional de entrada. Outras abordagens utilizam **voxels** como representação do objeto 3D de entrada mas não são tratadas neste trabalho.

Muitos algoritmos utilizados no processamento de objetos tridimensionais utilizam cálculos relacionados à geometria analítica, álgebra linear, estatística ou cálculo diferencial e integral. Para que seja possível implementar métodos para realizar estes cálculos, é preciso utilizar os algoritmos de cálculo numérico correspondentes. Entretanto, algoritmos utilizados em cálculo numérico, ao serem implementados, podem produzir erros em seu resultado se estes forem comparados com aqueles retornados utilizando uma abordagem analítica (GUIDI, 2014). Portanto é preciso um ambiente confiável para que cálculos mais complexos possam ser efetuados.

Para efetuar cálculos complexos, geralmente utiliza-se a linguagem e ambiente de desenvolvimento MATLAB [1] . Através dela, é possível desenvolver algoritmos envolvendo cálculos utilizando funcionalidades fornecidas pela linguagem. Além disso, é possível realizar plotagens gráficas em até três dimensões. A linguagem possui alto nível de abstração, o que agiliza o processo de desenvolvimento pela simplicidade. Isto é uma vantagem ao ser comparada com linguagens de nível mais baixo de abstração, como a linguagem C.

Apesar de sua simplicidade, a linguagem MATLAB não possui um alto desempenho. Aruoba e Fernández-Villaverde (2014) fizeram um estudo de comparação entre as linguagens C++, Fortran, Java, Julia, Python, Matlab, Mathematica e R. Segundo o autor, um programa em Matlab possui tempo de execução em torno de 9 a 11 vezes mais lento que um programa em C++. Embora este tempo seja bem melhor comparado a um programa escrito em Python (45 vezes mais lento que C++) ou escrito em R (475 a 491 vezes mais lento que C++) (ARUOBA; FERNáNDEZ-VILLAVERDE, 2014), seria melhor se fosse possível desenvolver de modo mais eficiente em um ambiente mais confiável para cálculos numéricos.

A linguagem MATLAB apresenta técnicas para optimizar o tempo de execução pelo modo que um programa é escrito (MATHWORKS, 2016e). Conforme MathWorks (2016g), a linguagem é otimizada quando realiza operações envolvendo matrizes e vetores ao invés de comandos para iterações como *for* ou *while*. Além disto, felizmente,

[1] Link para mais informações sobre MATLAB: http://www.mathworks.com/products/matlab/

este ambiente disponibiliza uma API denominada **MEX**, no qual é possível programar funções em MATLAB utilizando linguagem C ou C++ (MATHWORKS, 2016b). Desta forma é possível desenvolver funcionalidades em uma linguagem com maior desempenho computacional que o MATLAB (ARUOBA; FERNáNDEZ-VILLAVERDE, 2014).

A vantagem de desenvolver utilizando **MEX** é a possibilidade de invocar métodos da linguagem MATLAB (tais como `svd()` para o cálculo da decomposição por valores singulares) pelo programa escrito em C através da função `mexCallMATLABWithTrap` fornecida por esta API(MATHWORKS, 2016c). No entanto, a chamada de funções da linguagem MATLAB através da API MEX pode reduzir o desempenho do programa (ALTMAN, 2014). Uma solução para isto seria desenvolver em C os métodos numéricos necessários, os quais podem ser encontrados no livro Numerical Recipes [2] .

Neste trabalho é utilizado o ambiente de desenvolvimento MATLAB para plotagem gráfica dos modelos tridimensionais e cálculos numéricos. Os algoritmos são desenvolvidos utilizado linguagem C através da API MEX, a qual utiliza a função `mexCallMATLABWithTrap` desta API para chamar funções nativas do MATLAB com tratamento de exceções. Os objetos 3D de entrada são obtidos através de arquivos Wavefront OBJ, os quais contém informações de vértices e faces do objeto (BOURKE, 2016).

[2] Link para Numerical Recipes: http://numerical.recipes/

3 Fundamentação Teórica

3.1 Representação de objetos 3D

Para realizar adequadamente o processamento digital tanto para imagens 2D quanto para objetos 3D, é necessário o conhecimento de como esta imagem é representada e, portanto, como estes dados estão estruturados. Esta seção apresentará as principais formas de representar objetos tridimensionais.

3.1.1 Nuvem de pontos

Uma nuvem de pontos (Figura 4) é um conjunto de pontos multidimensionais. No caso de uma nuvem de pontos 3D, trata-se de pontos tridimensionais, onde cada um destes representam os eixos de coordenadas X, Y e Z (POINTCLOUD.ORG, 2011).

Equipamentos para escaneamento a laser 3D, tais como LS3D [1] utilizam esta representação para os objetos capturados (FRANçA, 2013).

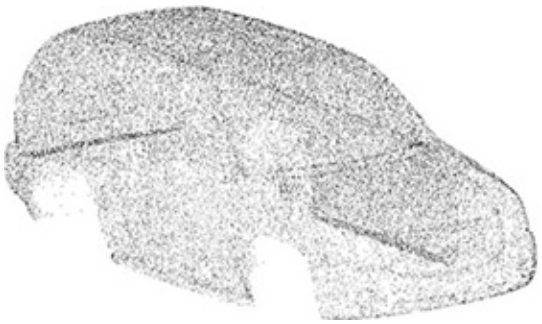

Figura 4 – Exemplo de representação de objeto 3D por nuvem de pontos
Fonte: ShapeQuest (1999)

3.1.2 Malha triangular

Segundo Botsch et al. (2006), uma **malha triangular** $\mathcal{M}$ (Figura 5) é um tipo de representação da superfície de um objeto 3D o qual é constituído de componentes geométricos e topológicos. De acordo com o mesmo autor, cada componente é definida da seguinte forma:

1. **Topologia**: consiste de um conjunto de vértices $\mathcal{V} = \{v_1, \ldots, v_n\}$ e um conjunto de faces triangulares $\mathcal{F} = \{f_1, \ldots, f_m\}$. Cada face $f_i \in \mathcal{V}^3$ é uma 3-tupla contendo três vértices de $\mathcal{V}$ distintos entre si. Para representar a conectividade entre os vértices,

[1] Link para LS3D - Laser Scanner 3D: http://www.ss-ls3d.com.br/index.html

pode-se definir um conjunto de arestas $\mathcal{E} = \{f_1, \ldots, f_e\}$ da malha, onde cada aresta $f_i \in \mathcal{V}^2$ é uma 2-tupla contendo o vértice de origem e de destino.

2. **Geometria**: consiste em um conjunto de posições do vértice no espaço tridimensional $\mathcal{P} = \{p_1, \ldots, p_n\}$, onde cada $p_i = (x_{v_i}, y_{v_i}, z_{v_i}) \in \mathbb{R}^3$ corresponde à posição do vértice $v_i \in \mathcal{V}$ no espaço.

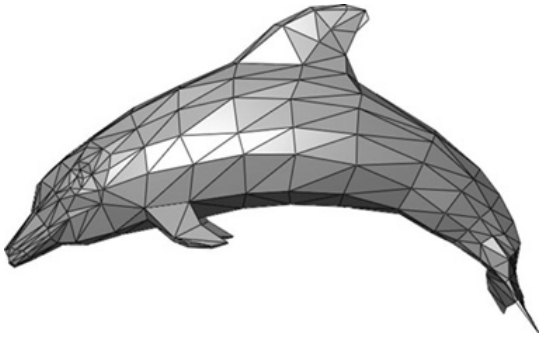

Figura 5 – Exemplo de representação de objeto 3D por malhas triangulares
Fonte: Power (2012)

A estrutura de dados para uma malha triangular pode ser construída utilizando uma tabela para a posição dos vértices no espaço e uma tabela de faces, onde cada face armazena o índice de cada um dos três vértices que a compõe (Tabela 1).

Tabela de vértices	
$\mathbf{V_1}$	x_1, y_1, z_1
$\mathbf{V_2}$	x_2, y_2, z_2
$\mathbf{V_3}$	x_3, y_3, z_3
$\mathbf{V_4}$	x_4, y_4, z_4
$\mathbf{V_5}$	x_5, y_5, z_5

(a) Tabela da Vértices

Tabela de faces	
$\mathbf{F_1}$	V_1, V_2, V_3
$\mathbf{F_2}$	V_2, V_4, V_3
$\mathbf{F_3}$	V_2, V_5, V_4

(b) Tabela da Faces

Tabela 1 – Representação de malhas por tabelas de vértices e faces

3.1.3 Representação volumétrica ou por voxels

Um **voxel** (Figura 6) constitui de uma unidade (ou célula) em uma grade uniforme, sendo análogo ao conceito de **pixel** em imagens 2D (FUNKHOUSER, 2002). Para uma imagem 3D binária, cada voxel podem ser representados por apenas um bit indicando se pertence ou não ao objeto 3D (XIE; THOMPSON; PERUCCHIO, 2003).

Segundo Xie, Thompson e Perucchio (2003), objetos 3D representado por voxels são armazenados em uma matriz tridimensional. Outras abordagens utilizam uma octree no lugar de uma matriz para a representação (LAINE; KARRAS, 2010).

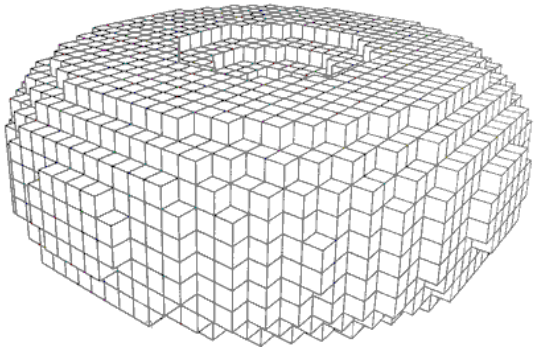

Figura 6 – Exemplo de representação de objeto 3D por voxels
Fonte: Funkhouser (2002)

3.1.4 Caixa delimitadora (bounding box)

A **caixa delimitadora** ou ***bounding box*** (Figura 7) é uma estrutura em formato de caixa que define os limites do objeto 3D nos eixos X, Y e Z (ERSOY, 2010).

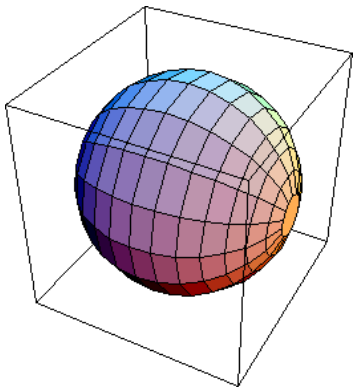

Figura 7 – Exemplo de caixa delimitadora ou bounding box
Fonte: Ersoy (2010) adaptada pelo autor

3.2 Conceitos de geometria analítica

3.2.1 Vetores

Segundo Venturi (2015), define-se vetor como "um conjunto de todos os segmentos orientados de mesma direção, de mesmo sentido e de mesmo comprimento". Dado dois pontos A e B, determina-se um vetor $\vec{v}$ de origem no ponto A e destino no ponto B como:

$$\vec{v} = B - A \tag{3.1}$$

Seja $\vec{i} = (1, 0, 0)$, $\vec{j} = (0, 1, 0)$ e $\vec{k} = (0, 0, 1)$ vetores da base canônica no $\mathbb{R}^3$. O vetor $\vec{u} = (x_1, y_1, z_1)$ é expresso pela seguinte combinação linear (STEINBRUCH; WINTERLE,

1987a):

$$\vec{v} = x_1\vec{i} + y_1\vec{j} + z_1\vec{k} \tag{3.2}$$

Em Steinbruch e Winterle (1987a) pode-se encontrar as seguintes definições:

- O **módulo** ou norma (VENTURI, 2015) de um vetor $\vec{v}$, expresso pela notação $|\vec{v}|$, é o seu comprimento. Considerando que $\vec{v} = (x_1, y_1, z_1)$ está no $\mathbb{R}^3$, o módulo é calculado através da equação:

$$|\vec{v}| = \sqrt{x_1^2 + y_1^2 + z_1^2} \tag{3.3}$$

- Dois vetores $\vec{u} = (x_1, y_1, z_1)$ e $\vec{u} = (x_2, y_2, z_2)$ são **iguais** se, e somente se, possuírem mesma direção, sentido e módulo. Em outras palavras, se $x_1 = x_2$, $y_1 = y_2$ e $z_1 = z_2$.
- O vetor cujo módulo é zero é denominado **vetor nulo**. É expresso pela notação $\vec{0}$.
- Dado o vetor $\vec{v}$, o **vetor oposto** $\vec{u} = -\vec{v}$ é um vetor de mesmo módulo e direção de $\vec{v}$, porém em sentidos opostos.
- O vetor no qual $|\vec{v}| = 1$ é denominado **vetor unitário**.
- O **versor** $\vec{u}$ de um vetor não nulo $\vec{v}$ é um vetor unitário que possui mesmo sentido e direção de $\vec{v}$. É calculado através da equação:

$$\vec{u} = \frac{\vec{v}}{|\vec{v}|} \tag{3.4}$$

- Dois vetores são **colineares** se possuem a mesma direção.
- Três ou mais vetores são **coplanares** se pertencem a um mesmo plano.

3.2.2 Produto escalar

Conforme Steinbruch e Winterle (1987a), o **produto escalar** ou **produto interno**(VENTURI, 2015) de dois vetores $\vec{u} = (x_1, y_1, z_1)$ e $\vec{v} = (x_2, y_2, z_2)$ é um valor escalar definido como:

$$\vec{u} \cdot \vec{v} = x_1x_2 + y_1y_2 + z_1z_2 \tag{3.5}$$

Também é utilizado a notação $\langle \vec{u}, \vec{v} \rangle$ para o produto escalar (STEINBRUCH; WINTERLE, 1987a). O produto escalar pode, conforme Venturi (2015), ser definido como sendo como:

$$\vec{u} \cdot \vec{v} = |\vec{u}||\vec{v}| \cos \alpha \tag{3.6}$$

onde α é o menor ângulo formado por $\vec{u}$ e $\vec{v}$. Assim, o valor de α está entre 0 e π.

Analisando a Equação 3.6, nota-se que o sinal do valor escalar de $\langle \vec{u}, \vec{v} \rangle$ depende do sinal de $\cos \alpha$, uma vez que o módulo de um vetor é sempre positivo (VENTURI, 2015). Portanto, pode-se obter as seguintes conclusões sobre o ângulo formado entre $\vec{u}$ e $\vec{v}$ (STEINBRUCH; WINTERLE, 1987a)

- O ângulo α é agudo quando $\langle \vec{u}, \vec{v} \rangle > 0$.
- O ângulo α é obtuso quando $\langle \vec{u}, \vec{v} \rangle < 0$.
- O ângulo α é reto ou igual a zero quando $\langle \vec{u}, \vec{v} \rangle = 0$.

3.2.3 Produto vetorial

Sejam $\vec{u} = (x_1, y_1, z_1)$ e $\vec{v} = (x_2, y_2, z_2)$ vetores no $\mathbb{R}^3$. O **produto vetorial**, expresso pela notação $\vec{u} \times \vec{v}$, retorna um vetor ortogonal a $\vec{u}$ e $\vec{v}$ obtido através da fórmula (STEINBRUCH; WINTERLE, 1987a):

$$\vec{u} \times \vec{v} = (y_1 z_2 - y_2 z_1, z_1 x_2 - z_2 x_1, x_1 y_2 - x_2 y_1) \tag{3.7}$$

Segundo Steinbruch e Winterle (1987a), a norma do produto vetorial de dois vetores $\vec{u}$ e $\vec{v}$ pode ser definido como:

$$|\vec{u} \times \vec{v}| = |\vec{u}||\vec{v}| \sin \alpha \tag{3.8}$$

onde α é o menor ângulo formado por $\vec{u}$ e $\vec{v}$. Assim, o valor de α está entre 0 e π.

3.2.4 Produto misto

Sejam $\vec{u}$, $\vec{v}$ e $\vec{w}$ vetores em $\mathbb{R}^3$. O **produto misto** $(\vec{u}, \vec{v}, \vec{w})$ é definido como o produto escalar entre o vetor $\vec{v}$ e do produto vetorial de $\vec{v}$ e $\vec{w}$ (STEINBRUCH; WINTERLE, 1987a). Matematicamente:

$$(\vec{u}, \vec{v}, \vec{w}) = \vec{u} \cdot (\vec{v} \times \vec{w}) \tag{3.9}$$

3.2.5 Condições e propriedades entre vetores

Sejam $\vec{u}$, $\vec{v}$ e $\vec{w}$ vetores no $\mathbb{R}^3$. As seguintes propriedades e condições podem ser afirmadas de acordo com Steinbruch e Winterle (1987a):

- **Condições de colinearidade**: dois vetores $\vec{u}$ e $\vec{v}$ são colineares se, e somente se, existir um valor real k tal que $\vec{u} = k\vec{v}$. De mesmo modo se existir um valor real m tal que $\vec{v} = m\vec{u}$. Se $\vec{u}$ ou $\vec{v}$ não for vetor nulo, pode-se concluir que dois vetores são colineares se o produto vetorial entre eles resultar no vetor nulo $\vec{0}$.
- **Condições de ortogonalidade**: dois vetores $\vec{u}$ e $\vec{v}$ são ortogonais se, e somente se, o produto escalar entre eles for igual a zero.
- **Condições de coplanaridade**: três vetores $\vec{u}$, $\vec{v}$ e $\vec{w}$ podem ser coplanares se o produto misto de $\vec{u}$, $\vec{v}$ e $\vec{w}$ for igual a zero.

3.2.6 Ângulo entre vetores e suas relações trigonomêtricas

Sejam $\vec{u}$ e $\vec{v}$ e α o ângulo formado entre eles. De acordo com Steinbruch e Winterle (1987a), através do produto escalar (Equação 3.6), o valor do cosseno de α pode ser expresso pela equação:

$$\cos\alpha = \frac{\langle \vec{u}, \vec{v} \rangle}{|\vec{u}||\vec{v}|} \tag{3.10}$$

De modo análogo, pode-se obter o seno do ângulo α através da norma do produto vetorial de $\vec{u}$ e $\vec{v}$ (Equação 3.8) como sendo (STEINBRUCH; WINTERLE, 1987a):

$$\operatorname{sen}\alpha = \frac{|\vec{u} \times \vec{v}|}{|\vec{u}||\vec{v}|} \tag{3.11}$$

Outras relações trigonométricas podem ser definidas. Conforme Iezzi (1978), o cálculo da tangente pode ser obtido pela seguinte relação trigonométrica:

$$\tan\alpha = \frac{\operatorname{sen}\alpha}{\cos\alpha} \tag{3.12}$$

O cálculo da cotangente pode ser expresso pela equação (IEZZI, 1978):

$$\cot\alpha = \frac{\cos\alpha}{\operatorname{sen}\alpha} \tag{3.13}$$

Através destas equações, pode-se reformular o cálculo da tangente como sendo:

$$\tan\alpha = \frac{\operatorname{sen}\alpha}{\cos\alpha} = \operatorname{sen}\alpha\frac{1}{\cos\alpha} = \frac{|\vec{u}\times\vec{v}|}{|\vec{u}||\vec{v}|}\frac{|\vec{u}||\vec{v}|}{\langle\vec{u},\vec{v}\rangle}$$

$$\tan\alpha = \frac{|\vec{u}\times\vec{v}|}{\langle\vec{u},\vec{v}\rangle} \tag{3.14}$$

Como a cotangente é o inverso da tangente (IEZZI, 1978), o cálculo da cotangente pode ser reformulado como o inverso da equação 3.14, ficando:

$$\cot\alpha = \frac{\langle\vec{u},\vec{v}\rangle}{|\vec{u}\times\vec{v}|} \tag{3.15}$$

3.2.7 Cálculo de áreas

Sejam A, B, C e D pontos tais que $\overrightarrow{AB} = \overrightarrow{CD}$. Sejam $\vec{u}$ e $\vec{v}$ vetores tais que $\vec{u} = \overrightarrow{AC}$ e $\vec{v} = \overrightarrow{AB}$. Seja α o ângulo formado entre $\vec{u}$ e $\vec{v}$ (Figura 8). A área do paralelogramo formado pelos pontos A, B, C e D pode ser obtida através da norma do produto vetorial de $\vec{u}$ e $\vec{v}$ (STEINBRUCH; WINTERLE, 1987a). Ou seja:

$$Area_{ABCD} = |\vec{u}\times\vec{v}| \tag{3.16}$$

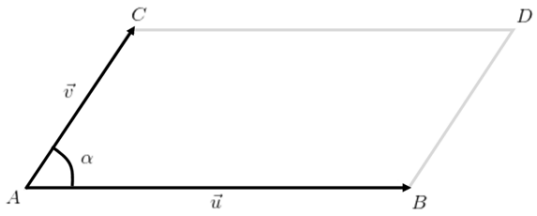

Figura 8 – Paralelogramo formado por dois vetores
Fonte: Steinbruch e Winterle (1987a) adaptada pelo autor

A partir da área do paralelogramo (Equação 3.16) pode-se obter a área do triângulo formado pelos pontos A, B e C, isto é, a área do triângulo formado pelos vetores $\vec{u}$ e $\vec{v}$ (Figura 9) como:

$$Area_{ABC} = \frac{|\vec{u}\times\vec{v}|}{2} \tag{3.17}$$

3.2.8 Planos no espaço

De acordo com Venturi (2015), um plano pode ser definido por um ponto e dois vetores não colineares. Seja $P_0 = (x_0, y_0, z_0)$ um ponto tal que P_0 pertence ao plano β e $\vec{u} = (x_1, y_1, z_1)$ e $\vec{v} = (x_2, y_2, z_2)$ vetores não colineares. Para que um ponto $P = (x, y, z)$

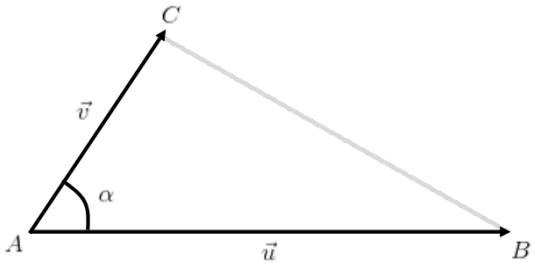

Figura 9 – Triângulo formado por dois vetores
Fonte: Steinbruch e Winterle (1987a) adaptada pelo autor

pertença ao plano β é necessário que os vetores $\vec{u}$, $\vec{v}$ e $\overrightarrow{P_0P} = (P - P_0)$ sejam coplanares (VENTURI, 2015). Conforme condições de coplanaridade entre vetores, um ponto P_0 pertence ao plano β se:

$$(\overrightarrow{P_0P}, \vec{u}, \vec{v}) = 0 \tag{3.18}$$

A partir da Equação 3.18, pode-se desenvolver a equação geral do plano. Utilizando Equações 3.5 e 3.7, temos:

$$\begin{aligned}(\overrightarrow{P_0P}, \vec{u}, \vec{v}) &= \langle \overrightarrow{P_0P}, \vec{u} \times \vec{v} \rangle \\ &= \langle \overrightarrow{P_0P}, (y_1z_2 - y_2z_1, z_1x_2 - z_2x_1, x_1y_2 - x_2y_1) \rangle \\ &= (x - x_0)(y_1z_2 - y_2z_1) + (y - y_0)(z_1x_2 - z_2x_1) + (z - z_0)(x_1y_2 - x_2y_1) \\ &= 0\end{aligned}$$

Considerando $a = y_1z_2 - y_2z_1$, $b = z_1x_2 - z_2x_1$ e $c = x_1y_2 - x_2y_1$, a equação pode ser reescrita semelhante a equação presente em Steinbruch e Winterle (1987a). Logo:

$$\begin{aligned}(\vec{PP_0}, \vec{u}, \vec{v}) &= (x - x_0)a + (y - y_0)b + (z - z_0)c \\ &= ax + by + cz + (-ax_0 - by_0 - cz_0) \\ &= 0\end{aligned}$$

Considerando $d = -ax_0 - by_0 - cz_0$, a equação geral do plano é definida como (STEINBRUCH; WINTERLE, 1987a):

$$\beta : ax + by + cz + d = 0 \tag{3.19}$$

Sejam β um plano e $\vec{u}$ e $\vec{v}$ vetores não colineares tais que $\vec{u} \in \beta$ e $\vec{v} \in \beta$. O **vetor normal** $\vec{n}$ do plano β (Figure 10) é um vetor ortogonal ao plano β que pode ser obtido

através do produto vetorial entre $\vec{u}$ e $\vec{v}$ (STEINBRUCH; WINTERLE, 1987a). Isto é:

$$\vec{n} = \vec{u} \times \vec{v} \tag{3.20}$$

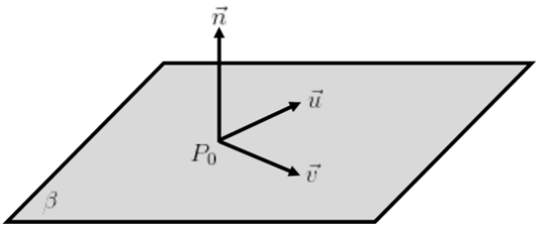

Figura 10 – Normal de um plano
Fonte: Steinbruch e Winterle (1987a) adaptada pelo autor

3.2.9 Projeção de vetores

Sejam $\vec{u}$ e $\vec{v}$ vetores e α o ângulo formado entre eles. Segundo Steinbruch e Winterle (1987a), a projeção (ou **ou vetor projetor**) do vetor $\vec{u}$ sobre o vetor $\vec{v}$ é um vetor colinear a $\vec{v}$ dado pela equação:

$$proj_{\vec{u}}\vec{v} = \frac{\langle \vec{u}, \vec{v} \rangle}{\langle \vec{u}, \vec{u} \rangle}\vec{v} \tag{3.21}$$

3.2.10 Projeção ortogonal de um ponto sobre o plano

Seja β um plano, $\vec{n}$ o vetor normal de β e O um ponto tal que $O \in \beta$. Seja P_0 um ponto no $\mathbb{R}^3$. A **projeção ortogonal** de P_0 sobre o plano β é um ponto P tal que o vetor $\overrightarrow{OP}$ é ortogonal ao vetor normal $\vec{n}$ e o vetor $\overrightarrow{P_0P}$ é colinear ao vetor normal $\vec{n}$ (VENTURI, 2015). Ou seja:

$$\overrightarrow{OP}\vec{n} = 0 \Rightarrow (P - O)\vec{n} = 0 \tag{3.22}$$

e

$$\overrightarrow{P_0P} = \lambda\vec{n} \Rightarrow P = P_0 + \lambda\vec{n} \tag{3.23}$$

Conforme Venturi (2015), ao substituir valor de P da Equação 3.23 na Equação 3.22, obtem:

$$
\begin{aligned}
(P_0 + \lambda\vec{n} - O)\vec{n} &= 0 \\
P_0\vec{n} + \lambda\vec{n}\vec{n} - O\vec{n} &= 0 \\
P_0\vec{n} - O\vec{n} &= \lambda\vec{n}\vec{n} \\
\lambda &= \frac{(P_0 - O)\vec{n}}{\vec{n}\vec{n}} \\
\lambda &= \frac{\langle\overrightarrow{OP_0}, \vec{n}\rangle}{\langle\vec{n}, \vec{n}\rangle}
\end{aligned}
$$

Considere $\vec{u} = \overrightarrow{OP_0}$. Ao substituir o valor de λ na Equação 3.23, obtem a fórmula da projeção ortogonal de um ponto P sobre o plano β como sendo:

$$
P = P_0 + \frac{\langle\vec{u}, \vec{n}\rangle}{\langle\vec{n}, \vec{n}\rangle}\vec{n} \tag{3.24}
$$

3.2.11 Transformações geométricas

Seja $P_0 = (x_0, y_0, z_0)$ um ponto e $\vec{v} = (a, b, c)$ um vetor. Segundo Bogomolny (1996), a **translação** do ponto P_0 na direção de $\vec{v}$ é um ponto tal que $\overrightarrow{P_0P} = \vec{v}$. Isto implica que:

$$
\begin{aligned}
\overrightarrow{P_0P} &= \vec{v} \\
P - P_0 &= \vec{v} \\
P &= P_0 + \vec{v} \\
(x, y, z) &= (x_0, y_0, z_0) + (a, b, c)
\end{aligned}
$$

Portanto, a translação do ponto P_0 é um ponto P tal que:

$$
(x, y, z) = (x_0 + a, y_0 + b, z_0 + c) \tag{3.25}
$$

A **rotação** tridimensional transforma um ponto $P_0 = (x_0, y_0, z_0)$ em outro ponto $P = (x, y, z)$ por rotacionar P_0 em torno de um dos eixos coordenados. Para isto é utilizado uma matriz de rotação própria para cada eixo (ANDRADE, 2000).

Uma matriz de rotação M é uma matriz de transformação linear. Portanto, o ponto P pode ser encontrado por um sistema linear na forma matricial (STEINBRUCH; WINTERLE, 1987b). Ou seja:

$$
\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} \tag{3.26}
$$

Seja α o ângulo de rotação. Segundo Andrade (2000), as matrizes de rotação para os eixos x, y e z para o ângulo α são, respectivamente, as matrizes $M_x(\alpha)$, $M_y(\alpha)$ e $M_y(\alpha)$:

$$M_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\operatorname{sen}\alpha \\ 0 & \operatorname{sen}\alpha & \cos\alpha \end{pmatrix} \tag{3.27}$$

$$M_y(\alpha) = \begin{pmatrix} \cos\alpha & 0 & -\operatorname{sen}\alpha \\ 0 & 1 & 0 \\ \operatorname{sen}\alpha & 0 & \cos\alpha \end{pmatrix} \tag{3.28}$$

$$M_z(\alpha) = \begin{pmatrix} \cos\alpha & -\operatorname{sen}\alpha & 0 \\ \operatorname{sen}\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.29}$$

3.3 Autovalores e autovetores

Seja x um vetor e A uma matriz. O vetor x é um **autovetor** se o vetor resultado da multiplicação entre A e x for múltiplo escalar de x. Isto pode ser expresso na seguinte equação (BORTOLI et al., 2003):

$$Ax = \lambda x \tag{3.30}$$

Os valores escalares λ expressos pela Equação 3.30 são denominados **autovalores**. Para o cálculo dos autovalores, como consequência da Equação 3.30, é preciso que (BORTOLI et al., 2003):

$$det(A - \lambda I) = 0 \tag{3.31}$$

O polinômio representado pela Equação 3.31 é denominado **polinômio característico** (BORTOLI et al., 2003). Os autovalores e autovetores podem ser apenas encontrados em matriz quadráticas, sendo que nem todas estas matrizes possuem esta propriedade. Dado uma matriz A de ordem n que contenha autovetores, existem n autovetores e n autovalores (SMITH, 2002). Além disso, se os autovalores são distintos entre si, logo os autovetores são **linearmente independentes** (BORTOLI et al., 2003).

3.4 Análise de Componentes Principais

A **análise de componentes principais** (*Principal Components Analysis* ou PCA) é uma abordagem estatística utilizado para transformar uma quantidade de va-

riáveis originais correlacionadas em um conjunto de variáveis independentes chamadas de **componentes principais**. Tal algoritmo é utilizado em diversas aplicações, tais como reconhecimento de padrões (como reconhecimento de faces e dígitos) e algoritmos de classificação (OLIVEIRA, 2012), estando também relacionado com a redução de dados, sendo possível realizar a análise de m variáveis de entrada observando as componentes principais mais relevantes com perda mínima de informação (VARELLA, 2008).

Seja $X_{n\times m}$ os dados de entrada sendo m a quantidade de variáveis ou atributos e n a quantidade de instâncias. O objetivo do algoritmo é transformar as variáveis correlacionadas $X_1, X_2, \ldots, X_m$ nas componentes principais $Y_1, Y_2, \ldots, Y_m$, as quais são independentes entre si. Para analisar o grau de correlação entre as variáveis da matriz X é necessário calcular a matriz de covariância S (ou matriz de correlação R), enquanto os componentes principais são obtidos através do cálculo dos autovalores e autovetores da matriz S (ou da matriz R). Caso utilizar a matriz de covariância é preciso padronizar os valores da matriz X, uma vez que matriz de covariância é sensível às unidades de medidas utilizadas em cada atributo(VARELLA, 2008).

Oliveira (2010) assim descreve os passos para obter as componentes principais da matriz X:

1. Normalize a matriz X, onde para cada dado $x_{j,i}$ da variável X_i subtrai seu valor pela média de X_i. Armazene os valores normalizados na matriz Z.
2. Calcule a matriz de covariância Σ para os dados normalizados Z.
3. Calcule os autovalores λ e os autovetores V da matriz de covariância Σ.
4. Rearranje os autovetores V de forma que seus correspondentes autovalores estejam ordenados de forma decrescente.
5. Como as componentes principais são combinações lineares das variáveis de entrada, constrói as componentes principais Y conforme descrito por Varella (2008):

$$Y_i = V_{i,1}X_1 + V_{i,2}X_2 + \ldots + V_{i,m}X_m$$

3.5 Diagramas de Voronoi

Diagramas de Voronoi são estruturas capazes de agrupar pontos em $\mathbb{R}^d$ em **regiões de Voronoi**. Para isto, é dado um conjunto $Z = \{z_1, z_2, \ldots, z_n\}$ de n pontos **sementes**, os quais irão definir as regiões. Cada região V_i está associada a um ponto semente z_i, para $i = 1, 2, \ldots, n$. Um ponto p pertence a uma região de Voronoi V_i se este ponto estiver mais próximo de z_i do que os demais pontos sementes $z_j \in Z - \{z_i\}$.

Com isto são definidas n regiões delimitadas por fronteiras que dividem os pontos de cada região, formando uma espécie de "mosaico" (Figura 11) (AURENHAMMER, 1991).

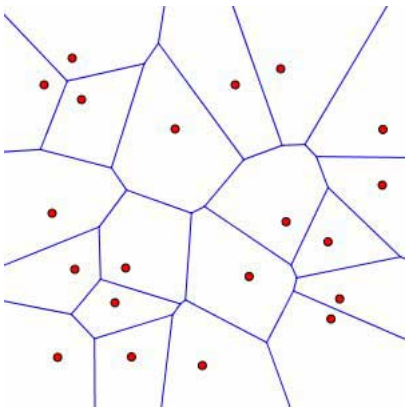

Figura 11 – Exemplo de diagrama de Voronoi
Fonte: Austin (2006)

Seja $z_i \in Z$ e $z_j \in Z$ pontos sementes tais que $i \neq j$ e $\delta(p, q)$ a distância euclidiana entre dois pontos p e q. A dominância $dom(z_i, z_j)$ de z_i sobre z_j é um subconjunto de pontos compostos por pontos que estão mais próximos de z_i que z_j (AURENHAMMER, 1991):

$$dom(z_i, z_j) = \{x \in \mathbb{R}^d | \delta(x, z_i) \leq \delta(x, z_j)\} \tag{3.32}$$

Baseado na Equação , uma região de Voronoi V_i formada pelo ponto z_i é o subconjunto de pontos em $\mathbb{R}^d$ que estejam em todas as dominâncias de z_i com todos os demais pontos $z_j \in Z - \{z_i\}$ (AURENHAMMER, 1991):

$$V_i = \bigcap_{z_j \in Z - \{z_i\}} dom(z_i, z_j) \tag{3.33}$$

A Equação 3.32 e a Equação 3.33 podem ser simplificadas na seguinte fórmula (VALETTE; CHASSERY, 2004):

$$V_i = \{x \in \mathbb{R}^d | \delta(x, z_i) \leq \delta(x, z_j), j = 1, 2, \ldots, n, i \neq j\} \tag{3.34}$$

3.6 Triangulação de Delaunay

Seja $V = p_1, p_2, \ldots, p_i, \ldots, p_n \subseteq \mathbb{R}^2, n \geq 3$ um conjunto de pontos não colineares. Uma **triangulação** $T(V)$ é um grafo onde todos os vértices são conectados de tal maneira que suas arestas não se cruzam, formando faces triangulares. A triangulação $T(V)$ é uma

triangulação de Delaunay se a circunferência formada pelos pontos de cada triângulo $\Delta(p_i, p_j, p_k) \in T(V)$ não possuir em seu interior qualquer ponto $p \in V$. Tal característica, conhecida como **critério de Delaunay**, garante que este tipo de triangulação seja única e minimiza o maior ângulo existente nesta triangulação (Figura 12) (PITERI et al., 2007).

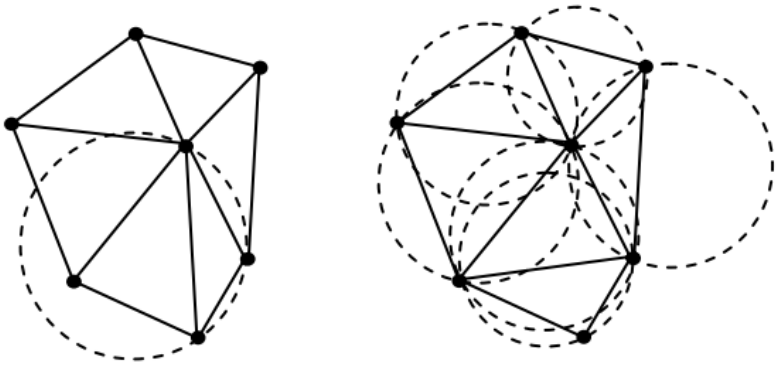

Figura 12 – Comparação entre uma triangulação que não satisfaz o critério de Delaunay (esquerda) e uma triangulação de Delaunay (direita)
Fonte: Piteri et al. (2007)

É possível notar que existe uma relação entre a triangulação de Delaunay, o diagrama de Voronoi e o **fecho convexo**, sendo este definido pelas arestas encontradas na borda de toda a triangulação de Delaunay. Nota-se também que a triangulação de Delaunay forma o **grafo dual** do diagrama de Voronoi (Figura 13) (PITERI et al., 2007).

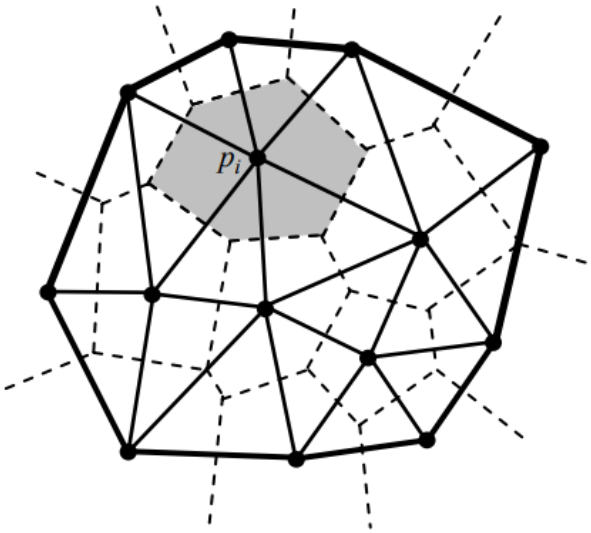

Figura 13 – Relação entre triangulação de Delaunay, diagramas de Voronoi e fecho convexo
Fonte: Piteri et al. (2007)

3.7 Estimativa do plano tangente

Seja $X = \{x_1, x_2, \ldots, x_i, \ldots, x_n\}$ um conjunto de pontos tal que $X \in \mathbb{R}^3$, $\pi_i = (o_i, \vec{n_i})$ o plano tangente associado a x_i tal que o_i é o ponto de centro de π_i e $\vec{n_i}$ o vetor normal deste plano. O plano π_i é estimado a partir de um subconjunto de X representando os k vizinhos mais próximos de x_i. Seja $N(x_i)$ o conjunto dos k vizinhos mais próximos de x_i. O ponto de centro o_i do plano π_i é definido como o centróide de $N(x_i)$. Para encontrar o vetor normal $\vec{n_i}$, é calculado a análise dos componentes principais de $N(x_i)$, obtendo três componentes principais devido a matriz composta pelos pontos $N(x_i)$ possuir três dimensões. O vetor $\vec{n_i}$ é definido pelo autovetor associado ao terceiro autovalor correspondente em ordem decrescente. O algoritmo descrito é definido pelo trabalho de Hoppe et al. (1992). Em suma, os passos para extração do plano tangente são definidos pelo Algoritmo 1:

Algoritmo 1: Extração de plano tangente

Entrada:

$X \leftarrow$ conjunto de pontos

$x_i \leftarrow$ ponto tal que $x_i \in X$

$k \leftarrow$ inteiro

Saída:

Plano tangente $\pi_i = (o_i, \vec{n_i})$, onde o_i é o ponto de centro e $\vec{n_i}$ a normal do plano

1 **início**

2 $N(x_i) \leftarrow k$ vizinhos mais próximos de x_i

3 $o_i \leftarrow$ centróide dos pontos de $N(x_i)$

4 $pca \leftarrow$ analise dos componentes principais de $N(x_i)$

5 $\vec{n_i} \leftarrow$ autovetor associado ao terceiro menor autovalor

6 **retorna** $(o_i, \vec{n_i})$

7 **fim**

3.8 Curvas de Hermite

As curvas de Hermite são curvas paramétricas representadas por polinômios de terceiro grau. Sua formulação requer quatro componentes: dois pontos $P1$ e $P2$, que são os pontos iniciais e finais da curva, e os vetores $T1$ e $T2$, os quais, respectivamente, são os vetores tangentes nos pontos $P1$ e $P2$ na curva. Os vetores $T1$ e $T2$ são os responsáveis por definir o formato da curva a partir de $P1$ até $P2$ (Figura 14) (AZEVEDO; CONCI, 2003).

De modo geral, para um instante $t \in [0, 1]$, as coordenadas dos pontos da curva de Hermite podem ser obtidas através dos seguintes polinômios (AZEVEDO; CONCI, 2003):

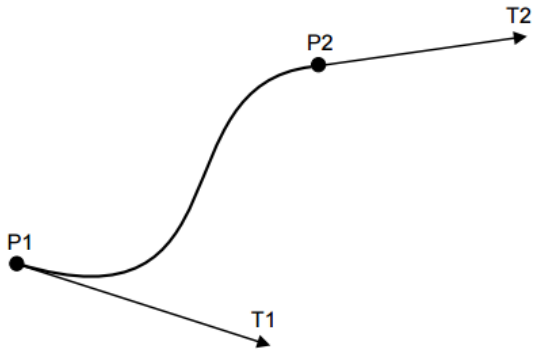

Figura 14 – Curvas de Hermite
Fonte: Azevedo e Conci (2003)

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x \quad y(t) = a_y t^3 + b_y t^2 + c_y t + d_y \quad z(t) = a_z t^3 + b_z t^2 + c_z t + d_z$$

Conforme Azevedo e Conci (2003), os polinômios podem ser expressos na seguinte notação matricial:

$$x(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} C_x \quad y(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} C_y \quad z(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} C_z$$

onde

$$C_x = \begin{bmatrix} a_x & b_x & c_x & d_x \end{bmatrix}^T \quad C_y = \begin{bmatrix} a_y & b_y & c_y & d_y \end{bmatrix}^T \quad C_z = \begin{bmatrix} a_z & b_z & c_z & d_z \end{bmatrix}^T$$

Uma curva de Hermite é definida ao encontrar os valores para a, b, c e d a partir dos valores de $P1$, $P2$, $T1$ e $T2$. O ponto $P1 = (P1_x, P1_y, P1_z)$ é obtido calculando cada coordenada de $P1$ através dos polinômios para $t = 0$. Matematicamente, isto significa que $P1 = (P1_x, P1_y, P1_z) = (x(0), y(0), z(0))$ e pode ser expresso como (AZEVEDO; CONCI, 2003):

$$P1_x = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} C_x \quad P1_y = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} C_y \quad P1_z = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} C_z$$

De modo análogo, o ponto $P2 = (P2_x, P2_y, P2_z)$ é obtido calculando cada coordenada de $P1$ através dos polinômios para $t = 1$ (AZEVEDO; CONCI, 2003):

$$P2_x = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} C_x \quad P2_y = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} C_y \quad P2_z = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} C_z$$

Como os vetores $T1$ e $T2$ estão relacionados à tangente da curva no ponto, é necessário obter a derivada de cada polinômio $x(t)$, $y(t)$ e $z(t)$, resultando nos polinômios abaixo (AZEVEDO; CONCI, 2003):

$$x'(t) = 3a_x t^2 + 2b_x t + c_x \quad y'(t) = 3a_y t^2 + 2b_y t + c_y \quad z'(t) = 3a_z t^2 + 2b_z t + c_z$$

A derivada dos polinômios podem ser expressas na sua forma matricial (AZEVEDO; CONCI, 2003):

$$x'(t) = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} C_x \quad y'(t) = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} C_y \quad z'(t) = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} C_z$$

Como $T1$ é o vetor tangente ao ponto $P1$ na curva, portanto $T1 = (T1_x, T1_y, T1_z)$ é obtido ao encontrar o ponto $(x'(t), y'(t), z'(t))$ quando $t = 0$, sendo expresso como (AZEVEDO; CONCI, 2003):

$$T1_x = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} C_x \quad T1_y = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} C_y \quad T1_z = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} C_z$$

De mesmo modo, $T2$ é obtido encontrando o ponto $(x'(t), y'(t), z'(t))$ quando $t = 1$ de acordo com a seguinte formulação (AZEVEDO; CONCI, 2003):

$$T2_x = \begin{bmatrix} 3 & 2 & 1 & 0 \end{bmatrix} C_x \quad T2_y = \begin{bmatrix} 3 & 2 & 1 & 0 \end{bmatrix} C_y \quad T2_z = \begin{bmatrix} 3 & 2 & 1 & 0 \end{bmatrix} C_z$$

Segundo Azevedo e Conci (2003), todas as condições anteriores podem ser representadas em uma única expressão para cada coordenada:

$$G_x = H^{-1} C_x, \quad G_y = H^{-1} C_y, \quad G_z = H^{-1} C_z$$

onde

$$G_x = \begin{bmatrix} P1 & P2 & T1 & T2 \end{bmatrix}_x^T \quad G_y = \begin{bmatrix} P1 & P2 & T1 & T2 \end{bmatrix}_y^T \quad G_z = \begin{bmatrix} P1 & P2 & T1 & T2 \end{bmatrix}_z^T$$

$$H^{-1} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}$$

Seja H a matriz inversa de H^{-1}. Como consequência, os vetores C_x, C_y e C_z podem ser encontrados pelas fórmulas abaixo (AZEVEDO; CONCI, 2003):

$$C_x = HG_x, \quad C_y = HG_y, \quad C_z = HG_z$$

onde

$$H = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Considere a matrix linha $T = [t^3 \quad t^2 \quad t \quad 1]$ e a matriz quadrada $G_h = [G_x \quad G_y \quad G_z]$. Segundo Azevedo e Conci (2003), um ponto em uma curva de Hermite delimitada entre os pontos $P1$ e $P2$ pode ser calculado pela seguinte equação:

$$P(t) = THG_h \tag{3.35}$$

A Equação 3.35 pode ser simplificada, resultando no seguinte polinômio, no qual $t \in [0, 1]$ (AZEVEDO; CONCI, 2003):

$$P(t) = (2t^3 - 3t^2 + 1)P1 + (-2t^3 + 3t^2)P2 + (t^3 - 2t^2 + t)T1 + (t^3 - t^2)T2 \tag{3.36}$$

3.9 Suavização laplaciana

O processo de **suavização laplaciana** é baseado na equação diferencial de difusão de calor, definida pela seguinte fórmula (DESBRUN et al., 1999):

$$\frac{\partial u}{\partial t} = \lambda \Delta u \tag{3.37}$$

Na Equação 3.37, o operador Δu é o **operador laplaciano**, definido como sendo (PEIRCE, 2014):

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \tag{3.38}$$

Conforme Desbrun et al. (1999), um dos meios de resolver a Equação 3.37 é através do **método de Euler**, o qual é um método de cálculo numérico iterativo que pode ser encontrado em Guidi (2014). Seja $v_i = (x_i, y_i, z_i)$ um vértice de uma malha triangular.

Segundo Taubin (1995), a Equação 3.37 pode ser resolvida explicitamente pela seguinte fórmula iterativa:

$$v_{i+1} = v_i + \lambda \Delta u(v_i) \tag{3.39}$$

A Equação 3.39 é a fórmula explícita para a **suavização laplaciana**. O operador laplaciano Δu discreto é definido como a somatório ponderado da diferença entre cada vértice v_j vizinhos a v_i com v_i, conforme a seguinte equação (TAUBIN, 1995):

$$\Delta u(v_i) = \sum_{j \in N(v_i)} \mathbf{w_{ij}}(v_j - v_i) \tag{3.40}$$

Segundo Nealen et al. (2006), a soma de todos os pesos de $\mathbf{w_{ij}}$ para um vértice v_i deve ser igual a 1. O peso $\mathbf{w_{ij}}$ da aresta $e = (i, j)$ da malha triangular é definido como sendo (TAUBIN, 1995):

$$\mathbf{w_{ij}} = \frac{w_{ij}}{\sum_{j \in N(v_i)} w_{ij}} \tag{3.41}$$

O resultado do operador laplaciano definido pela Equação 3.40 pode variar dependendo do peso w_{ij} a ser utilizado. Quando $w_{ij} = 1$, é definido o **operador laplaciano uniforme** ou operador "guarda-chuva" (Figura 15). Sendo $d = |N(v_j)|$ o número de vértices vizinhos, este operador é definido pela seguinte equação (DESBRUN et al., 1999):

$$\Delta u(v_i) = \frac{1}{d} \sum_{j \in N(v_i)} (v_j - v_i) \tag{3.42}$$

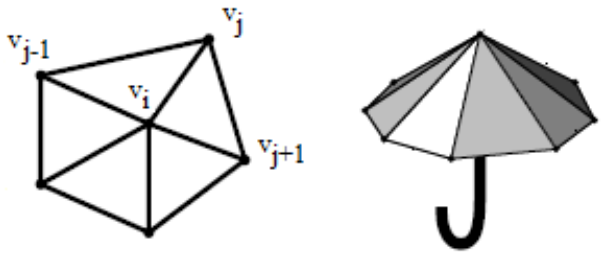

Figura 15 – Operador "guarda-chuva" para suavização laplaciana
Fonte: Kobbelt et al. (1998) adaptada pelo autor

Outro alternativa para o peso w_{ij} utilizada é aquele envolvendo a cotangente dos ângulos opostos à aresta e_{ij} (Figura 17). Sendo α_{ij} e β_{ij} os ângulos opostos à aresta e_{ij}, o peso w_{ij} é definido como sendo (NEALEN et al., 2006):

$$w_{ij} = \cot \alpha_{ij} + \cot \beta_{ij} \tag{3.43}$$

Outro tipo de peso a ser considerado envolve o cálculo da tangente dos ângulos adjacentes à aresta e_{ij}) (Figura 17). Sendo θ_{ij}^1 e θ_{ij}^2 os ângulos opostos à aresta e_{ij}, o peso w_{ij} é definido como sendo (SORKINE, 2005):

$$w_{ij} = \frac{\tan(\theta_{ij}^1/2) + \tan(\theta_{ij}^2/2)}{\|v_i - v_j\|} \tag{3.44}$$

O cálculo do peso tangente da Equação 3.44 utiliza a tangente da metade do ângulo θ, o qual pode ser calculado pela seguinte equação encontrada em Iezzi (1978):

$$\tan\theta = \pm\sqrt{\frac{1 - cos\theta}{1 + cos\theta}} \tag{3.45}$$

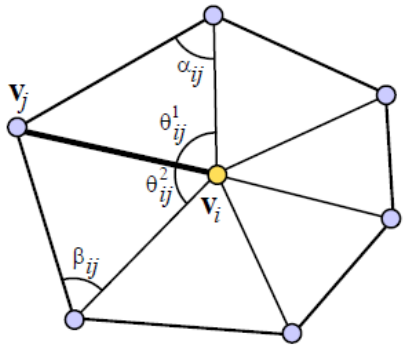

Figura 16 – Ângulos utilizados pelos pesos cotangente e tangente
Fonte: Sorkine (2005)

A Figura 17 abaixo apresenta o resultado entre os pesos uniforme, cotangente e tangente seguindo processo de suavização laplaciana expresso pela Equação 3.39 utilizando 10 iterações.

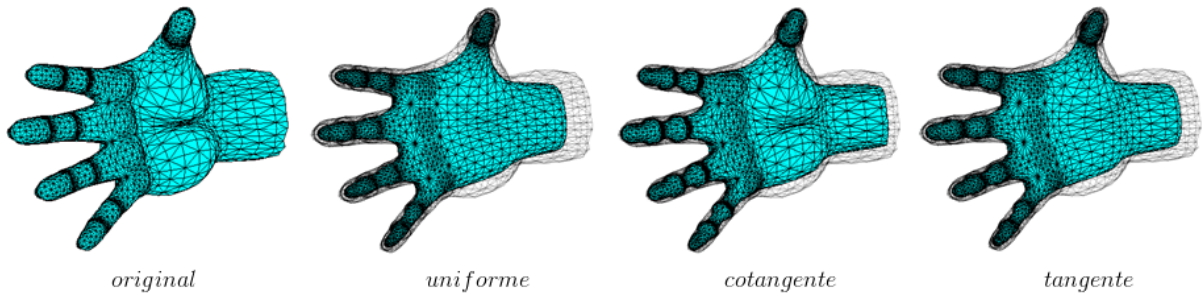

Figura 17 – Suavização laplaciana utilizando os pesos uniforme, cotangente e tangente
Fonte: Elaborada pelo autor

3.10 Matrizes esparsas CSR e CCS

Uma matriz A é esparsa quando a quantidade de elementos nulos é muito maior que a quantidade de elementos não nulos. Algumas estruturas de dados podem ser utilizadas a fim de reduzir a complexidade de espaço de uma matriz esparsas, como as matrizes

esparsas **CSR** (*Compressed Row Storage*) e **CCS** (*Compressed Column Storage*) (BARRETT et al., 1994).

As matrizes esparsas **CSR** e **CCS** são formatos utilizados para representar computacionalmente as informações de uma matriz esparsa de tal forma que não seja necessário armazenar elementos nulos (iguais a zero). Estas estruturas podem ser úteis quando é preciso representar uma matriz com uma elevada quantidade de linhas e colunas que pode conter poucos elementos diferentes de zero. Seja A uma matriz de ordem n e nnz a quantidade de elementos não nulos em A. Uma estrutura CSR ou CCS utiliza $2nnz + n + 1$ elementos comparado com o armazenamento de n^2 elementos (BARRETT et al., 1994).

O formato CSR utiliza três vetores: um **vetor de dados**, um **vetor de índices de colunas** e outro **vetor de ponteiro de linhas**. O vetor de dados `data` de comprimento nnz contém todos os elementos não nulos da matriz na mesma sequência que obtida ao percorrer a matriz linha por linha. O vetor de índices de colunas `col_index` de tamanho nnz armazena a coluna em que um elemento está na matriz. Isto significa que `col_index[c]` informa em qual coluna o elemento em `data[c]` está na matriz. O vetor de ponteiro de linhas `row_ptr` de comprimento $n+1$ informa a posição no vetor de dados onde inicia uma nova linha na matriz. Isto é, o valor de `row_ptr[r]` indica a posição do primeiro elemento não nulo da linha `r` no vetor `data`. Com isto, para obter todos os elementos de uma linha `r`, basta recuperar os valores de `data[k]` tais que `row_ptr[r]` $\leq$ `k` $<$ `row_ptr[r + 1]` (BARRETT et al., 1994).

Baseado no exemplo encontrado em Barrett et al. (1994), considere a seguinte matriz A:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 0 & 7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 \\ 0 & 4 & 0 & 0 & 2 & 0 \end{bmatrix}$$

Os vetores utilizados pelo formato CSR são armazenados do seguinte modo:

$$\texttt{data} = \begin{bmatrix} 1 & 2 & 3 & 9 & 3 & 7 & 7 & 3 & 4 & 2 \end{bmatrix}$$
$$\texttt{col\_index} = \begin{bmatrix} 1 & 5 & 1 & 2 & 6 & 2 & 4 & 6 & 2 & 5 \end{bmatrix}$$
$$\texttt{row\_ptr} = \begin{bmatrix} 1 & 3 & 6 & 8 & 8 & 9 & 11 \end{bmatrix}$$

O formato CCS é praticamente idêntico ao formato CRS, diferenciando do formato CRS apenas na organização dos dados, sendo estes armazenados conforme as colunas de

uma matriz. Para isto, os vetores `col_index` e `row_ptr` são substituidos pelos os vetores `row_index` e `col_ptr`, os quais armazenam a posição das linhas de cada elemento e a posição onde inicia uma coluna, respectivamente. Neste formato, cada coluna `c` é definida pelos valores de `data[k]` tais que `col_ptr[c]` $\leq$ `k` $<$ `row_ptr[c + 1]`. Considerando a matriz A do exemplo anterior, o formato CCS é armazenado desta maneira (BARRETT et al., 1994):

$$\texttt{data} = \begin{bmatrix} 1 & 3 & 9 & 7 & 4 & 7 & 2 & 2 & 3 & 3 \end{bmatrix}$$
$$\texttt{row\_index} = \begin{bmatrix} 1 & 2 & 2 & 3 & 6 & 3 & 1 & 6 & 2 & 5 \end{bmatrix}$$
$$\texttt{col\_ptr} = \begin{bmatrix} 1 & 3 & 6 & 6 & 7 & 9 & 11 \end{bmatrix}$$

4 Análise comparativa

Na literatura existem diversas abordagens para a extração de esqueleto de objetos 3D. Cada abordagem é utilizada de acordo com o tipo de representação do objeto tridimensional, tais como voxels, malhas triangulares ou nuvem de pontos. Este capítulo apresenta uma análise de alguns dos algoritmos de esqueletização de objetos 3D existentes. A Tabela 2 mostra os trabalhos que são abordados neste capítulo. A Tabela 3 apresenta uma descrição sucinta de cada abordagem.

Tabela de abordagens de esqueletização		
Categoria	**Publicação**	**Representação**
Baseados em eixo de simetria rotacional	Curve Skeleton Extraction from Incomplete Point Clouds (TAGLIASACCHI; ZHANG; COHEN-OR, 2009)	Nuvem de pontos
Baseados em contração	Skeleton Extraction by Mesh Contraction (AU et al., 2008)	Malhas triangulares
	Mean Curvature Skeletons (TAGLIASACCHI et al., 2012)	Malhas triangulares
	Curve skeleton extraction by coupled graph contraction and surface clustering (JIANG et al., 2013)	Malhas triangulares
	Point Cloud Skeletons via Laplacian-Based Contraction (CAO et al., 2010)	Nuvem de pontos

Tabela 2 – Abordagens de esqueletização

4.1 Esqueletização por eixo de simetria rotacional

Tagliasacchi, Zhang e Cohen-Or (2009) propõem um algoritmo para extração do esqueleto de um objeto tridimensional representado por uma nuvem de pontos utilizando o conceito de **eixo de simetria rotacional** (*rotational symmetric axis* ou *ROSA*). Trata-se de um algoritmo iterativo no qual são definidos planos de corte no objeto 3D de onde se obtém o eixo de simetria rotacional de um conjunto de pontos neste plano de corte. Cada ponto calculado é incluído como um ponto no esqueleto.

O algoritmo também tem como objetivo computar o esqueleto mesmo quando existe perda de informação da nuvem de pontos de entrada. Para cumprir este objetivo,

Tabela de abordagens de esqueletização	
Publicação	**Descrição**
Curve Skeleton Extraction from Incomplete Point Clouds (TAGLIASACCHI; ZHANG; COHEN-OR, 2009)	Busca extrair através de planos de corte no objeto 3D a fim de encontrar eixos de simetria rotacional, os quais irão compor o esqueleto.
Skeleton Extraction by Mesh Contraction (AU et al., 2008)	Utiliza o algoritmo de suavização laplaciana para iterativamente contrair a forma até a extração do esqueleto.
Mean Curvature Skeletons (TAGLIASACCHI et al., 2012)	Utiliza o algoritmo de suavização laplaciana para contração da forma tridimensional semelhante ao trabalho de Au et al. (2008). Entretanto, busca otimizar o processo de suavização ao remodelar e obter os pólos de Voronoi da forma como pré-processamento. O processo de suavização é aplicado sobre os pólos de Voronoi.
Curve Skeleton Extraction by Coupled Graph Contraction and Surface Clustering (JIANG et al., 2013)	Utiliza uma abordagem combinando contração do grafo do esqueleto e agrupamento de vértices da superfície utilizando diagramas de Voronoi de tal forma que cada nó do esqueleto possa ser mapeado a um agrupamento de vértices.
Point Cloud Skeletons via Laplacian-Based Contraction (CAO et al., 2010)	Adaptação do processo de contração de uma malha triangular por suavização laplaciana de Au et al. (2008) para nuvem de pontos. Para isto é utilizado a triangulação de Delaunay para construir as conexões entre os vértices.

Tabela 3 – Descrição das abordagens de esqueletização

o algoritmo também requer como entrada os vetores normais de cada ponto (TAGLIASACCHI; ZHANG; COHEN-OR, 2009), que podem ser obtidos através de fotometria (NEHAB et al., 2005) ou calculados através dos pontos originais (HOPPE et al., 1992). Além disto, o uso de eixos de simetria rotacional requer que as regiões da forma 3D, com exceção das articulações, devem ser geralmente cilíndricas (TAGLIASACCHI; ZHANG; COHEN-OR, 2009).

Tagliasacchi, Zhang e Cohen-Or (2009) utiliza a disponibilidade dos vetores normais de cada ponto em sua definição para o eixo de simetria rotacional. Seja **S** um subconjunto local dos pontos $p = (x_p, \vec{v_p})$, onde p representa um ponto no espaço tal que x_p indica sua posição e $\vec{v_p}$ seu vetor normal. Segundo os autores, para que o ponto p seja o mais rotationalmente simétrico a **S** é necessário cumprir requisitos em relação ao vetor normal e a posição de p (Figura 18), os quais são:

1. O vetor normal $\vec{v_p}$ deve possuir o ângulo com valor mais próximo dos demais vetores normais dos pontos do conjunto **S**. Isto implica que a variação entre o ângulo formado entre o vetor normal $\vec{v_p}$ e os demais vetores normais deve ser mínima.

2. A posição x_p do ponto p busca minimizar o somatório de quadrado das distâncias entre o ponto p e a reta formada por cada ponto em **S** e seu vetor normal correspondente.

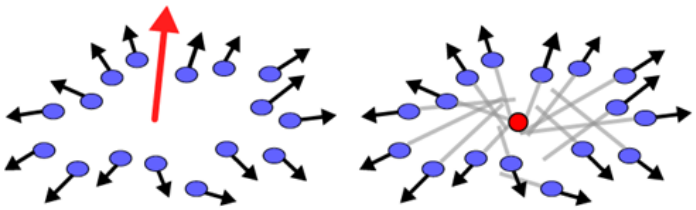

Figura 18 – Ponto *ROSA* conforme requisito de orientação (à esquerda) e posição (à direita)
Fonte: Tagliasacchi, Zhang e Cohen-Or (2009)

O algoritmo de Tagliasacchi, Zhang e Cohen-Or (2009) consiste em realizar iterativamente **planos de corte** ao longo do objeto tridimensional, no qual para cada conjunto de pontos que definem uma região no plano é obtido o eixo de simetria rotacional. Este plano deve ser definido de tal modo que a variação dos ângulos formados entre os vetores normais dos pontos e a normal do ponto *ROSA* seja mínima. Os autores simplificam a definição de um plano de corte ótimo "ancorando" a busca por este plano em um ponto da amostra, isto é, assumindo que um ponto da nuvem de pontos pertença ao plano de corte.

Seja p_i um ponto pertencente à nuvem de pontos e π_i um plano de corte de vetor normal $\vec{v_i}$ tal que $p_i \in \pi_i$. A construção de um plano de corte requer identificar quais pontos pertencem a uma **faixa estreita**, cujos pontos estão a uma distância menor que δ. A variável δ é definido como $2,5\%$ do comprimento da **diagonal da caixa delimitadora** da nuvem de pontos de entrada (TAGLIASACCHI; ZHANG; COHEN-OR, 2009).

O algoritmo também requer identificar quais pontos dentro da faixa estreita de π_i pertencem à **vizinhança relevante** N_i, de tal forma que $p_i \in N_i$. Isto porque os pontos que estão na faixa estreita do plano π_i podem pertencer a agrupamentos diferentes. A Figura 19 apresenta como identificar uma vizinhança relevante influencia no cálculo do eixo de simetria rotacional. É possivel notar como as normais dos pontos auxiliam o algoritmo a identificar tais agrupamentos (TAGLIASACCHI; ZHANG; COHEN-OR, 2009).

Para identificar a vizinhança relevante N_i utiliza-se a formulação para a **distância de Mahalanobis** encontrada em Lehtinen et al. (2008), a qual envolve a posição e a nor-

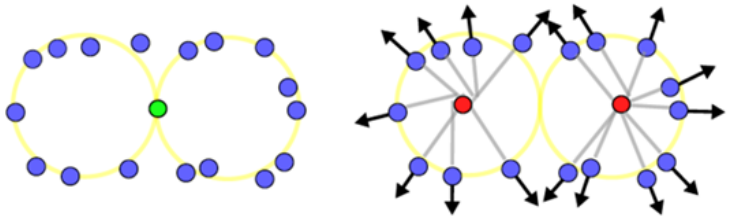

Figura 19 – Posição e orientação dos pontos auxiliando identificação dos agrupamentos
Fonte: Tagliasacchi, Zhang e Cohen-Or (2009)

mal dos pontos. Para auxiliar o processo para obter a vizinhança relevante N_i do ponto p_i, é construído previamente um grafo, chamado neste algoritmo de **grafo de Mahalanobis**, onde cada vértice corresponde a um ponto da nuvem de pontos de entrada. Cada aresta $e = (p_j, p_k)$ neste grafo existe se, e somente se, a distância de Mahalanobis d_{Mah} entre os pontos p_j e p_k for menor que um limiar ϵ_{Mah}. Construído o grafo de Mahalanobis, a vizinhança relevante N_i de um ponto p_i pode ser encontrada buscando a **componente conexa** no grafo que contém o ponto p_i como vértice. A vizinhança relevante N_i é definida pelos pontos que fazem parte desta componente conexa e encontram-se dentro da faixa estreita do plano π_i. Pode-se utilizar a **busca em largura** a partir de p_i para obter os pontos da componente conexa (TAGLIASACCHI; ZHANG; COHEN-OR, 2009).

O algoritmo busca encontrar um **plano de corte ótimo**, isto é, um plano de corte cujo vetor normal é o mais rotacionalmente simétrico em relação aos vetores normais dos pontos da vizinhança. Dado que p_i pertence ao plano de corte, resta calcular o vetor normal deste plano. Para isto, o algoritmo realiza um processo iterativo. Primeiramente, selecione um vetor normal inicial v_i^0 aleatoriamente dentre os vetores perpendiculares à normal de p_i. Seguindo os requisitos para a definição de um eixo de simetria rotacional, a orientação do vetor normal do plano é modificada de tal modo que minimize a variação entre os ângulos formados pela normal do plano e os vetores normais dos pontos da vizinhança N_i (TAGLIASACCHI; ZHANG; COHEN-OR, 2009).

Conforme Tagliasacchi, Zhang e Cohen-Or (2009), para a $(t+1)$-ésima iteração, o vetor normal para o plano de corte é definida através da Equação 4.1.

$$v_i^{t+1} = \underset{v \in \mathbb{R}^3, \quad |v|=1}{argmin} \quad \text{var}\{\langle v, n(p_j)\rangle : p_j \in N_i^t\} \tag{4.1}$$

onde N_i^t é a vizinhança de p_i na t-ésima iteração e $n(p_j)$ é o vetor normal unitário do ponto p_j. Ao observar a Equação 4.1 e relação entre o produto escalar e as condições de ortogonalidade entre vetores conforme Steinbruch e Winterle (1987a), percebe-se que para se obter um melhor plano de corte é preciso que o vetor normal do plano seja o mais ortogonal possível dentre as normais dos pontos da vizinhança. A Equação 4.1, segundo Tagliasacchi, Zhang e Cohen-Or (2009) pode ser resolvida através de um problema de

minimização de uma forma quadrática $v^T \mathrm{M} v$, sujeito a $|v| = 1$, onde M é uma matriz definida como:

$$\mathrm{M} = \begin{bmatrix} \overline{\mathcal{X}^2} - \bar{\mathcal{X}}^2 & 2\overline{\mathcal{X}\mathcal{Y}} - 2\bar{\mathcal{X}}\bar{\mathcal{Y}} & 2\overline{\mathcal{X}\mathcal{Z}} - 2\bar{\mathcal{X}}\bar{\mathcal{Z}} \\ 2\overline{\mathcal{X}\mathcal{Y}} - 2\bar{\mathcal{X}}\bar{\mathcal{Y}} & \overline{\mathcal{Y}^2} - \bar{\mathcal{Y}}^2 & 2\overline{\mathcal{Y}\mathcal{Z}} - 2\bar{\mathcal{Y}}\bar{\mathcal{Z}} \\ 2\overline{\mathcal{X}\mathcal{Z}} - 2\bar{\mathcal{X}}\bar{\mathcal{Z}} & 2\overline{\mathcal{Y}\mathcal{Z}} - 2\bar{\mathcal{Y}}\bar{\mathcal{Z}} & \overline{\mathcal{X}^2} - \bar{\mathcal{X}}^2 \end{bmatrix}$$

onde $\mathcal{X}$, $\mathcal{Y}$ e $\mathcal{Z}$ representam os valores das coordenadas x, y e z dos vetores normais dos pontos da vizinhança N_i^t.

Seja p_i um ponto e π_i^* um plano de corte ótimo tal que $p_i \in \pi_i^*$. Seja N_i^* a vizinhança relevante em π_i^* tal que $N_i^* \in p_i$. Seja r_i^* o ponto de eixo de simetria rotacional para os pontos da vizinhança N_i^*. Segundo Tagliasacchi, Zhang e Cohen-Or (2009), o ponto r_i^* pode ser calculado através da Equação 4.2:

$$r_i^* = \underset{r \in \mathbb{R}^3}{argmin} \sum_{p_j \in N_i^*} ||(r - p_j) \times \mathrm{n}(p_j)||^2 \tag{4.2}$$

Esta equação busca minimizar o somatório de quadrado das distâncias entre o ponto r_i^* e a reta formada pela posição e orientação de cada ponto na vizinhança N_i^*. O produto vetorial expresso na Equação 4.2 é simplesmente a fórmula para a distância entre um ponto e uma reta encontrada em Steinbruch e Winterle (1987a), sendo $\left|\mathrm{n}(p_j)\right| = 1$.

O procedimento de extração do eixo de simetria rotacional através de planos de cortes resulta em pontos bem definidos quando aplicados nas ramificações do objeto 3D. Contudo, em regiões não geralmente cilíndricas, tais como as articulações, tal abordagem pode resultar em pontos bem espalhados. Para isto é necessário um pós-processamento para as regiões de articulação (TAGLIASACCHI; ZHANG; COHEN-OR, 2009).

Após extrair os pontos de eixo de simetria rotacional, é aplicado a **suavização Laplaciana** nos pontos resultantes, onde as conexões necessárias para aplicar o processo de suavização são extraídas através das relações de proximidades entre os pontos pela distância de Mahalanobis. Com esta estratégia, é possível reduzir o ruído causado pela dispersão dos pontos de eixo de simetria rotacional nas articulações, além de tornar mais nítidas as conexões entre as junções e os ramos (TAGLIASACCHI; ZHANG; COHEN-OR, 2009).

Contudo, a suavização Laplaciana não é capaz de convergir a nuvem de pontos em uma curva. Para isto, após a etapa de suavização, é aplicado um processo de afinamento através de **mínimos quadrados móveis**, no qual os pontos ROSA são movidos a uma linha ajustada localmente através da análise de componentes principais local (TAGLIASACCHI; ZHANG; COHEN-OR, 2009). Este algoritmo utilizado para afinamento pode ser encontrado em Lee (1999). O afinamento por mínimos quadrados móveis é apli-

cado nos ramos devido a sua forma parecida com uma linha, o que não é encontrado nos pontos presentes em articulações. Para isto, a fim de distinguir ramos de articulações, é calculado uma **medida de linearidade** $\psi(r_i)$ no ponto r_i conforme a seguinte equação (TAGLIASACCHI; ZHANG; COHEN-OR, 2009):

$$\psi(r_i) = \frac{\lambda_i^{(1)}}{\lambda_i^{(1)} + \lambda_i^{(2)} + \lambda_i^{(3)}} \tag{4.3}$$

Os valores de $\lambda_i^{(1)}$, $\lambda_i^{(2)}$ e $\lambda_i^{(3)}$ são obtidos pela **análise dos componentes principais local** no ponto r_i, onde $\lambda_i^{(j)}$ é o j-ésimo é o maior autovalor obtido. Para que um ponto r_i esteja em um ramo, é preciso que a medida $\psi(r_i)$ seja menor que um limiar ϵ_{MLS}. Caso este critério seja satisfeito, é aplicado em r_i o afinamento por mínimos quadrados móveis. Empiricamente, este limiar é definido como $\epsilon_{MLS} = 0.4$ (TAGLIASACCHI; ZHANG; COHEN-OR, 2009).

Como os procedimentos anteriores pode afetar a centralidade do esqueleto, os pontos são recentralizados utilizando o mesmo cálculo de extração do ponto de eixo de simetria rotacional expresso pela Equação 4.2. Caso um ponto r_i esteja em um ramo, este é definido pelo eixo de simetria rotacional para uma pequena vizinhança de r_i. Caso r_i esteja em uma articulação, todos os pontos desta articulação são utilizados, resultando em um único ponto de simetria rotacional localizado no centro da articulação. Para distinguir entre ramos e articulação, é utilizado o critério definido pela Equação 4.3, onde um ponto de ramo é identificado quando $\psi(r_i) < \epsilon_{MLS}$. Em seguida é utilizado a suavização Laplaciana novamente para reorganizar os pontos conforme anteriormente. Por final, os pontos são conectados com pequenas curvas conforme em Lee (1999), resultando no esqueleto do objeto 3D (TAGLIASACCHI; ZHANG; COHEN-OR, 2009).

A Figura 20 mostra o processo de tratamento das articulações. Em suma, o processo é realizado após a etapa de extração dos eixos de simetria rotacional através de planos de corte, cujo resultado está presente na Figura 20a. Em seguida é aplicado a suavização Laplaciana sobre os pontos (Figura 20b) e o afinamento dos ramos via mínimos quadrados móveis (Figura 20c). Os pontos dos ramos e articulações são então recentralizados através do eixo de simetria rotacional (Figura 20d), redistribuidos utilizando suavização Laplaciana (Figura 20e) e conectados com pequenos segmentos (Figura 20f), o que resulta no esqueleto final do objeto 3D (TAGLIASACCHI; ZHANG; COHEN-OR, 2009).

4.1.1 Análise do processo de esqueletização

O algoritmo, conforme Tagliasacchi, Zhang e Cohen-Or (2009), consegue ser robusto mesmo com perda de informação da nuvem de pontos de entrada. Isto significa que este algoritmo é capaz de extrair o esqueleto mesmo se a forma estiver incompleta,

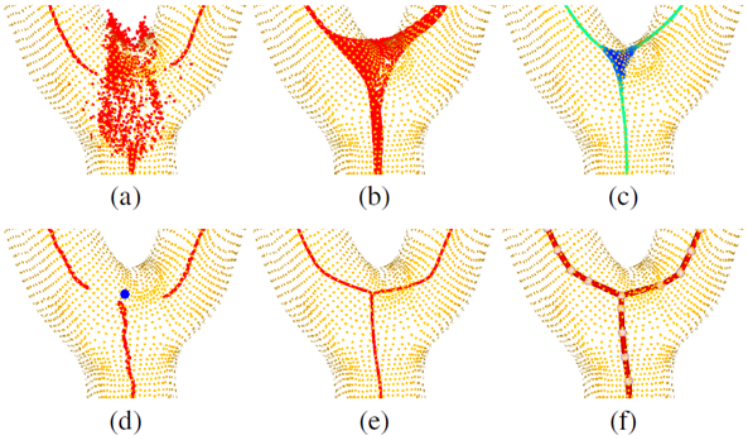

Figura 20 – Etapas de pós-processamento para tratamento de articulações
Fonte: Tagliasacchi, Zhang e Cohen-Or (2009)

com uma grande quantidade de dados perdida (TAGLIASACCHI; ZHANG; COHEN-OR, 2009). Segundo os mesmos autores, a perda de informação em nuvem de pontos é comum quando o objeto é obtido através do processo de digitalização 3D a *laser*. Para cumprir este objetivo, o algoritmo também requer como entrada os vetores normais de cada ponto. Nota-se que o uso dos vetores normais é crucial para o sucesso do algoritmo. (TAGLIASACCHI; ZHANG; COHEN-OR, 2009).

Os vetores normais para os pontos de entrada são obtidos através do algoritmo proposto por Hoppe et al. (1992). A publicação de Hoppe et al. (1992) sobre reconstrução da superfície de um objeto 3D representado por nuvem de pontos aborda um algoritmo simples para extração do vetor normal dos pontos. Trata-se de um trabalho referenciado inclusive pelo MATLAB através da função `pcnormals` (MATHWORKS, 2016a). Nesta publicação encontra-se um algoritmo para estimar o plano tangente (Seção 3.7) de um ponto utilizando análise de componentes principais.

No entanto, o algoritmo para estimar o plano tangente pode resultar dois vetores normais válidos: um vetor normal $\vec{n}$ e seu vetor oposto $(-\vec{n})$. Estas normais possuem mesma direção e módulo porém podem estar orientados para o lado externo ou interno do objeto 3D. Apenas estimar os planos tangentes aos pontos não garantem uma orientação consistente destes vetores (HOPPE et al., 1992). A Figura 21 apresenta um exemplo de extração dos vetores normais dos planos tangentes aos pontos do modelo tridimensional de um tórus utilizando busca por 10 vizinhos mais próximos.

Para solucionar esta inconsistência na orientação dos vetores, Hoppe et al. (1992) propõe um algoritmo para uma propagação consistente dos vetores normais. O autor parte da idéia de que para dois pontos x_i e x_j da nuvem de pontos, os planos tangente a estes pontos para uma superfície densa e suavizada são aproximadamente paralelos. Matematicamente, isto significa que $\langle \vec{n_i}, \vec{n_j} \rangle \approx \pm 1$, onde $\vec{n_i}$ e $\vec{n_j}$ são os vetores normais para, respectivamente, os planos tangente aos pontos x_i e x_j. Nota-se que, conforme

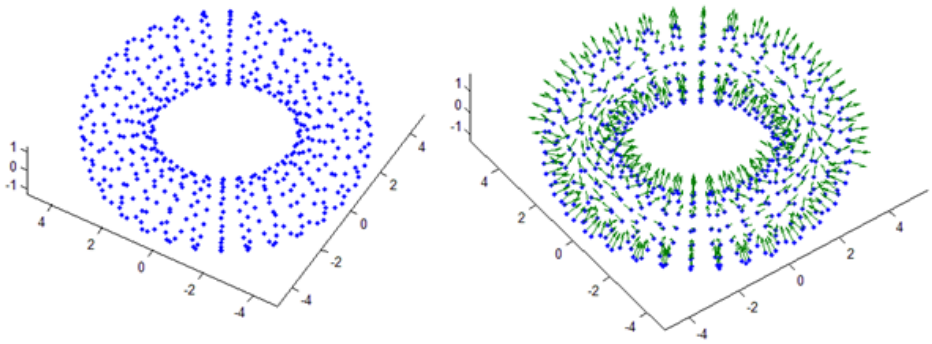

Figura 21 – Extração de normais de planos tangentes sem orientação consistente
Fonte: Elaborada pelo autor

Hoppe et al. (1992), $\vec{n_i}$ e $\vec{n_j}$ possuem orientação consistente quando seu produto escalar se aproxima de 1 e possuem orientação praticamente opostas quando o produto escalar entre eles se aproxima de -1.

Seja x_i e x_j pontos onde $\vec{n_i}$ e $\vec{n_j}$ são, respectivamente, os vetores normais estimados de tal forma que x_j está próximo de x_i e $\vec{n_i}$ e $\vec{n_j}$ são aproximadamente paralelos. Sucintamente, a solução procura corrigir a orientação de um vetor $\vec{n_i}$ comparando-o com outro vetor $\vec{n_j}$, cuja orientação já foi corrigida pelo algoritmo. Para isto bastaria calcular o produto escalar entre os vetores $\vec{n_i}$ e $\vec{n_j}$ e avaliar se o resultado é positivo (mais próximo de 1) ou negativo (mais próximo de -1). Sabendo que a orientação do vetor $\vec{n_j}$ é consistente, caso o resultado for negativo, o vetor $\vec{n_i}$ possui orientação oposta ao vetor $\vec{n_j}$ e, portanto, é corrigida fazendo $\vec{n_i} = -\vec{n_i}$. Caso a orientação seja positiva, ambos os vetores são consistentes (HOPPE et al., 1992).

Para utilizar o critério para correção da orientação dos vetores, é necessário uma abordagem para identificar a relação de proximidade entre os pontos e a relação de paralelismo entre os vetores. O algoritmo constrói um grafo, denominado ***grafo de Riemann***, onde cada vértice v_i deste grafo está associado ao centróide o_i do plano tangente ao ponto x_i. Cada aresta $e = (i, j)$ neste grafo indica que os centróides o_i e o_j estão geometricamente próximos. Para construir as arestas este grafo é realizado o seguinte procedimento: para cada centróide o_i, busque os k pontos centróide mais próximos de o_i. Em seguida, crie uma aresta entre o vértice associado ao centróide o_i e ao vértice associado a cada centróide o_j próximo a o_i. A relação de paralelismo entre os vetores normais $\vec{n_i}$ e $\vec{n_j}$ das centróides o_i e o_j é definida acrescentando um peso w_e na aresta $e = (i, j)$ como sendo $w_e = 1 - \left|\langle \vec{n_i}, \vec{n_j} \rangle\right|$. Com esta fórmula, quanto menor for o peso, maior a relação de paralelismo entre os dois vetores (HOPPE et al., 1992).

Hoppe et al. (1992) procura resolver o critério de proximidade entre os pontos construíndo um grafo interligando os pontos com seus k vizinhos mais próximos. O autor também procura solucionar o critério de paralelismo entre vetores normais de pontos

próximos adicionando um peso em cada aresta do grafo de tal forma que pode-se extrair esta informação obtendo arestas de menor peso. Após estes passos, a abordagem de Hoppe et al. (1992) calcula a **árvore geradora mínima** deste grafo. Através desta árvore, o algoritmo identifica a ordem de propagação da correção da orientação dos vetores. Esta propagação se inicia com o ponto x_i cuja coordenada z é a maior dentre o conjunto de pontos. O vetor normal associado a este ponto é corrigido orientando-o para o lado positivo do eixo z. A propagação é realizada realizando uma **busca em profundidade** na árvore geradora mínima extraída do grafo, onde cada vetor $\vec{n_i}$ do ponto representado pelo vértice visitado v_i é corrigido pelo vetor $\vec{n_j}$ do ponto associado ao vértice pai de v_i na árvore. A correção é realizada observando o produto escalar entre $\vec{n_i}$ e $\vec{n_j}$, sabendo que o vetor $\vec{n_j}$ já foi corrigido pelo algoritmo (HOPPE et al., 1992).

No entanto, a abordagem proposta por Hoppe et al. (1992) não é capaz de garantir consistência da orientação dos vetores normais em determinados cenários. Isto pode ser observado pela publicação de König e Gumhold (2009), onde são comparas as soluções propostas por Hoppe et al. (1992) e Xie et al. (2003) para extração de vetores normais e consistência na orientação por propagação, além de propor um método para correção da orientação dos vetores normais.

Segundo König e Gumhold (2009), as soluções propostas por Hoppe et al. (1992) e Xie et al. (2003) seguem procedimentos semelhantes, tais como extração do vetor normal através da análise de componentes principais, construção do grafo de Riemann, extração da árvore geradora mínima e correção da orientação. Logo, König e Gumhold (2009) concentra sua análise em dois cálculos necessários para o algoritmo, os quais, para os autores, são:

- medir o grau de desconfiança u entre dois vetores normais $\vec{n_i}$ e $\vec{n_j}$ incidentes. Esta medida auxilia o algoritmo a escolher quais vetores possuem maior confiança para ser realizada a propagação da orientação dos vetores. Este valor é atribuído como peso para a aresta e_{ij} no grafo de Riemann, definida quando o ponto p_i pertence aos k vizinhos mais próximos de p_j e vice-versa;
- definir um critério f para a correção da orientação de um vetor normal $\vec{n_i}$ comparado a um outro vetor normal $\vec{n_j}$, cuja orientação já foi corrigida pelo algoritmo.

Devido ao fato de Hoppe et al. (1992) supor que vetores normais de pontos próximos são aproximadamente paralelos, o critério para correção da orientação para dois vetores normais $\vec{n_i}$ e $\vec{n_j}$ é definido como $f_{Hoppe} = \langle \vec{n_i}, \vec{n_j} \rangle < 0$; assim como a medida de desconfiança é definida como $u_{Hoppe} = 1 - \left| \langle \vec{n_i}, \vec{n_j} \rangle \right|$. Considerando a suposição de Hoppe et al. (1992), este critério de correção observa apenas o sentido de dois vetores normais de pontos vizinhos (KÖNIG; GUMHOLD, 2009).

Contudo, as condições definidas por Hoppe et al. (1992) não consegue propagar a orientação quando dois vetores normais estão em lados distintos de uma região aguda (KÖNIG; GUMHOLD, 2009). A Figura 22 exemplifica o processo de propagação em uma região aguda. Observando os dois pontos vizinhos na extremidade da região aguda é possível notar como a correção da orientação dos vetores normais pode inclusive propagar erros para outros vetores normais.

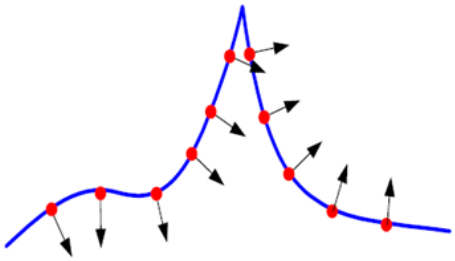

Figura 22 – Abordagem de Hoppe aplicada a uma região aguda
Fonte: Xie et al. (2003)

Para contornar tais problemas, é apresentado por Xie et al. (2003) um outro critério de correção. Sejam p_i e p_j pontos vizinhos de forma que exista uma aresta e_{ij} conectando estes pontos no grafo de Riemann e $\vec{n_i}$ e $\vec{n_j}$ os vetores normais dos pontos p_i e p_j, respectivamente, no qual $\vec{n_i}$ já foi corrigido pelo algoritmo. Em seu novo critério, o vetor $\vec{n_i}$ é refletido em um plano ortogonal a aresta e_{ij}, formando o vetor $\vec{n_i}$'. Portanto é calculado o critério de Hoppe et al. (1992) utilizando os vetores $\vec{n_i}$' e $\vec{n_j}$ (KÖNIG; GUMHOLD, 2009). A Figura 23 exemplifica geometricamente o este critério de correção.

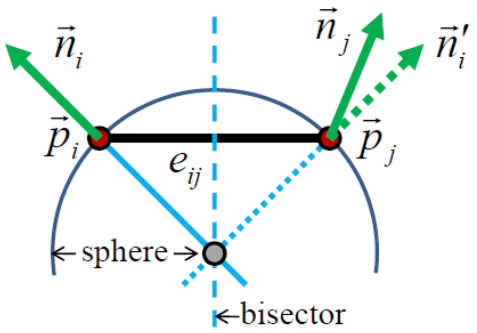

Figura 23 – Critério de correção da orientação de vetores normais de Xie
Fonte: König e Gumhold (2009)

A medida de desconfiança de Xie et al. (2003) é expresso de forma semelhante ao definido por Hoppe et al. (1992) como sendo $u_{Xie} = \langle \vec{n_i'}, \vec{n_j} \rangle$. O critério de inversão de Xie et al. (2003) f_{Xie} pode ser expressa matematicamente pelas seguintes equações König e Gumhold (2009):

$$f_{Xie} = \langle \vec{n_i'}, \vec{n_j} \rangle < 0 \tag{4.4}$$

$$\vec{n_i}' = \vec{n_i} - 2\langle \hat{e}_{ij}, \vec{n_i} \rangle \hat{e}_{ij} \tag{4.5}$$

$$\hat{e}_{ij} = (p_i - p_j)/||p_i - p_j|| \tag{4.6}$$

A solução de König e Gumhold (2009) amplia a solução proposta por Xie et al. (2003) ao permitir variação da curvatura e ao calcular um **plano de referência** a fim de previnir problemas com regiões agudas. Para isto, são utilizadas quatro tipos de **curvas de Hermite** para quatro possíveis grupos considerados pelo algoritmo, conforme mostrado na Figura 24. Tais curvas são definidas em um plano de referência construído a partir dos vetores normais $\vec{n_i}$, $\vec{n_j}$ e pelo vetor unitário $\hat{e}_{ij}$. Cada grupo contém duas **curvas vermelhas** e duas **curvas azuis** (KÖNIG; GUMHOLD, 2009).

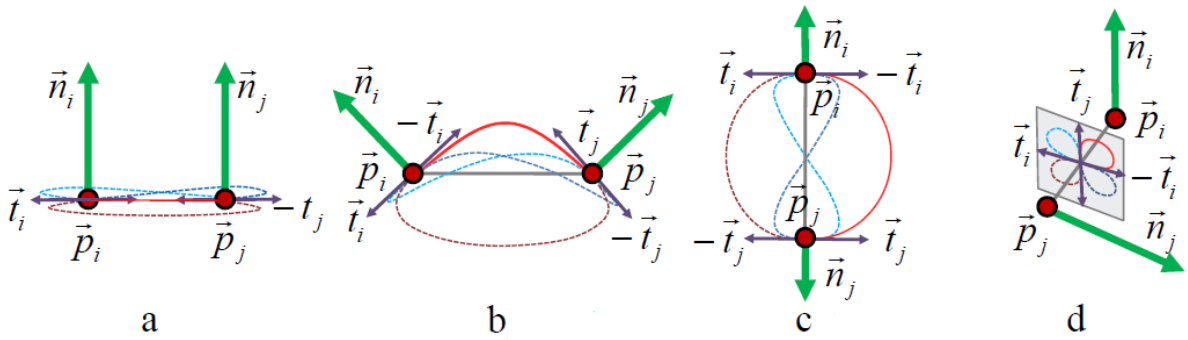

Figura 24 – Os quatro possíveis grupos de curvas de Hermite utilizados por König e Gumhold

Fonte: König e Gumhold (2009)

Dado que a orientação de $\vec{n_j}$ já foi corrigida, o critério para inversão da orientação do vetor $\vec{n_j}$ requer calcular a **complexidade** das quatro curvas de Hermite em um plano de referência formado por estes vetores e o vetor $\hat{e}_{ij}$. Com a complexidade calculada, compara a curva vermelha de menor complexidade e a curva azul de maior complexidade e inverte $\vec{n_j}$ se a complexidade da curva azul for a menor. Seja C_{red} e C_{blue} o valor da menor complexidade entre as curvas vermelhas e azuis, respectivamente. O critério de inversão f_{Konig} e a medida de desconfiança u_{Konig} podem ser expressos matematicamente como sendo (KÖNIG; GUMHOLD, 2009):

$$f_{Konig} = C_{blue} < C_{red} \tag{4.7}$$

$$u_{Konig} = \frac{min\{C_{blue}, C_{red}\}}{max\{C_{blue}, C_{red}\}} \tag{4.8}$$

Para obtem o plano de referência, é calculado a direção de seu vetor normal $\vec{n}_{ref}$ a partir dos vetores $\vec{n_i}$, $\vec{n_j}$ e $\hat{e}_{ij}$. Caso os três sejam paralelos (Figura 24c), o vetor normal $\vec{n}_{ref}$

é definido como qualquer vetor ortogonal aos vetores $\vec{n_i}$, $\vec{n_j}$ ou $\hat{e}_{ij}$ (KÖNIG; GUMHOLD, 2009). Um vetor ortogonal pode ser obtido aplicando uma matriz de rotação, rotacionando o vetor em 90º, conforme explicado por (WEISSTEIN, 1994). Caso os três vetores $\vec{n_i}$, $\vec{n_j}$ e $\hat{e}_{ij}$ formarem um plano (Figura 24a), o vetor normal $\vec{n}_{ref}$ é o mesmo vetor deste plano (KÖNIG; GUMHOLD, 2009).

O último caso a ser analisado é quando os três vetores $\vec{n_i}$, $\vec{n_j}$ e $\hat{e}_{ij}$ são linearmente independentes (Figura 24b e 24d). Neste caso, o plano de referência $\vec{n}_{ref}$ é calculado como sendo o produto vetorial entre $\vec{n_i}$ e $\vec{n_j}$. Contudo, caso o vetor $\vec{n_j}$ for aproximadamente paralelo a $\vec{n_i}$, o vetor $\vec{n}_{ref}$ deveria ser ortogonal ao plano formado por $\hat{e}_{ij}$. Por este motivo é calculado mais um vetor normal do plano de referência, conforme equação abaixo (KÖNIG; GUMHOLD, 2009):

$$\vec{n}_{ref} = \vec{n_i} \times \vec{n_j} + \langle \vec{n_i}, \vec{n_j} \rangle \left(\frac{\vec{n_i} \pm \vec{n_j}}{\left| \vec{n_i} \pm \vec{n_j} \right|} \times \hat{e}_{ij} \right). \tag{4.9}$$

Como as curvas de Hermite requer dois pontos e dois vetores tangente (AZEVEDO; CONCI, 2003), é necessário determinar quais são os vetores tangente. Primeiramente, os vetores normais $\vec{n_i}$ e $\vec{n_j}$ são projetados no plano de referência definido pela normal $\vec{n}_{ref}$. A partir destes vetores normais projetados, são obtidos os vetores tangentes $\vec{t_i}$, $-\vec{t_i}$, $\vec{t_j}$ e $-\vec{t_j}$, onde $\vec{t_i}$ e $\vec{t_j}$ são obtidos por rotacionar $\vec{n_i}$ e $\vec{n_j}$ em 90º no sentido anti-horário e $-\vec{t_i}$ e $-\vec{t_j}$ são obtidos por rotacionar os mesmos vetores em 90º no sentido horário. O módulo de cada vetor é definido como sendo o dobro do comprimento da aresta e_{ij} para que seja invariável a escala. As duas curvas vermelhas são calculadas utilizando os pares de vetores $(\vec{t_i}, \vec{t_j})$ e $(-\vec{t_i}, -\vec{t_j})$. As duas curvas azuis utilizam os pares de vetores $(\vec{t_i}, -\vec{t_j})$ e $(-\vec{t_i}, \vec{t_j})$ (KÖNIG; GUMHOLD, 2009).

A complexidade é obtida ao calcular a mudança do ângulo ocorrida quando a tangente percorre a curva de Hermite. Para uma curva sem pontos de inflexão, é calculado o ângulo entre as tangentes da curva nos pontos p_i e p_j. Se a curva possuir pontos de inflexão, esta é dividida em seus pontos de inflexão, no qual a complexidade da curva é obtida pelo somatório da complexidade de cada parte. Para calcular o ângulo das tangentes nos pontos p_i e p_j, observa-se o sinal da curvatura em p_i ou p_j: caso qualquer um destes for negativo, calcula-se o ângulo no sentido horário; do contrário, calcule-o no sentido anti-horário. Por exemplo, para uma curva de Hermite definida pelos vetores tangente $(-\vec{t_i}, \vec{t_j})$, o cálculo da complexidade da curva observará o ângulo entre os vetores $\vec{t_j}$ e $\vec{t_j}$ (KÖNIG; GUMHOLD, 2009).

Pode-se encontrar um ponto de inflexão de uma curva de Hermite $c_k(t)$ se a derivada primeira $c'_k(t)$ e a derivada segunda $c''_k(t)$ forem paralelas (KÖNIG; GUMHOLD, 2009). Portanto, seja q_1 e q_2 os pontos finais e iniciais da curva e $\vec{m_1}$ e $\vec{m_2}$ os vetores

tangentes a estes pontos, respectivamente. Para Azevedo e Conci (2003), uma curva $c_k(t)$, para $t \in [0, 1]$, pode ser expressa pela seguinte equação (detalhes da Seção 3.8):

$$c_k(t) = (2t^3 - 3t^2 + 1)q_1 + (-2t^3 + 3t^2)q_2 + (t^3 - 2t^2 + t)\vec{m_1} + (t^3 - t^2)\vec{m_2} \tag{4.10}$$

Logo, conforme König e Gumhold (2009), as derivadas $c'_k(t)$ e $c''_k(t)$ são :

$$c'_k(t) = (6t^2 - 6t)q_1 + (-6t^2 + 6t)q_2 + (3t^2 - 4t + 1)\vec{m_1} + (3t^2 - 2t)\vec{m_2} \tag{4.11}$$
$$c''_k(t) = (12t - 6)q_1 + (-12t + 6)q_2 + (6t - 4)\vec{m_1} + (6t - 2)\vec{m_2} \tag{4.12}$$

Para encontrar os pontos de inflexão da curva $c_k(t)$, é construído uma matriz quadrática de segunda ordem A com os vetores $c'_k(t)$ e $c''_k(t)$. Para que estas derivadas sejam paralelas, o determinante da matriz A deve ser igual a zero. Considere $det(u|v)$ o determinante da matriz quadrática de segunda ordem formada por dois vetores bidimensionais u e v arranjados como linhas ou colunas desta matriz. O cálculo do determinante da matriz A pode ser simplificado pelo polinômio $p(t) = at^2 + bt + c$ tal que (KÖNIG; GUMHOLD, 2009):

$$a = det(\vec{m_1}, \vec{v}) \tag{4.13}$$
$$b = 2det(\vec{m_1}, \vec{w}) \tag{4.14}$$
$$c = det(\vec{v}, \vec{w}) \tag{4.15}$$

onde

$$\vec{v} = 6(q_1 - q_2) - 4\vec{m_1} - 2\vec{m_2} \tag{4.16}$$
$$\vec{w} = 3(\vec{m_1} + \vec{m_2}) - 6(q_1 - q_2) \tag{4.17}$$

A abordagem em König e Gumhold (2009) consegue corrigir a orientação dos vértices com maior eficácia em relação às abordagens de Hoppe et al. (1992) e Xie et al. (2003), sendo capaz de resolver problemas como a propagação da orientação em regiões agudas. Entretanto, o algoritmo em König e Gumhold (2009) possui um tempo computacional aproximadamente 10 vezes maior do que as demais abordagens, enquanto Hoppe et al. (1992) e Xie et al. (2003) apresentam entre si praticamente o mesmo tempo computacional. Isto pode ser observado através dos dados fornecidos por König e Gumhold (2009) ao comparar cada abordagem calculando os vetores normais do objeto 3D de um tetraedro, de um tricerátopo e de um conjunto de engrenagens (Figura 25). Estes dados estão contidos na Tabela 4 e na Tabela 5.

Figura 25 – Objetos 3D de um tetraedro, tricerátopo e engrenagens, nesta ordem, em nuvem de pontos
Fonte: König e Gumhold (2009)

Número de falhas na correção da orientação			
Objeto 3D	**Hoppe**	**Xie**	**König e Gumhold**
Tetraedro	2822	2822	0
Tricerátopo	63	5	0
Engrenagem	2064	2139	2025

Tabela 4 – Número de falhas na correção da orientação

Tempo de processamento entre cada abordagem			
	Hoppe	**Xie**	**König e Gumhold**
Tempo para cada aresta	0,023 ms	0,026 ms	0,21 ms
Tetraedro	3,17 s	3,19 s	30,0 s
Tricerátopo	3,02 s	3,40 s	27,9 s
Engrenagem	3,82 s	4,32 s	34,7 s

Tabela 5 – Tempo de processamento entre cada abordagem

4.2 Algoritmo para pré-processamento da malha

Muitos algoritmos envolvendo objetos representados por malhas triangulares consistem em diversas iterações sobre seu conjunto de vértices ou conjunto de arestas. A construção do operador Laplaciano, etapa utilizada para a suavização Laplaciana realizada por Au et al. (2008) e Cao et al. (2010) é um exemplo de um procedimento que requer realizar cálculos avaliando cada vértice da malha triangular. Contudo, existes objetos 3D que utilizam um grande número de vértices e faces para modelar uma série de detalhes em sua forma, como, por exemplo, o objeto 3D *Stanford Armadillo* [1] , o qual contém 106289 vértices e 212574 faces triangulares (Figura 26).

[1] Link para Link para objeto Stanford Armadillo: http://www.prinmath.com/csci5229/OBJ/index.html

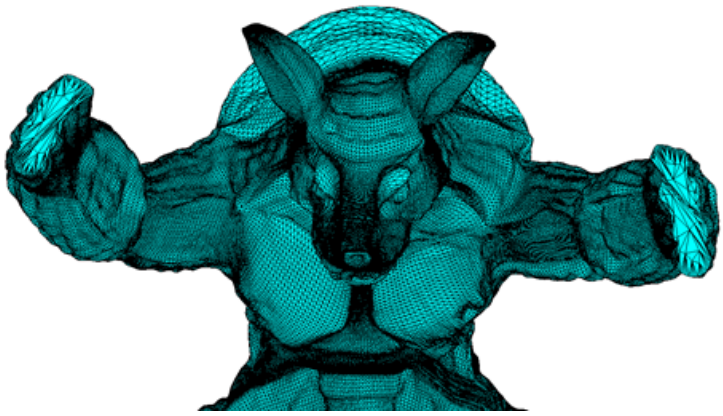

Figura 26 – Plotagem do objeto 3D "Stanford Armadillo" em MATLAB
Fonte: Elaborada pelo autor

Observando a estrutura do esqueleto na Figura 27, nota-se intuitivamente que a esqueletização não depende de detalhes da forma do objeto 3D apresentados na Figura 26. Em cenários em que não é necessário relacionar a malha de entrada com o esqueleto final, realizar um pré-processamento na malha de entrada pode ser útil a fim de reduzir o número de vértices e faces, simplificando o objeto 3D, desde que não altere a geometria e a topologia do objeto 3D original.

Figura 27 – Exemplo de esqueletização do objeto "Stanford Armadillo"
Fonte: Jiang et al. (2013)

O algoritmo proposto por Valette e Chassery (2004) apresenta uma forma de simplificar um objeto tridimensional sem afetar sua geometria e topologia (Figura 28). Sua solução utiliza **diagramas de Voronoi centroidais**, que são diagramas de Voronoi cuja semente localiza-se no centro da região de Voronoi. Seja $Z = \{z_i | i = 1, \ldots, n\}$ os pontos sementes do diagrama de Voronoi, $\rho(x)$ uma função de densidade e V_i a região de Voronoi correspondente à semente z_i. Cada ponto z_i de um diagrama de Voronoi centroidal é definido como o ponto centróide da região V_i, calculado através da seguinte média ponderada contínua (VALETTE; CHASSERY, 2004):

$$z_i = \frac{\int_{V_i} x\rho(x)dx}{\int_{V_i} \rho(x)dx} \tag{4.18}$$

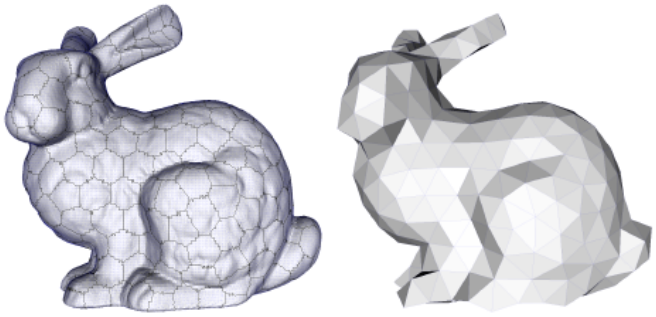

Figura 28 – Objeto 3D simplificado utilizando algoritmo de Valette e Chassery
Fonte: Valette e Chassery (2004) adaptada pelo autor

Conforme Valette e Chassery (2004), o diagrama de Voronoi centroidal também minimiza o somatório E abaixo, no qual quanto menor o valor do somatório mais próximo os n pontos sementes estão de pertencerem ao centro da região de Voronoi V_i:

$$E = \sum_{i=1}^{n} \int_{V_i} \rho(x)|x - z_i|^2 dx \tag{4.19}$$

O algoritmo busca construir um **diagrama de Voronoi centroidal aproximado** que minimize a Equação 4.19 para uma malha triangular M. Para isto é necessario discretizar esta equação. O diagrama de Voronoi deste algoritmo irá construir grupos de faces como regiões. Portanto, cada região V_i será composta por um conjunto não vazio de faces C_j. Cada região deve conter uma ou mais faces e cada face não pode pertencer a mais de uma região. A fronteira de uma região de Voronoi é, portanto, formada por um conjunto de arestas da malha M que a contorna. Logo, a Equação 4.19 pode ser reescrita pela equação a seguir considerando uma região V_i como um conjunto de faces C_j (VALETTE; CHASSERY, 2004):

$$E = \sum_{i=1}^{n} \left(\sum_{C_j \in V_i} \int_{C_j} \rho(x)|x - z_i|^2 dx \right) \tag{4.20}$$

Considere que a função $\rho(x)$ seja uniforme. Portanto, o resultado da integral $\rho_j = \int_{C_j} \rho(x)dx$ resulta na **área** da face C_j. Seja γ_j o ponto **centróide** da face C_j. O algoritmo aproxima a face C_j pelo centróide γ_j e peso ρ_j. Com isto, a integral presente na Equação 4.20 é simplificada para a seguinte equação (VALETTE; CHASSERY, 2004):

$$E = \sum_{i=1}^{n} \left(\sum_{C_j \in V_i} \rho_j |\gamma_j - z_i|^2 \right) \tag{4.21}$$

O ponto centróide γ_j da face C_j é calculado como sendo (VALETTE; CHASSERY,

2004):

$$\gamma_j = \frac{\int_{C_j} x dx}{\int_{C_j} dx} \tag{4.22}$$

Entretanto, o algoritmo propõe construir um diagrama de Voronoi centroidal aproximado e, portanto, não saberá a priori a localização dos pontos sementes z_i. Por definição, sabe-se que, para este diagrama, z_i é o ponto central da região de Voronoi V_i. Por isto, z_i é calculado como o ponto centróide $\overline{\gamma_i}$ de V_i, o que é obtido calculando a média dos pontos centróides das faces C_j pertencentes a V_i ponderada por ρ_j. Este ponto é calculado como sendo (VALETTE; CHASSERY, 2004):

$$\overline{\gamma_i} = \frac{\sum_{C_j \in V_i} \rho_j \gamma_j}{\sum_{C_j \in V_i} \rho_j} \tag{4.23}$$

Então, é encontrada uma equação discreta que aproxima a Equação 4.19 (VALETTE; CHASSERY, 2004):

$$F = \sum_{i=1}^{n} \left(\sum_{C_j \in V_i} \rho_j |\gamma_j - \overline{\gamma_i}|^2 \right) \tag{4.24}$$

A Equação 4.24 pode ser reescrita ao substituir $\overline{\gamma_i}$ pela Equação 4.23 (VALETTE; CHASSERY, 2004):

$$F = \sum_{i=1}^{n} \left(\sum_{C_j \in V_i} \rho_j |\gamma_j|^2 - \frac{|\sum_{C_j \in V_i} \rho_j \gamma_j|^2}{\sum_{C_j \in V_i} \rho_j} \right) \tag{4.25}$$

Apesar da Equação 4.25 ser uma aproximação discreta da Equação 4.19, a definição do ponto centróide γ_j e da área ρ_j da face triangular C_j foi definida através de um cálculo contínuo em Valette e Chassery (2004). Os valores para γ_j e ρ_j podem ser aproximados através de um cálculo discreto. Seja u, v e w os vértices da face triangular C_j. O ponto centróide de C_j é calculado encontrando o seu baricentro de C_j pela seguinte equação (MELO, 2015):

$$\gamma_j = \frac{u + v + w}{3} \tag{4.26}$$

Para os vértices u, v e w da face C_j, a área da face triangular C_j é encontrado através da seguinte fórmula (STEINBRUCH; WINTERLE, 1987a):

$$\rho_j = \frac{|\vec{uv} \times \vec{uw}|}{2} \tag{4.27}$$

Seja M uma malha triangular e n a quantidade total de regiões de Voronoi. O algoritmo inicia selecionando n faces distintas aleatoriamente, criando n regiões de Voronoi para cada face selecionada. As demais faces não selecionadas inicialmente são adicionadas a uma região denominada **região nula**. Em seguida é obtido o conjunto das **arestas de borda** destas regiões, nas quais, neste passo inicial, correspondem às arestas de cada uma das n faces iniciais (VALETTE; CHASSERY, 2004).

O algoritmo itera sobre este conjunto de arestas de borda. Para cada aresta de borda e é analisado as faces triangulares $C_i \in V_i$ e $C_j \in V_j$ adjacentes a esta aresta, onde V_i e V_j são regiões de Voronoi distintas. Caso V_i for a região nula, C_i é adicionado à região V_j. Caso V_j for a região nula, C_j é adicionado à região V_i. Caso ambas as regiões não forem a região nula, é realizado um teste sobre C_i e C_j a fim de minimizar a energia F expressa pela Equação 4.25 (VALETTE; CHASSERY, 2004).

Para duas faces adjacentes $C_m \in V_k$ e $C_n \in V_l$, onde V_k e V_l são duas regiões distintas, é realizado um teste local para verificar qual seria a situação que minimizaria a energia F. Para isto, são observados três cenários que são mostrados na Figura 29 (VALETTE; CHASSERY, 2004):

- F_{init}: configuração inicial, onde $C_m \in V_k$ e $C_n \in V_l$ (Figura 29 a);
- F_1 : configuração onde V_k expande, na qual $C_m \in V_k$ e $C_n \in V_k$ (Figura 29 b);
- F_2 : configuração onde V_l expande, na qual $C_m \in V_l$ e $C_n \in V_l$ (Figura 29 c).

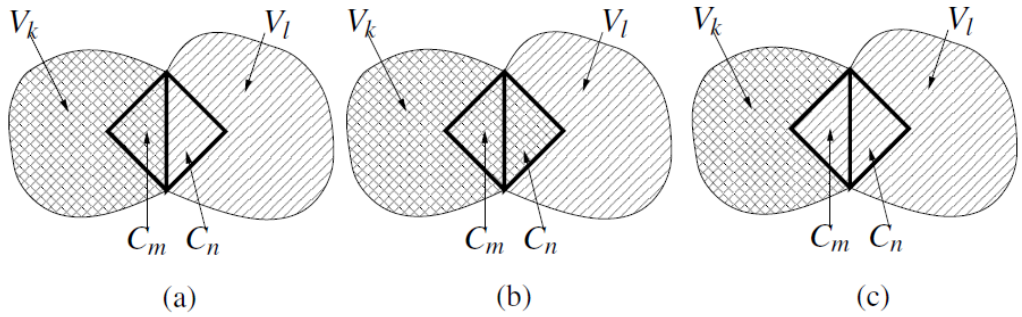

Figura 29 – Os cenários para teste de minificação na construção de um diagrama de Voronoi centroidal aproximado
Fonte: Valette e Chassery (2004) adaptada pelo autor

Para o teste, é calculado a energia F para os casos em F_{init}, F_1 e F_2. Aquele que minimize a Equação 4.25 é a configuração a ser aplicada pelo algoritmo. Isto é, caso F_2 possuir o menor valor entre F_{init} e F_1, então a face C_m é removida da região V_l e adicionada à região V_l. Como o teste local apenas avalia as regiões V_k e V_l, não há a necessidade de calcular a energia F para todas as n regiões conforme expresso pela Equação 4.25. Para este teste, esta equação pode ser calculada apenas para as regiões V_k

e V_l. Logo, a Equação 4.25 pode ser simplificada para o teste de minificação da seguinte forma (VALETTE; CHASSERY, 2004):

$$L_1 = \sum_{C_j \in V_k} \rho_j |\gamma_j|^2 - \frac{|\sum_{C_j \in V_k} \rho_j \gamma_j|^2}{\sum_{C_j \in V_k} \rho_j} + \sum_{C_j \in V_l} \rho_j |\gamma_j|^2 - \frac{|\sum_{C_j \in V_l} \rho_j \gamma_j|^2}{\sum_{C_j \in V_l} \rho_j} \tag{4.28}$$

Entretanto, os termos $\sum_{C_j \in V_k} \rho_j |\gamma_j|^2$ e $\sum_{C_j \in V_l} \rho_j |\gamma_j|^2$ não são necessários no cálculo, pelo fato da união das regiões V_k e V_l não mudarem nos três cenários analisados. Então, o cálculo é simplificado pela seguinte equação (VALETTE; CHASSERY, 2004):

$$L_2 = -\left(\frac{|\sum_{C_j \in V_k} \rho_j \gamma_j|^2}{\sum_{C_j \in V_k} \rho_j} + \frac{|\sum_{C_j \in V_l} \rho_j \gamma_j|^2}{\sum_{C_j \in V_l} \rho_j} \right) \tag{4.29}$$

A Figura 30 apresenta como funciona o processo de expansão para a construção do diagrama de Voronoi centroidal aproximado utilizando 500 regiões de Voronoi. A Figura 30a apresenta o objeto 3D original, enquanto as demais apresentam o procedimento desde a inicialização (Figura 30b) até o resultado final (Figura 30d). A Figura 30c apresenta o resultado do algoritmo com 2 iterações, enquanto para a Figura 30d foram utilizadas 12 iterações. As partes escuras observadas representam a região nula.

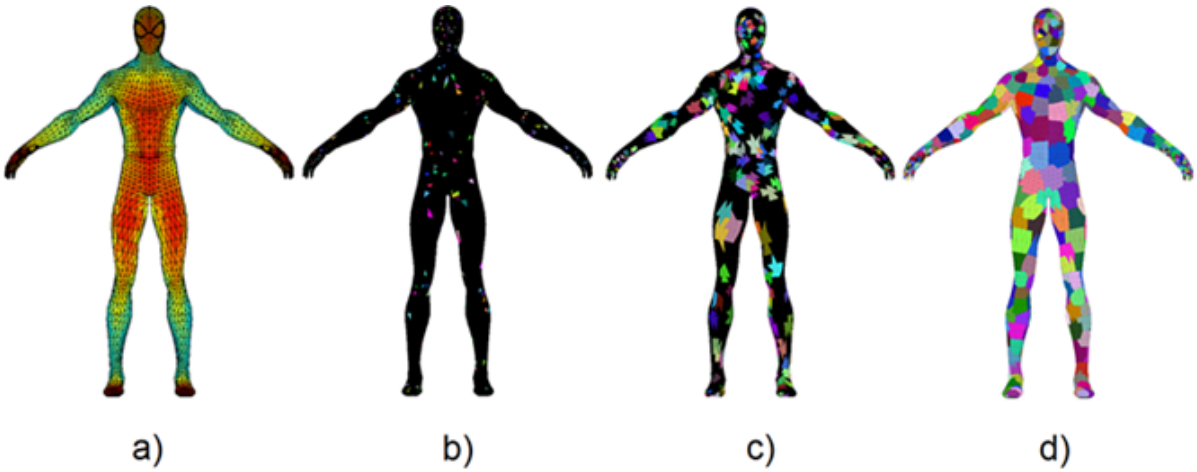

Figura 30 – Etapas para construção de um diagrama de Voronoi centroidal
Fonte: Elaborada pelo autor

Com o diagrama de Voronoi centroidal calculado, o próximo passo é extrair uma malha triangular mais simples. Uma vez que é possível obter uma **triangulação de Delaunay** através de um diagrama de Voronoi, extrair a malha triangular simplificada requer simplesmente construir a triangulação de Delaunay correspondente ao diagrama de Voronoi centroidal previamente calculado. É preciso então definir como definir os vértices e as faces da malha triangular final (VALETTE; CHASSERY, 2004).

Os vértices utilizados pela triangulação são os n pontos sementes de cada região de Voronoi. Em outras palavras, são utilizados os centróides das n regiões de Voronoi para

compor os n vértices da malha triangular final. No entanto, o centróide de uma região frequentemente não é definido na superfície do objeto 3D original, principalmente para regiões côncavas ou convexas (Figura 31). Portanto, é selecionado como vértice para a triangulação final o vértice na superfície (pertencente à malha triangular original) que esteja o mais próximo possível do centróide de uma região de Voronoi (VALETTE; CHASSERY, 2004).

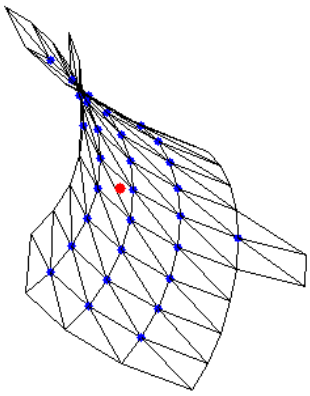

Figura 31 – Cálculo do centróide a partir de uma conjunto de vértices
Fonte: Elaborada pelo autor

Para construir a triangulação, é observado a conectividade entre as regiões do diagrama de Voronoi centroidal. Para isto, pode-se observar os vértices da superfície que pertencem às arestas de borda deste diagrama. Para cada vértice u, verifica se u está presente em três regiões de Voronoi distintas. Caso esta condição seja satisfeita, significa que estas três regiões são vizinhas. Com isto, construa um triângulo com os pontos semente destas três regiões (Figura 32) (VALETTE; CHASSERY, 2004).

Contudo, é bastante comum situações onde em um vértice da superfície u encontram-se quatro ou mais regiões de Voronoi presentes. Em situações onde existem quatro regiões distintas presentes em um vértice u, são criados dois triângulos, conforme Figura 32. Esta abordagem pode ser generalizada. Para m regiões de Voronoi, o algoritmo irá construir $m - 2$ triângulos (VALETTE; CHASSERY, 2004).

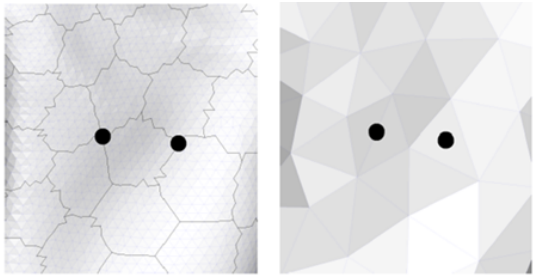

Figura 32 – Construção da triangulação a partir de um diagrama de Voronoi centroidal
Fonte: Valette e Chassery (2004)

É preciso observar que este algoritmo de simplificação não é recomendado para

uma malha triangular que contém pouca quantidade de vértices e faces, uma vez que a simplificação pode distorcer a geometria do objeto 3D original. A Figura 33 mostra o resultado deste algoritmo para uma malha triangular 3D com 512 vértices e 1024 faces. É possível notar inclusive a redução do volume no objeto simplificado.

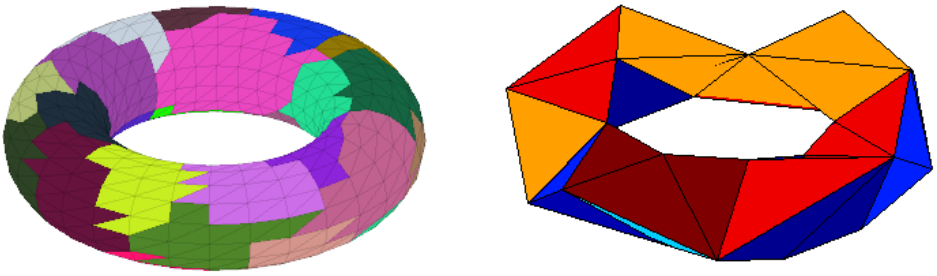

Figura 33 – Simplificação de uma malha triangular 3D com poucos vértices e faces
Fonte: Elaborada pelo autor

4.3 Esqueletização de malhas por suavização Laplaciana

Au et al. (2008) apresenta um algoritmo para a extração do esqueleto de um objeto 3D representado por malhas triangulares. De forma sucinta, sua proposta corresponde em iterativamente contrair a malha tridimensional, até que esta possuir um volume praticamente igual a zero.

Em cada iteração, a contração é realizada por minimizar uma fórmula que envolve um **termo de contração** e um **termo de atração**. Enquanto o termo de contração é calculado utilizando implicitamente o **operador discreto de Laplace**, o termo de atração utiliza as informações geométricas dos vértices da malha original para que durante as iterações a forma contraída mantenha características geométricas no objeto original. Para cada um dos termos é atribuído um peso que é ajustado a cada iteração (AU et al., 2008).

Com o decorrer das iterações, as conexões entre os vértices encontradas na malha original são mantidas, sendo alterado apenas a posição dos vértices devido à contração da malha. Portanto, o processo de contração converte a malha original para uma malha com volume aproximadamente zero, mantendo toda a conectividade da malha original (Figura 34). Por esta razão, o algoritmo propõe um processo de **cirurgia de conectividade**, o qual remove o excesso de arestas presentes na malha, removendo todas as faces da malha contraída e mantendo os vértices e arestas que representam a geometria e a topologia da objeto original, o que resulta no esqueleto final (AU et al., 2008).

Durante a cirurgia de conectividade é construído um mapeamento onde é associado um conjunto de vértices da malha original a cada nó do esqueleto. Este mapeamento é utilizado na etapa de refinamento, a qual possui o objetivo de posicionar cada nó do esqueleto ao centro da região da malha associada a ele (AU et al., 2008).

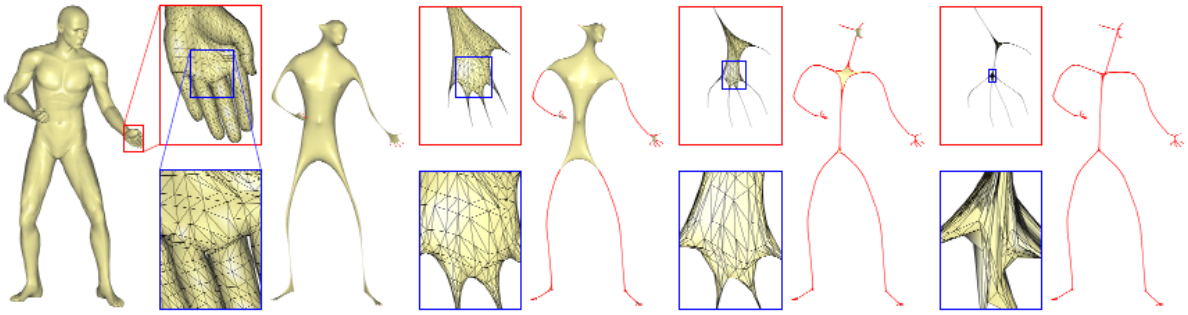

Figura 34 – Processo de contração da malha por suavização Laplaciana
Fonte: Au et al. (2008)

Seja $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ uma malha triangular, onde $\mathbf{V_{n\times 3}} = [v_1^T, v_2^T, \ldots, v_i^T, \ldots, v_n^T]^T$ é a matriz de vértices e $\mathbf{E}$ o conjunto das arestas que representam a conexão entre dois vértices na malha. Cada i-ésima linha da matriz $\mathbf{V}$ corresponde à posição do vértice $v_i = [x_{v_i}\ y_{v_i}\ z_{v_i}]$. O processo de contração é baseado na resolução da equação discreta de Laplace $\mathbf{LV'} = \mathbf{0}$, onde $\mathbf{V'}$ é a matriz de vértices suavizada e $\mathbf{L}$ é o operador de Laplace definido como uma matriz $n \times n$ onde (AU et al., 2008):

$$\mathbf{L}_{ij} = \begin{cases} \omega_{ij} = \cot\alpha_{ij} + \cot\beta_{ij} & \text{se } (i,j) \in \mathbf{E} \\ \sum_{(i,k)\in\mathbf{E}} -\omega_{ik} & \text{se } i = j \\ 0 & \text{caso contrário} \end{cases} \tag{4.30}$$

Na Equação 4.30, α_{ij} e β_{ij} são os ângulos opostos à aresta (i, j), conforme Figura 35 (AU et al., 2008).

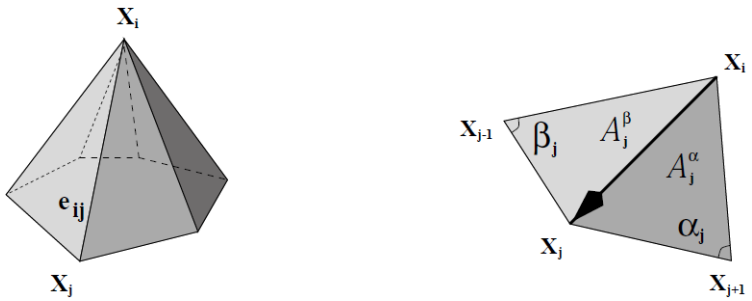

Figura 35 – Ângulos para cálculo do operador de Laplace utilizando cotangente
Fonte: Desbrun et al. (1999)

Seja v_k e v_l os vértices dos ângulos α_{ij} e β_{ij}, respectivamente. Pode-se observar que α_{ij} é formado pelos vetores $\overrightarrow{v_k v_i}$ e $\overrightarrow{v_k v_j}$ e β_{ij} é formado pelos vetores $\overrightarrow{v_l v_i}$ e $\overrightarrow{v_l v_j}$. Com isto, a cotangente pode ser facilmente calculada utilizando relações trigonométricas entre ângulos formados por vetores, conforme visto na seção 3.2.6 (MEYER et al., 2002). Desta

forma, o peso $\omega_{ij} = \cot \alpha_{ij} + \cot \beta_{ij}$ pode ser reescrito como sendo:

$$\omega_{ij} = \frac{\langle \overrightarrow{v_k v_i}, \overrightarrow{v_k v_j} \rangle}{\left| \overrightarrow{v_k v_i} \times \overrightarrow{v_k v_j} \right|} + \frac{\langle \overrightarrow{v_l v_i}, \overrightarrow{v_l v_j} \rangle}{\left| \overrightarrow{v_l v_i} \times \overrightarrow{v_l v_j} \right|} \tag{4.31}$$

Contudo, é necessário acrescentar restrições à equação de Laplace $\mathbf{LV'} = \mathbf{0}$, uma vez que esta não admite uma única solução para $\mathbf{V'}$ que $\mathbf{L}$ é uma matriz singular. Cada linha da equação de Laplace fornece as restrições de contração. Além destas, devem ser acrescentadas as restrições de atração, a qual considera os vértices de $\mathbf{V}$. Para cada uma das restrições são atribuídos pesos que são ajustados a cada iteração. Portanto, o processo de contração definido pelo algoritmo consiste em resolver o seguinte sistema linear sobredeterminado (AU et al., 2008).

$$\begin{bmatrix} \mathbf{W_L L} \\ \mathbf{W_H} \end{bmatrix} \mathbf{V'} = \begin{bmatrix} 0 \\ \mathbf{W_H V} \end{bmatrix} \tag{4.32}$$

A Equação 4.32 estende a equação de Laplace $\mathbf{LV'} = \mathbf{0}$ acrescentando restrições de contração e atração. Os pesos de contração e atração são definidos por matrizes diagonais de ordem n, as quais são, respectivamente, $\mathbf{W_L}$ e $\mathbf{W_H}$. Considere $\mathbf{W_{L,i}}$ e $\mathbf{W_{H,i}}$ a i-ésima diagonal das matrizes de pesos de contração e atração, nesta ordem. A Equação 4.32, por ser um sistema linear sobredeterminado, pode ser resolvido através da minimização da seguinte equação (AU et al., 2008):

$$\|\mathbf{W_L L V'}\|^2 + \sum_i \mathbf{W_{H,i}^2} \|\mathbf{v'_i} - \mathbf{v_i}\|^2 \tag{4.33}$$

O processo de contração é realizado por meio de diversas iterações, onde em cada t-ésima iteração os vértices $\mathbf{V^t}$ são contraídos para os vertices $\mathbf{V^{t+1}}$, até convergir para uma estrutura semelhante ao esqueleto. Para cada t-ésima iteração, por consequência da contração dos vértices $\mathbf{V^t}$, também é atualizado o operador de Laplace $\mathbf{L^t}$ assim como as matrizes de peso $\mathbf{W_L^t}$ e $\mathbf{W_H^t}$. O algoritmo inicializa $\mathbf{W_L^0} = 10^{-3}\sqrt{A}$ e $\mathbf{W_H^0} = 1.0$ para toda a diagonal, onde A corresponde à média da área das faces. Em seguida, enquanto a volume da malha contraída for maior que um limiar $\epsilon_{vol} = 10^{-6}$, a iteração é realizada através dos seguintes passos, iniciando com t igual a 0 (AU et al., 2008):

1. Calcular o operador de Laplace $\mathbf{L^t}$ baseado nos vértices de $\mathbf{V^t}$ utilizando a Equação 4.30;

2. Contrair os vértices da malha utilizando a Equação 4.32 para obter $\mathbf{V^{t+1}}$. Isto

significa resolver o sistema linear

$$\begin{bmatrix} \mathbf{W_L^t L} \\ \mathbf{W_H^t} \end{bmatrix} \mathbf{V'^{t+1}} = \begin{bmatrix} 0 \\ \mathbf{W_H^t V^t} \end{bmatrix};$$

3. Atualizar os pesos de contração e atração como sendo $\mathbf{W_L^{t+1}} = s_L \mathbf{W_L^t}$ e $\mathbf{W_{H,i}^{t+1}} = \mathbf{W_{H,i}^t} \sqrt{A_i^0 / A_i^t}$. O parâmetro s_L é utilizado para controlar a velocidade da contração, sendo $s_L = 2.0$ definido empiricamente por Au et al. (2008). Os valores de A_i^0 e A_i^t são definidos como a área da vizinhança do i-ésimo vértice da malha original e na t-ésima iteração, nesta ordem. Esta área é definida como a soma da área de todas as faces que contém o vértice i.

Apesar do processo de contração da malha tridimensional resultar em uma malha contraída visivelmente semelhante a um esqueleto, a quantidade de vértices e a conexão são as mesmas da malha original (Figura 34). Para isto é realizado um procedimento para converter a malha contraída no esqueleto final, chamado de **cirurgia de conectividade**. O algoritmo consiste basicamente em realizar o **colapso de arestas** na malha contraída, removendo todas as faces, de modo que a geometria e topologia seja preservada no esqueleto final. Além disto, é utilizado uma função de custo capaz de medir o quanto a operação de colapso de uma aresta da malha influencia na geometria e topologia do esqueleto final. Portanto, o processo de cirurgia de conectividade é realizado iterativamente, onde para cada iteração o colapso de arestas é executado para a aresta de custo mínimo (AU et al., 2008).

O colapso de arestas é feito por meio de um colapso de semi-arestas, para simplificação. O colapso de semi-aresta $(i \rightarrow j)$ une o vértice i com o vértice j em um único vértice de posição $\tilde{v}_j$. Tal operação remove as faces incidentes à aresta (i, j), conforme observado na Figura 36. O algoritmo não realiza a operação em uma aresta (i, j) se existe um vértice k adjacente aos vértices i e j tal que (i, j, k) não é uma face na malha. (AU et al., 2008).

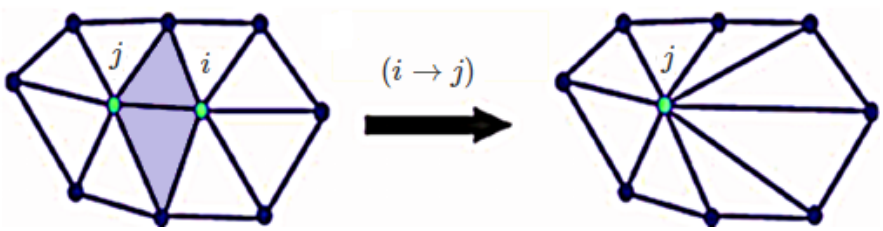

Figura 36 – Exemplo de colapso de semi-aresta
Fonte: Mamou e Ghorbel (2009) adaptada pelo autor

O custo total de uma aresta é uma soma ponderada de dois termos: um relacionado ao **custo da forma** e outro relacionado ao **custo da amostragem**. Considere (i, j) uma

aresta da malha. Seja $\mathcal{F}_a(i,j)$ e $\mathcal{F}_b(i,j)$ o custo da forma e o custo da amostragem, respectivamente, para esta aresta. Seja w_a e w_b os pesos para o custo da forma e para o custo de amostragem, respectivamente, definidos empiricamente como sendo $w_a = 1.0$ e $w_b = 0.1$. O custo total $\mathcal{F}(i,j)$ da aresta (i,j) é definido como sendo (AU et al., 2008):

$$\mathcal{F}(i,j) = w_a\mathcal{F}_a(i,j) + w_b\mathcal{F}_b(i,j) \tag{4.34}$$

Para o **custo da forma** é calculado o somatório das distâncias quadradas entre um vértice e as faces incidentes a ele. Deste modo, é possível medir o quanto uma operação de colapso de aresta afetaria a geometria do esqueleto. Como a cirurgia de conectividade é aplicada em uma malha contraída e, por isto, as faces praticamente não possuem área, esta métrica é simplificada. Para isto é necessário definir para toda aresta (i,j) a matriz $\mathbf{K}_{ij}$, onde a distância entre um ponto $\mathbf{p}$ a aresta (i,j) seja calculada como sendo $\mathbf{p^T}(\mathbf{K_{ij}^T}\mathbf{K_{ij}})\mathbf{p}$. Seja $\vec{a} = (a_x, a_y, a_z)$ o vetor normalizado da aresta (i,j) e $b = (b_x, b_y, b_z) = a \times \tilde{v}_i$. A matriz $\mathbf{K}_{ij}$ é definida como sendo (AU et al., 2008):

$$\mathbf{K}_{ij} = \begin{bmatrix} 0 & -a_z & a_y & -b_x \\ a_z & 0 & -a_x & -b_y \\ -a_y & -a_x & 0 & -b_z \end{bmatrix} \tag{4.35}$$

Portanto, o custo calculado para o vértice i é definido pelo somatório da distâncias quadradas do ponto às arestas adjacentes a (i,j), conforme a seguinte equação (AU et al., 2008):

$$F_i(\mathbf{p}) = \mathbf{p}^T \sum_{(i,j)\in\mathbf{E}} (\mathbf{K}_{ij}^T\mathbf{K}_{ij})\mathbf{p} = \mathbf{p}^T\mathbf{Q}_i\mathbf{p} \tag{4.36}$$

Através da Equação 4.36 é calculado o custo da forma (dado pela função $\mathcal{F}_a(i,j)$) para o colapso de aresta $(i \to j)$. Sendo $\tilde{v}_j$ a posição do vértice resultante do colapso de arestas, o custo da forma $\mathcal{F}_a(i,j)$ é dado pela seguinte equação (AU et al., 2008):

$$\mathcal{F}_a(i,j) = F_i(\tilde{v}_j) + F_i(\tilde{v}_i) \tag{4.37}$$

Pode-se afirmar que quanto menor o valor de $\mathcal{F}_a(i,j)$ (Equação 4.37), menor será a distorção causada pelo colapso de aresta $(i \to j)$ na geometria do esqueleto. Por este motivo o algoritmo seleciona a aresta de menor custo durante o processo de cirurgia de conectividade. Apesar do custo da forma conseguir manter a forma da malha contraída, este pode resultar em arestas muito longas no esqueleto em regiões retas (como regiões geralmente cilíndricas, por exemplo). Isto pode causar perda de informações relevantes

para o mapeamento entre um nó do esqueleto para um conjunto de vértices da malha original (AU et al., 2008).

Para impedir a criação de arestas muito extensas no esqueleto é calculado o custo da amostragem. Analisando a Figura 36 nota-se que a operação de colapso de arestas $(i \rightarrow j)$ modifica a largura de cada aresta adjacente ao vértice i. O custo de amostragem, portanto, é a distância total que as arestas adjacentes a i deslocam ao realizar o colapso de arestas $(i \rightarrow j)$. Sendo $\tilde{\mathbf{E}}$ o conjunto de arestas da malha contraída após a operação $(i \rightarrow j)$, este custo é calculado como sendo (AU et al., 2008):

$$\mathcal{F}_b(i,j) = \|\tilde{v}_i - \tilde{v}_j\| \sum_{(i,k)\in\tilde{\mathbf{E}}} \|\tilde{v}_i - \tilde{v}_k\| \tag{4.38}$$

Durante o processo de cirurgia de conectividade, todo colapso de arestas é registrado a fim de construir o mapeamento entre um nó do esqueleto e vértices da malha original. Isto significa que cada nó k presente no esqueleto é mapeado a um conjunto de vértices Π_k da malha original que foi contraído para o nó k. Este mapeamento é utilizado na etapa de **refinamento** do esqueleto, onde cada nó k do esqueleto é movido ao centro aproximado da região da malha Π_k. Este procedimento é realizado devido à possibilidade da forma da malha contraída estar deslocada do seu centro ou até mesmo estar fora da malha original (AU et al., 2008).

O refinamento do esqueleto possui o objetivo de mover cada nó do esqueleto ao centro da região da malha associada a ele. A Figura 37 apresenta, respectivamente, o esqueleto após a cirurgia de conectividade (à esquerda), as regiões da malha mapeadas para cada nó do esqueleto (centro) e o esqueleto final após o refinamento (à direita). A Figura 37 também destaca alguns nós do esqueleto que estão fora do centro de sua região da malha e suas posições corrigidas após o processo de refinamento(AU et al., 2008).

Figura 37 – Cirurgia de conectividade, mapeamento e refinamento do esqueleto
Fonte: Au et al. (2008)

Seja k o nó do esqueleto e Π_k a região da malha associada ao nó k. Uma vez que os vértices da borda de uma região são contraídos aproximadamente na mesma posição, o deslocamento entre o nó k e o centro da região é calculado através do deslocamento médio dos vértices da borda da região Π_k. Seja v_i a posição do vértice i da malha original e $\tilde{v}_i$ a posição deste vértice no final do processo de contração da malha. Seja também $\mathcal{S}_j$ os

índices dos vértices da borda j e $l_{j,i}$ o comprimento total das duas arestas pertencentes a borda j e adjacentes ao vértice i. O deslocamento médio d_j para a borda j, calculado durante o processo de contração, é definido como sendo (AU et al., 2008):

$$d_j = \frac{\sum_{i \in \mathcal{S}_j} l_{j,i}(\tilde{v}_i - v_i)}{\sum_{i \in \mathcal{S}_j} l_{j,i}} \tag{4.39}$$

Seja u_k a posição do nó k do esqueleto. O deslocamento do nó k é dado por atualizar a posição u_k, onde é realizado a diferença entre a posição u_k pelo deslocamento médio das bordas d presentes da região Π_k. O cálculo é realizado como sendo $u_k = u_k - d$. O deslocamento médio d é calculado conforme os seguintes casos (AU et al., 2008):

1. Caso o nó k pertencer a uma ramificação (a região Π_k possui um formato semelhante a um cilíndro), o deslocamento é definido como o deslocamento médio entre as duas bordas presentes em Π_k, sendo portanto $d = (d1 + d2)/2$.

2. Caso o nó k estiver na extremidade do esqueleto (onde Π_k possui apenas uma única borda), d é definido como apenas o deslocamento médio desta única borda.

3. Caso o nó estiver em uma articulação (onde Π_k possui mais que duas bordas, d é definido como o somatório ponderado do deslocamento médio de todas as bordas presentes em Π_k, cujo o peso é o comprimento do laço de cada fronteira.

Além da cirurgia de conectividade e refinamento dos nós do esqueleto pode ser necessário alguns tratamentos adicionais no esqueleto. Após o processo de refinamento é possível que a estrutura em algumas das articulações no esqueleto seja mais complexa que a mesma articulação na malha original. Isto pode ser notado pela Figura 37 (à esquerda) ao observar o ciclo presente no esqueleto em uma das articulações, o qual não existe na malha original. Portanto, é necessário um refinamento adicional para nós do esqueleto pertencentes a articulações. Este refinamento tem o objetivo de simplificar a complexidade destas estruturas (Figura 37 à direita), onde um nó em uma articulação e um outro adjacente a ele são fundidos se o nó resultante desta fusão possuir melhor centralização (AU et al., 2008).

O grau de centralidade σ_k para um nó de articulação k do esqueleto é calculado como sendo o desvio padrão das distâncias entre o nó k e os vértices da região Π_k. Desta forma, quanto menor a variação entre a distâncias do nó k e os vértices da região Π_k, mais próximo do centro de Π_k o nó k está localizado. Sendo σ'_k o grau de centralidade do nó resultante da fusão entre o nó k e um outro nó adjacente a ele. A fusão entre eles é realizada se $\sigma'_k < 0.9\sigma_k$, onde o nó adjacente a ser escolhido é aquele que possuir o melhor valor σ'_k dentre os nós vizinhos de k. O processo continua até que para nenhum nó vizinho a k a condição de fusão seja satisfeita (AU et al., 2008).

4.3.1 Análise do processo de esqueletização

O algoritmo proposto por Au et al. (2008) é insensível à rotação e a ruídos na superfície do objeto 3D pelo fato de implicitamente realizar a suavização Laplaciana durante o processo de contração, além de respeitar a geometria e topologia da malha original. Outra vantagem deste algoritmo está relacionada com a possibilidade de associar cada nó do esqueleto ao seu conjunto de vértices da malha original. Isto possui diversas aplicações como segmentação (Figura 38 à esquerda) e para animação do objeto 3D (Figura 38 à direita) (AU et al., 2008).

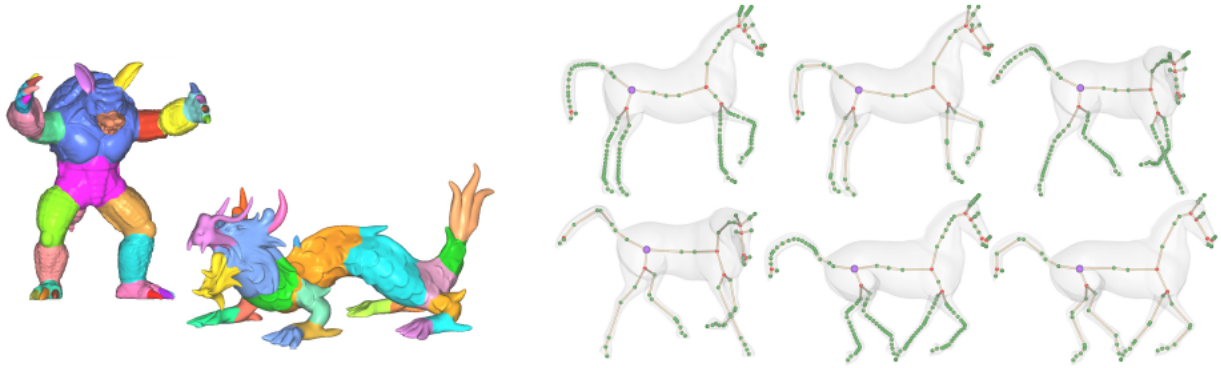

Figura 38 – Exemplo do uso do esqueleto para segmentação e animação
Fonte: Au et al. (2008)

Alguns detalhes relacionados ao processo de contração devem ser considerados. Um destes aspectos para o algoritmo está relacionado ao cálculo do volume da malha contraída, o que é necessário para satisfazer a condição de parada do processo iterativo de contração. O volume de uma malha 3D pode ser calculado de modo simples e prático conforme o algoritmo proposto por Zhang e Chen (2001). O cálculo para o volume de uma malha 3D pode ser encontrado de modo semelhante à Zhang e Chen (2001) na publicação de Desbrun et al. (1999), onde é calculado como sendo o somatório do volume de todos os tetraedros formados por um triângulo presente na superfície da malha e um ponto no espaço (por exemplo, a origem).

Outro detalhe é o calculo do sistema linear expresso pela Equação 4.32. Ao observar a Equação 4.30, nota-se que a maioria dos elementos da matriz $\mathbf{L}$ são nulos, pelo fato de geralmente cada vértice de uma malha triangular possuir poucos vizinhos com relação à quantidade de vértices existentes, o que conclui que a matriz $\mathbf{L}$ é uma **matriz esparsa**. Além disto, $\mathbf{L}$ é uma matriz simétrica e a matriz $\mathbf{M} = \mathbf{L}^T\mathbf{L}$ é uma matriz esparsa e definida positiva. Isto permite que o sistema linear na Equação 4.32 possa ser calculado utilizando a **decomposição de Cholesky** (SORKINE, 2005). Outras abordagens buscam solucionar este sistema através do **método do gradiente biconjugado precondicionado**, o qual pode ser utilizado para resolução de sistemas lineares esparsos (DESBRUN et al., 1999).

Ambos os métodos numéricos podem ser encontrados no livro *Numerical Recipes* [2] . Pelo fato do sistema linear utilizar matrizes esparsas, pode-se armazenar computacionalmente as matrizes no formato **CSR** ou **CCS** (Seção 3.10).

Uma vantagem em relação ao processo de contração da malha é a escolha dos pesos utilizados ao calcular o operador laplaciano **L**. O peso uniforme (Seção 3.9) produz uma malha suavizada que não respeita a geometria da malha original. Tal detalhe é corrigido ao utilizar o peso baseado no cálculo da cotangente (Equação 3.43 na Seção 3.9), como pode ser observado pela Figura 39, na qual nota-se o resultado da malha original (à esquerda) sendo contraída utilizando o peso uniforme (ao centro) e o cálculo da cotangente (à direita). Além disto, o peso utilizando o cálculo da cotangente evita que o vértice v_i se desloque quando este e seus vizinhos adjacentes $N(v_i)$ pertencem a um mesmo plano (DESBRUN et al., 1999). A Figura 40 (à esquerda) mostra a direção do vértice v_i ao utilizar o peso uniforme (direcionando para o centro da vizinhança de v_i) e o peso cotangente, enquanto a Figura 40 (à direita) mostra o deslocamento do vértice v_i ao utilizar o peso uniforme.

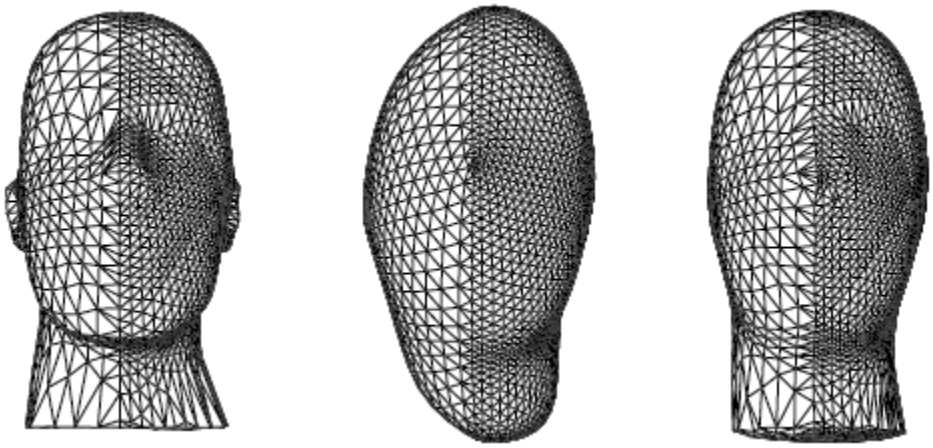

Figura 39 – Comparação entre peso uniforme e peso cotangente para suavização
Fonte: Desbrun et al. (1999) adaptada pelo autor

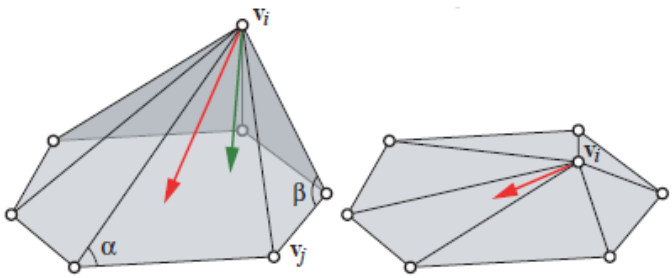

Figura 40 – Deslocamento de um vetor pelo peso uniforme e cotangente
Fonte: Nealen et al. (2006) adaptada pelo autor

[2] Link para Numerical Recipes: http://numerical.recipes/

A grande desvantagem do uso do peso utilizando a cotangente é a possibilidade de gerar valores inválidos como **pesos negativos** ou até mesmo **divisão por zero** (AU et al., 2008). A cotangente de um ângulo α é indefinida quando o ângulo α for igual a 0^o ou a 180^o e pode gerar pesos negativos quando α for maior que 90^o (IEZZI, 1978). A probabilidade destes erros ocorrerem aumentam durante as iterações, pois o processo de contração possui o objetivo de gerar uma malha com volume praticamente igual a zero, o que leva a faces sem nenhuma área (AU et al., 2008). Uma vez que um dos cálculos mais eficientes da cotangente para este algoritmo envolve a norma do produto vetorial (Equação 3.15) e que o produto vetorial está relacionado com a área do triângulo formado entre dois vetores (Equação 3.17), para que a área de uma face seja zero é necessário que os vetores que formam o ângulo α sejam colineares. Logo o ângulo α é igual a 0^o ou a 180^o, tornando o valor da cotangente indefinido (STEINBRUCH; WINTERLE, 1987a). A divisão por zero neste caso ocorre pois a norma do produto vetorial entre os vetores que formam o ângulo α ocorre no denominador da Equação 3.15.

Uma alternativa para o peso cotangente seria a utilização do **peso tangente** (Seção 3.9) no processo de suavização, o qual é bem definido e produz pesos positivos (FLOATER, 2003). Outra alternativa seria aplicar o algoritmo de *flipping* de arestas utilizado em **triangulações de Delaunay** para solucionar pesos cotangente negativos, de tal forma que não modifica a geometria da malha original. A idéia é aplicar o algoritmo de *flipping* em arestas que não formem triangulações de Delaunay. Isto pois triangulações que respeitam os critérios de Delaunay produzem pesos cotangentes positivos. Por esta razão, este algoritmo de correção utiliza como critério para o *flipping* de uma aresta se esta possui peso cotangente negativo (FISHER et al., 2006). A Figura 41 mostra um exemplo deste algorimo para correções da triangulação da malha, onde as arestas em vermelho representam as arestas que foram corrigidas aplicando o algoritmo de *flipping* de arestas.

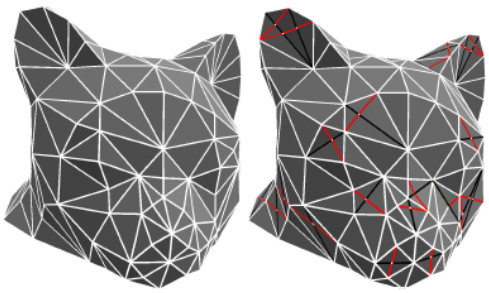

Figura 41 – Tratamento de pesos cotangentes por critérios de Delaunay
Fonte: Fisher et al. (2006)

A comparação entre os pesos apresentada pela Figura 42 utiliza uma malha triangular com 927 vértices e 1850 faces e o processo de suavização expresso pela Equação 3.39, onde a suavização é realizada utilizando 10, 50, 100, 120 e 200 iterações. As linhas representam, de cima para baixo, a suavização da malha utilizando o peso uniforme, o

peso cotangente e o peso tangente, nesta ordem.

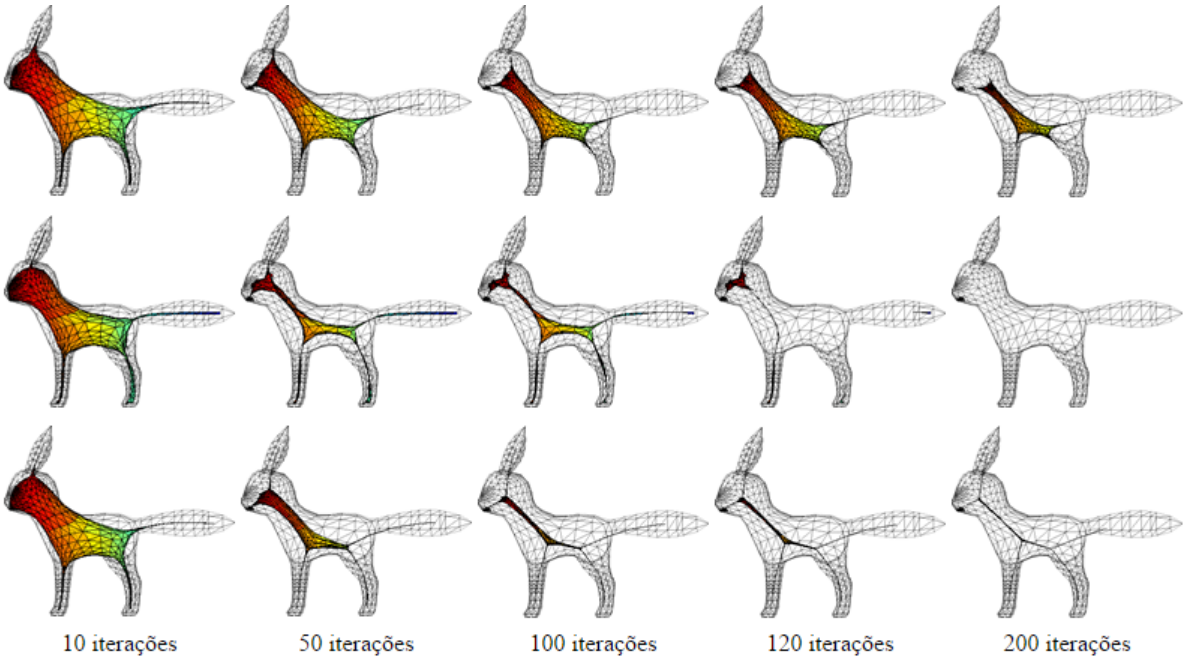

Figura 42 – Suavização de uma malha. De cima para baixo: peso uniforme, cotangente e tangente

Fonte: Elaborada pelo autor

Note na Figura 42 que o peso cotangente obtém uma estrutura mais próxima à geometria da malha original. No entanto, ocorre perda de informações devido à divisões por zero na 120-ésima iteração, uma vez que o MATLAB armazena estes valores como um valor indefinido (*Not-a-number*) (MATHWORKS, 2016d). Dentre os três pesos, o peso tangente apresenta maior estabilidade na Figura 42, aproximando a uma estrutura com volume praticamente nulo. Entretanto, a malha contraída por este peso não é tão fiel à geometria quanto o peso cotangente. Uma vantagem é que a cirurgia de conectividade e refinamento em Au et al. (2008) pode corrigir a centralidade do esqueleto para este peso.

Uma limitação em relação ao algoritmo é que apenas suporta superfícies fechadas para que seja possível construir um operador laplaciano bem definido. Ademais, o algoritmo não resulta em esqueletos com uma boa qualidade quando o objeto 3D de entrada for muito simplificado (AU et al., 2008). Pode-se observar que, para malhas tridimensionais que possuem partes combinadas, o processo de suavização irá contrair cada uma delas de tal maneira que estas poderão se desconectar do próprio objeto original (Figura 43).

O trabalho de Au et al. (2008) tem sido referenciado em diversas publicações, as quais o utilizam como base para seus próprios métodos. A publicação de Tagliasacchi et al. (2012) utiliza o algoritmo de suavização laplaciana para contração da forma tridimensional, otimizando a suavização ao remodelar e obter os pólos de Voronoi da forma como pré-processamento, aplicando o processo de suavização sobre os pólos de Voronoi.

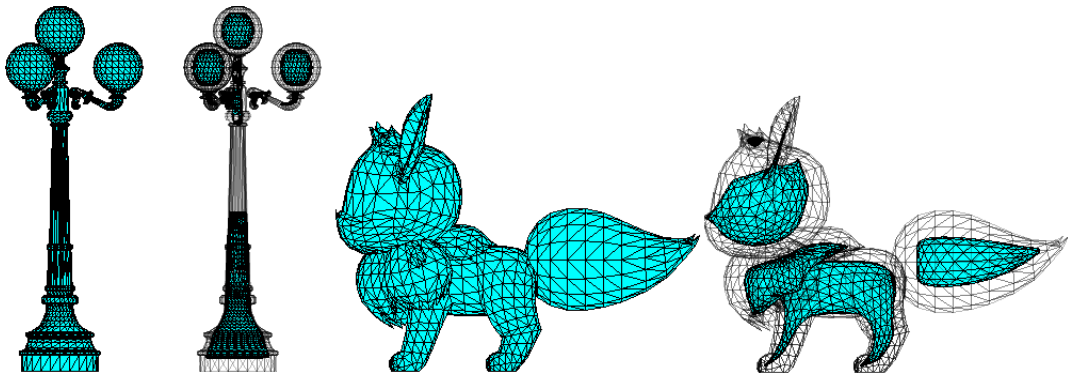

Figura 43 – Exemplo de desconectividade ao contrair uma malha tridimensional
Fonte: Elaborada pelo autor

A publicação de Cao et al. (2010), onde o algoritmo de esqueletização de Au et al. (2008) é adaptado para nuvem de pontos. As etapas encontradas no algoritmo de Cao et al. (2010) são praticamente idênticas à abordagem de Au et al. (2008), inclusive o processo de contração e construção do operador laplaciano. Apenas alguns aspectos divergem entre cada abordagem, cujos detalhes podem ser encontrados em Cao et al. (2010).

O principal detalhe da abordagem de Cao et al. (2010) a ser analisado é o **processo da construção do operador laplaciano**. O operador laplaciano calculado por Au et al. (2008) é calculado utilizando o peso cotangente entre vértices adjacentes a uma aresta na malha. Como o objeto 3D de entrada é representado por uma **nuvem de pontos**, Cao et al. (2010) busca estimar as ligações entre os pontos do objeto 3D através da **triangulação de Delaunay**, para que seja possível calcular o operador laplaciano em Au et al. (2008). Como o operador laplaciano em Au et al. (2008) analisa para cada vértice da malha apenas os vértices adjacentes, apenas as informações dos pontos mais próximos são utilizadas para construir a triangulação de Delaunay em Cao et al. (2010). Portanto, sendo $P = \{p_1, p_2, \ldots, p_i, \ldots, p_n\}$ os pontos de um objeto 3D, o autor segue os seguintes passos para estimar o operador laplaciano **L**:

1. Encontre os k pontos mais próximos $N(p_i)$ através do algoritmo **k-vizinhos mais próximos**, onde $k = 0.012n$ e n é a quantidade de pontos presentes na nuvem de pontos de entrada.

2. Estime o plano tangente π_i através da **análise de componentes principais** dos pontos $N(p_i)$, conforme algoritmo de Hoppe et al. (1992) descrito na Seção 3.7.

3. Projete os pontos $N(p_i)$ no plano π_i, obtendo os pontos $N_{proj}(p_i)$.

4. Calcule a **triangulação de Delaunay T** utilizando os pontos $N_{proj}(p_i)$.

5. Encontre $N_{ring}(p_i)$ como sendo os vizinhos adjacentes a p_i na triangulação **T** e construa as informações da i-ésima linha do operador laplaciano **L**.

Pelo fato da triangulação de Delaunay ser calculada utilizando os pontos projetados no plano π_i e uma vez que estes pontos estão no espaço, reduzir uma dimensão nos dados pode ser útil quando calcular a triangulação de Delaunay. Para isto, basta aplicar transformações de translação (Equação 3.25) e de rotação (Equação 3.26) a fim de deslocar e rotacionar o plano π_i para um dos planos coordenados.

Apesar da proposta de Cao et al. (2010) apresentar uma boa adaptação para a construção do operador laplaciano, é possível ocorrer distorções ao calcular a conectividade entre os pontos. Isto ocorre pois a triangulação de Delaunay é realizada sobre os pontos vizinhos a $p_i \in P$ que foram projetados em um plano. Isto pode ser observado quando π_i está próximo a uma articulação, onde a triangulação sobre os pontos $N_{proj}(p_i)$ podem conectar pontos inclusive de regiões distintas do objeto 3D. A Figura 44 apresenta uma malha triangular construída a partir das faces dos pontos vizinhos de cada ponto π_i geradas pelo algoritmo de Cao et al. (2010).

Como a abordagem de Cao et al. (2010) procura adaptar o algoritmo de Au et al. (2008) para nuvem de pontos, uma possível alternativa seria reconstruir a superfície do objeto 3D a partir de sua nuvem de pontos de entrada e, em seguida, aplicar o algoritmo de Au et al. (2008). Dey, Giesen e Hudson (2001) propõe um algoritmo para reconstrução da superfície de objetos 3D baseado em triangulação de Delaunay e diagramas de Voronoi.

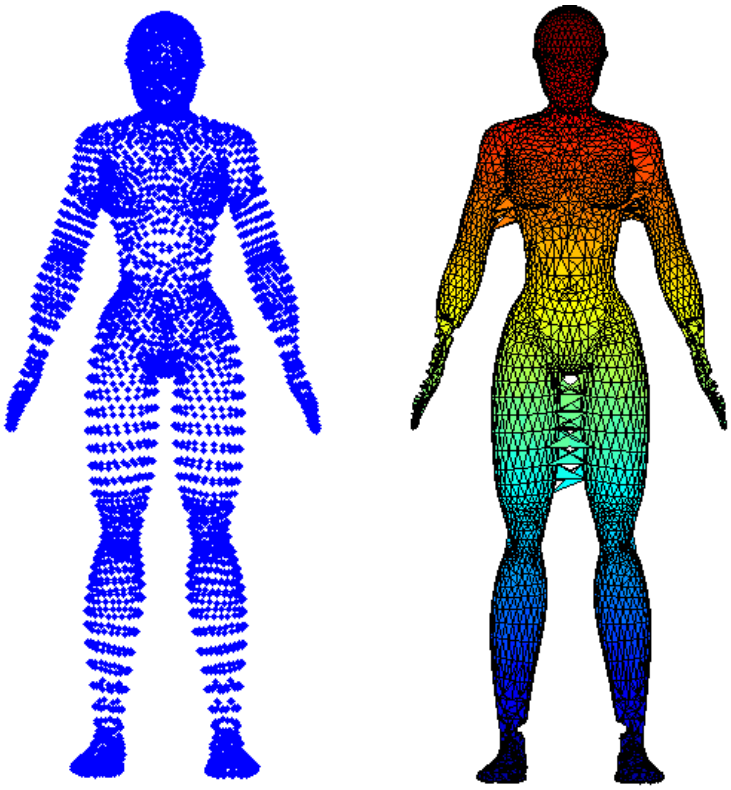

Figura 44 – Distorções ao construir malhas por triangulação de Delaunay
Fonte: Elaborada pelo autor

4.4 Esqueletização de malhas por contração e agrupamento

Assim como o processo de esqueletização proposto por Au et al. (2008), o algoritmo proposto por Jiang et al. (2013) também envolve contração da malha. O algoritmo converte a malha triangular 3D em um grafo, o qual será contraído iterativamente até que neste grafo resulte no esqueleto final. O processo de **contração do grafo** (Figura 45 acima) busca reduzir a quantidade de vértices até que todos os triângulos sejam removidos, resultando no esqueleto final (Figura 45d). Para cada iteração, é feita a contração do grafo seguida de um **agrupamento da superfície** (Figura 45), o qual agrupa os vértices da malha original em regiões definidas pela quantidade de vértices presentes no grafo. Também é proposto um **algoritmo de refinamento do esqueleto** como pós-processamento opcional (Figura 45e). Além do esqueleto, este algoritmo também realiza o **mapeamento** entre os vértices da superfície e cada vértice do esqueleto (JIANG et al., 2013).

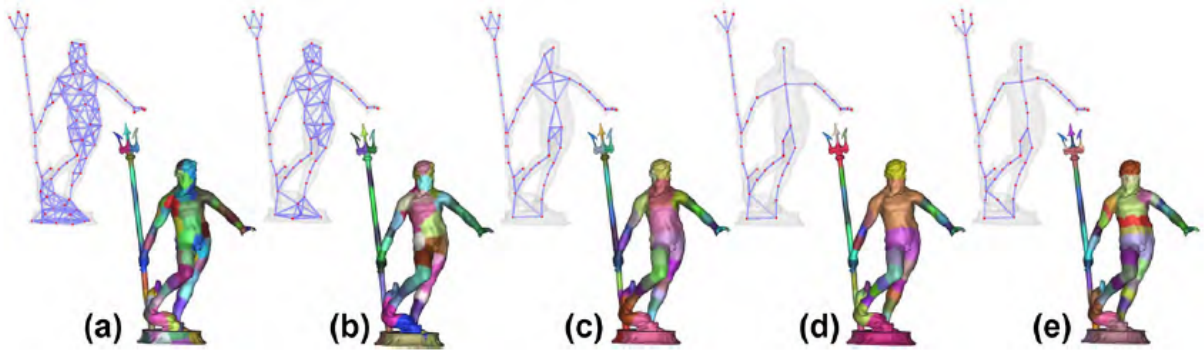

Figura 45 – Exemplo de esqueletização por contração e agrupamento
Fonte: Jiang et al. (2013)

A contração do grafo e o processo de agrupamento da superfície são feitos através do cálculo do **diagrama de Voronoi centroidal aproximado** (*approximated centroidal Voronoi diagram* ou ACVD). Este diagrama pode ser encontrado na publicação de Valette e Chassery (2004) (detalhes na Seção 4.2). No entanto, o diagrama de Voronoi em Valette e Chassery (2004) realiza o agrupamento das faces da malha triangular 3D, enquanto as etapas deste algoritmo realizam o agrupamento dos vértices. A contração do grafo realiza o agrupamento dos vértices pertencentes ao grafo enquanto o agrupamento da superfície realiza o agrupamento dos vértices presentes na superfície do objeto 3D, ou seja, os vértices da malha triangular. Além disto, são adicionadas restrições para que o esqueleto mantenha a topologia do objeto 3D e não fique super contraído (JIANG et al., 2013).

Seja $\mathcal{G}$ o grafo do esqueleto durante o processo de contração, $\mathcal{M}$ a malha triangular 3D de entrada e Ω o vetor que armazena em qual agrupamento pertence cada vértice da superfície. O algoritmo inicializa $\mathcal{G}$ convertendo a malha triangular $\mathcal{M}$ em um grafo, ou seja, constrói o grafo $\mathcal{G}$ como uma cópia da malha $\mathcal{M}$. O vetor Ω é inicializado ao

atribuir para cada vértice $v \in \mathcal{M}$ uma região independente. A contração do grafo e o agrupamento da superfície são executados conforme uma quantidade $\mathbf{N_s}$ de **vértices sementes** presentes no grafo, os quais deverão permanecer após a contração. Para isto uma etapa de **seleção** dos vértices sementes é necessária antes do processo de contração do grafo (JIANG et al., 2013).

Selecionado os vértices sementes, é realizado a contração do grafo. O processo de contração procura identificar qual região do vértice semente cada vértice do grafo irá pertencer. Cada vértice (semente ou não) em $\mathcal{G}$ está associado ao agrupamento da superfície em Ω definido na iteração anterior (Figura 46a). Isto significa que, para o estado inicial, cada vértice do grafo pertence ao seu próprio agrupamento, enquanto que para cada iteração do algoritmo, cada vértice em $\mathcal{G}$ está associado à região da superfície calculada pelo processo de agrupamento da superfície da iteração anterior. No processo de contração, quando um vértice $n_i \in \mathcal{G}$ é associado a um vértice semente n_s, os agrupamentos da superfície associados para n_i e n_s se fundem em um único agrupamento (Figura 46b) (JIANG et al., 2013).

Após esta etapa, o processo de agrupamento da superfície desloca alguns vértices para outras regiões da superfície de modo que esteja distribuída da melhor maneira possível. Este processo força uma **atualização** das **posições** e da **conectividade** entre os vértices do grafo $\mathcal{G}$, uma vez que o agrupamento da superfície controla a posição e a conexão entre os vértices de $\mathcal{G}$. O algoritmo busca estabelecer uma **consistência** entre os dois processos em cada iteração, pois a posição de cada vértice em $\mathcal{G}$ é definida como o centro de massa de cada região da superfície associada a ele, e a adjacência entre os vértices do grafo é controlada pela vizinhança entre as regiões da superfície em Ω. Para isto é realizada uma atualização dos vértices do grafo a fim de prover uma melhor distribuição dos vértices do esqueleto (Figura 46c). O Algoritmo 2 resume como o processo de esqueletização é realizado. Neste algoritmo, a quantidade de vértices sementes é multiplicada por um fator r definido empiricamente como igual a 0.7. Isto significa que para cada iteração a quantidade de vértices sementes é reduzida em 30% (JIANG et al., 2013).

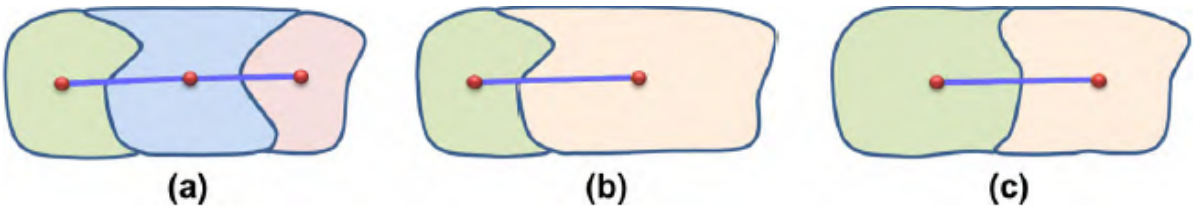

Figura 46 – Etapas de contração do grafo e agrupamento da superfície
Fonte: Jiang et al. (2013)

Antes de realizar a contração do grafo é necessário **selecionar** os vértices sementes. Para isto, é preciso adicionar algumas **restrições** a fim de manter as características topológicas do esqueleto final. Dentre eles estão a **contração excessiva** do grafo e o

afinamento dos vértices na extremidade do grafo (Figura 47). Para evitar tais distorções, alguns **vértices essenciais** devem ser selecionados como vértices sementes para que não desapareçam no processo de contração do grafo e, desta maneira, permanecerem até o esqueleto final. Portanto, tanto os vértices que definem a topologia do esqueleto quanto aqueles que se localizam na extremidade possuem prioridade na seleção de vértices sementes, os quais devem ser no máximo $\mathbf{N_s}$ vértices (JIANG et al., 2013).

Figura 47 – Etapas de contração e agrupamento sem restrições
Fonte: Jiang et al. (2013)

Algoritmo 2: esqueletização por contração e agrupamento

Entrada:
$\mathcal{M} \leftarrow$ malha triangular 3D
Saída:
$\mathcal{K} \leftarrow$ esqueleto em formato de grafo
$\Omega \leftarrow$ o vetor de agrupamentos da superfície, onde $\Omega = \{C_i | i = 1, \ldots, N\}$ e N a quantidade de vértices da malha

1 **início**
2 inicializa o grafo $\mathcal{G}$ como uma cópia da malha $\mathcal{M}$;
3 inicializa o vetor de agrupamento $\Omega = \{C_i | i = 1, \ldots, N\}$, onde cada vértice $v_i \in \mathcal{M}$ está em uma região C_i distinta;
4 $\mathbf{N_s} \leftarrow 200$;
5 **repita**
6 $\mathbf{N_s} \leftarrow r \times \mathbf{N_s}$;
7 seleciona os $\mathbf{N_s}$ vértices sementes em $\mathcal{G}$;
8 contração do grafo $\mathcal{G}$ utilizando $\mathbf{N_s}$ vértices sementes;
9 agrupamento da superfície em Ω;
10 atualiza contração $\mathcal{G}$ e agrupamento Ω;
11 **até** *nenhuma contração possível em* $\mathcal{G}$;
12 $\mathcal{K} \leftarrow \mathcal{G}$;
13 opcionalmente, realiza o refinamento do esqueleto $\mathcal{K}$;
14 **retorna** $(\mathcal{K}, \Omega)$;
15 **fim**

Como o grafo é composto por triângulos, alguns destes são **triângulos essenciais**,

pois contém informações sobre a topologia do esqueleto. Outros triângulos são considerados **triângulos não essenciais**, pois não necessários para definir a topologia do esqueleto. Esta classificação pode ser identificada através dos agrupamentos da superfície para cada vértice presente no triângulo, conforme a Figura 48. Seja $t = (u, v, w)$ um triângulo, onde $u, v, w \in \mathcal{G}$, e C_u, C_v e C_w as regiões da superfície em Ω. Este triângulo não é essencial caso não exista algum vértice da superfície que torne as regiões C_u, C_v e C_w adjacentes entre si, este triângulo é essencial (JIANG et al., 2013).

Como os vértices de cada triângulo essencial no grafo define a topologia do esqueleto, estes necessitam estar presentes no esqueleto final para evitar super contração. Por isto todos os vértices de triângulos essenciais são selecionados como sementes. Além disto, a fim de evitar afinamento nas extremidades, os vértices no grafo que possuem **grau menor ou igual a dois** são selecionados também como sementes. Para os triângulos não essenciais é selecionado apenas um de seus vértices como semente. Para isto é construída uma **fila de prioridades** para cada vértice do grafo ainda não selecionado de tal modo que os vértices que possuírem maior grau tenham maior prioridade da fila. Em seguida, o algoritmo seleciona como semente o vértice presente no início da fila e remove desta fila todos os demais que são adjacentes a este vértice, até que a fila esteja vazia. Caso a quantidade de vértices selecionados for menor do que $\mathbf{N_s}$, selecione aleatoriamente vértices sementes dentre aqueles que não foram selecionados até obtem $\mathbf{N_s}$ vértices sementes. Este processo pode ser resumido pelo Algoritmo 3 abaixo. (JIANG et al., 2013)

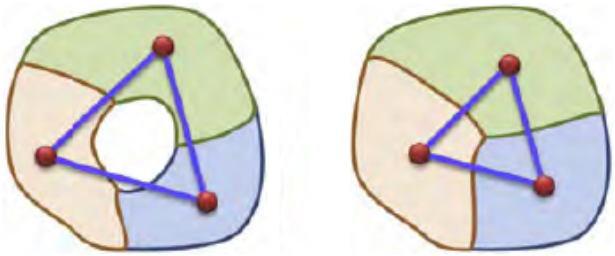

Figura 48 – Exemplo de triângulos essenciais e não essenciais no grafo, nesta ordem
Fonte: Jiang et al. (2013) adaptada pelo autor

Após a seleção dos vértices sementes é executado o processo de contração do grafo. O cálculo do diagrama de Voronoi centroidal para a contração do grafo e para o agrupamento da superfície é semelhante ao algoritmo de agrupamento **k-means**. Seja C_j uma região de Voronoi, n_i um vértice do grafo pertencente à região C_j, c_j o ponto centróide da região C_j e ω_i um peso calculado para cada vértice n_i. O processo de contração do grafo $\mathcal{G}$ procura calcular o diagrama de Voronoi centroidal que **minimize a energia** $\mathcal{F}_\mathcal{G}$ expressa pela seguinte equação (JIANG et al., 2013):

$$\mathcal{F}_\mathcal{G} = \sum_{j=1}^{\mathbf{N_s}} \left(\sum_{n_i \in C_j} \omega_i \|n_i - c_j\|^2 \right) \tag{4.40}$$

Algoritmo 3: Seleção de vértices sementes

Entrada:

$\mathcal{G} \leftarrow$ o grafo do esqueleto 3D;

$\mathbf{N_s} \leftarrow$ a quantidade de vértices sementes;

Saída:

$\mathcal{S} \leftarrow$ um conjunto de vértices do grafo selecionados como sementes;

```
1  início
2      inicializa conjunto de vértices S ← ∅;
3      inicializa fila de prioridades Q ← ∅;
4      para cada vértice v ∈ G faça
5          se grau(G, v) ≤ 2 então
6              S ← S ∪ {v};
7          fim
8          senão
9              para cada triângulo t = (u, v, w) no grafo G contendo v faça
10                 se triângulo t é essencial então
11                     S ← S ∪ {u, v, w};
12                 fim
13                 senão
14                     enfileire u, v, w em Q conforme maior grau no grafo;
15                 fim
16             fim
17         fim
18     fim
19     enquanto Q ≠ ∅ faça
20         v ← desenfileire(Q);
21         S ← S ∪ {v};
22         para cada aresta e = (u, v) ∈ G faça
23             remove u do fila Q;
24         fim
25     fim
26     enquanto |S| < N_s faça
27         escolha v aleatoriamente em G;
28         S ← S ∪ {v};
29     fim
30     retorna S
31 fim
```

O peso ω_i na Equação 4.40 é definido como sendo o inverso do somatório da

área dos triângulos formados por n_i e sua vizinhança adjacente $N_1(i)$. Matematicamente, $\omega_i = (\sum_{j \in N_1(i)} A_{ij})^{-1}$. Este peso é definido como 10^6 quando não existir triângulos vizinhos ao vértice n_i (JIANG et al., 2013). O cálculo do diagrama de Voronoi centroidal é realizado através do **algoritmo de Lloyd** para diagramas de Voronoi, de modo semelhante ao cálculo em Valette e Chassery (2004) (JIANG et al., 2013).

Este algoritmo é iniciado a partir de um conjunto de vértices sementes e suas respectivas regiões de Voronoi iniciais (JIANG et al., 2013). Pode-se observar que o algoritmo em Valette e Chassery (2004) é iniciado com um conjunto de faces da malha triangular definidas como sementes, onde para cada uma delas é criada uma região de Voronoi. As **arestas de borda** em Valette e Chassery (2004) são definidas inicialmente pelas arestas de cada face triangular definida como semente. De modo similar, para cada vértice semente em Jiang et al. (2013) é atribuída uma região de Voronoi do grafo e as arestas de borda iniciais são as arestas que contém cada vértice semente. Em seguida, cada região de Voronoi é expandida iterativamente até que não seja mais possível realizar a **expansão** de nenhuma região. Este processo é guiado pela Equação 4.40 de tal forma que este processo busque a cada iteração minimizar esta equação (JIANG et al., 2013).

O **processo de expansão** é realizado para cada aresta de borda $e_{ij} = (n_i, n_j)$, isto é, para toda aresta $e_{ij} = (n_i, n_j)$ do grafo onde os vértices n_i e n_j pertencem a regiões de Voronoi diferentes. Seja C_1 e C_2 regiões de modo que $n_i \in C_1$, $n_j \in C_2$ e $C_1 \neq C_2$. O processo avalia a expansão destas regiões em três possíveis cenários: mover n_j para C_1 (expandindo C_1), mover n_1 para C_2 (expandindo C_2), ou manter a configuração original. Esta avaliação consistem em executar o cenário que resulta em um menor valor para $\mathcal{F}_G$. Ou seja, a Equação 4.40 é calculada para o cenário de expansão de C_1 ($n_j \in C_1$ e $n_j \notin C_2$), o cenário de expansão de C_2 ($n_i \in C_2$ e $n_i \notin C_1$) e para o cenário original ($n_i \in C_1$ e $n_j \in C_2$). Os resultados são comparados e o algoritmo opta por executar o cenário que apresentou um menor valor para $\mathcal{F}_G$. Por exemplo, caso $\mathcal{F}_G$ obteve um valor mínimo para o cenário de expansão de C_1, então o algoritmo irá mover o vértice n_i de C_2 para C_1. Este processo encerra quando não for necessário realizar nenhuma expansão (JIANG et al., 2013).

No processo de contração do grafo, em cada iteração do processo de expansão definido pelo algoritmo de Lloyd, quando ocorre uma expansão e um vértice é movido de uma região de Voronoi do grafo para outra, as regiões do agrupamento da superfície em Ω também são atualizados. Ou seja, considere a aresta de borda $e_{ij} = (n_i, n_j)$ e as regiões de Voronoi do grafo C_1 e C_2, onde $n_i \in C_1$ e $n_j \in C_2$. Suponha que em algum instante o processo de contração do grafo expandiu C_2, movendo n_i de C_1 para C_2. A atualização do agrupamento da superfície é realizada por unir a região de Voronoi da superfície (em Ω) associada a n_i com a região de Voronoi da superfície associada ao vértice semente da região C_2 (Figura 46b). Isto guia o algoritmo a realizar o agrupamento da superfície para

distribuir estes vértices presentes na superfície e realocar os vértices do grafo para o centro de massa das regiões da superfície (Figura 46c) (JIANG et al., 2013).

Após o processo de contração do grafo, é realizado o **agrupamento da superfície**. O agrupamento da superfície também é um diagrama de Voronoi centroidal, portanto, pode-se utilizar procedimento semelhante ao que foi aplicado no processo de contração do grafo. No entanto, este agrupamento é realizado nos vértices da superfície da malha. Como a contração do grafo realiza a junção das regiões da superfície, então existem neste ponto $\mathbf{N_s}$ regiões da superfície em Ω. Um aspecto crucial deste cálculo é que o agrupamento da superfície definirá posteriormente a posição dos vértices do grafo como centro de massa de suas regiões. Por esta razão são adicionadas restrições no processo de expansão das regiões da superfície em relação à distância da região da superfície de seu centro de massa. Seja $d(c_j, C_j)$ a distância euclidiana da região da superfície C_j e do seu centro de massa c_j. Então o processo de agrupamento da superfície é realizado com o objetivo de minimizar a seguinte equação (JIANG et al., 2013):

$$\mathcal{F}_\Omega = \sum_{j=1}^{\mathbf{N_s}} \Big(\sum_{n_i \in C_j} \omega_i \|n_i - c_j\|^2 \Big) + 1/d^2(c_j, C_j) \tag{4.41}$$

O agrupamento da superfície também pode ser calculado utilizando o algoritmo de Lloyd. No entanto, o processo de expansão, neste caso, também busca maximizar a distância da superfície ao centro de massa, o que é expresso pelo segundo termo do somatório na Equação 4.41. Para calcular este segundo termo, o algoritmo busca o vértice da superfície mais próximo por meio de uma **kd-tree**. Sendo v_{kd} este vértice mais próximo, o segundo termo é calculado como sendo a menor distância encontrada entre o centro de massa e os vértices adjacentes a v_{kd} (JIANG et al., 2013).

Após o agrupamento da superfície, os vértices do grafo são revisados de forma que estes estejam no centro de massa de suas respectivas regiões da superfície. Com isto, é encerrada uma iteração do algoritmo de esqueletização. O procedimento se repete para as demais iterações, entretanto com uma redução de 30% da quantidade de vértices sementes, o que contrai ainda mais o grafo do esqueleto. A esqueletização encerra quando não existir mais nenhum triângulo presente no grafo para ser contraído (JIANG et al., 2013).

Opcionalmente, após as iterações, é possível realizar o **refinamento** do esqueleto final. Nesta etapa podem ser acrescentados novos vértices entre dois vértices adjacentes. Portanto, dado uma aresta $e_{ij} = (n_i, n_j)$ do esqueleto final, o refinamento calcula o novo vértice a ser acrescentado por encontrar o centro de massa da união das regiões da superfície associadas a n_i e n_j (JIANG et al., 2013).

4.4.1 Análise do processo de esqueletização

O algoritmo de Jiang et al. (2013) é capaz de extrair esqueletos que preservam propriedades desejadas de um esqueleto 3D tais como centralidade, conectividade, robustez e a preservação das características topológicas. Além disto, devido à atualização dos vértices do grafo realizado graça ao agrupamento da superfície, o algoritmo é robusto a ruídos e distorções na superfície (JIANG et al., 2013).

Uma grande vantagem desta abordagem é a possibilidade de ser utilizado quando o objeto 3D de entrada for representado por um nuvem de pontos ao invés de uma malha triangular. O algoritmo depende de um grafo que represente o objeto 3D, portanto, basta construir o grafo inicial a partir das informações dos k-vizinhos mais próximos dos pontos. Com isto é possível extrair esqueletos razoáveis a partir de nuvens de pontos, conforme Figura 50 (JIANG et al., 2013). Isto é uma vantagem encontrada em relação ao algoritmo de Tagliasacchi, Zhang e Cohen-Or (2009), uma vez que o algoritmo de Jiang et al. (2013) é mais simples de ser implementado, não necessitando encontrar os vetores normais dos pontos ou o uso de métodos numéricos como a decomposição de valores singulares.

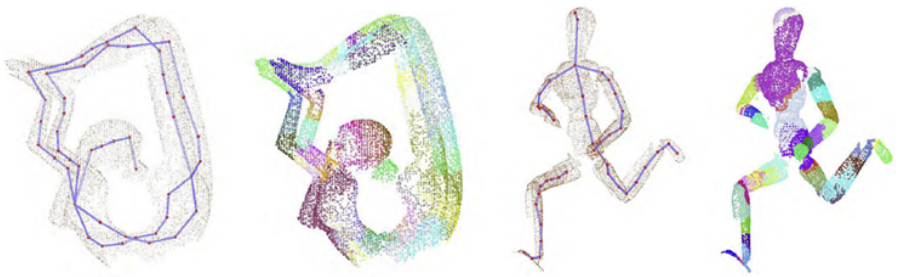

Figura 49 – Esqueletização do algoritmo adaptado para nuvens de pontos
Fonte: Jiang et al. (2013) adaptada pelo autor

Pode-se observar que este algoritmo possui maior centralidade do que o algoritmo de Au et al. (2008). Ao observar o objeto 3D de uma mão presente na Figura 50, pode-se notar que o algoritmo de Au et al. (2008) produz alguns ramos indesejados. Entretanto, em alguns cenários, esta desvantagem pode auxiliar em uma melhor representação do esqueleto do objeto 3D, o que é observado no objeto 3D de um coelho na Figura 50. Além disto, Jiang et al. (2013) não necessita que a superfície seja fechada (JIANG et al., 2013).

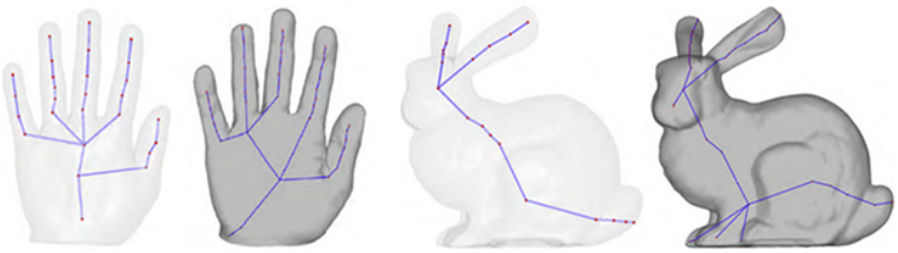

Figura 50 – Esqueleto 3D obtido pelos algoritmos de
Fonte: Jiang et al. (2013) adaptada pelo autor

5 Considerações Finais

5.1 Conclusão

Neste trabalho foi realizado uma análise das publicações mais refenciadas na literatura para extração de esqueletos de objetos 3D representados por nuvem de pontos ou malhas triangulares. Foi analisado os algoritmos de esqueletização publicados por Tagliasacchi, Zhang e Cohen-Or (2009), Au et al. (2008) e Jiang et al. (2013), além de apresentar os algoritmos propostos por Tagliasacchi et al. (2012) e Cao et al. (2010) para esqueletização e o algoritmo de Valette e Chassery (2004) para simplificação da malha triangular de um objeto 3D.

Foi analisado o algoritmo de Tagliasacchi, Zhang e Cohen-Or (2009) busca extrair o esqueleto de um objeto 3D representado por uma nuvem de pontos por calcular o ponto de eixo de simetria rotacional para cada região de pontos pertencentes a planos de corte feitos ao longo do objeto 3D. Este algoritmo requer que o objeto de entrada seja geralmente cilíndrico e consegue ser robusto mesmo com perda de informações graças aos vetores normais de cada ponto do objeto 3D. Como este algoritmo depende dos vetores normais dos pontos de entrada, foi analisada a publicação de Hoppe et al. (1992) e seu algoritmo para extrair vetores normais a partir de um conjunto de pontos, o qual é utilizado por Tagliasacchi, Zhang e Cohen-Or (2009). Em seguida, foi apresentada uma comparação entre o algoritmo de Hoppe et al. (1992) com os algoritmos de Xie et al. (2003) e König e Gumhold (2009) devido a falhas presentes no algoritmo de Hoppe et al. (1992) para correção da orientação dos vetores normais.

Foi analisado o algoritmo de Au et al. (2008), o qual obtem o esqueleto de uma malha triangular através de sucessivas contrações utilizando o algoritmo de suavização laplaciana. Também é aplicado neste algoritmo um processo de cirurgia de conectividade para remoção das faces triangulares e o refinamento do esqueleto final. Trata-se de um algoritmo robusto e insensível a ruídos. Além disto, a quantidade de vértices e faces do objeto 3D não influencia a qualidade do esqueleto final, desde que esta quantidade não seja muito pequena. Foi apresentado quais métodos poderiam ser utilizados para o cálculo do sistema linear e algoritmos para o cálculo do volume da malha como o publicado por Zhang e Chen (2001). Um aspecto crucial do algoritmo de Au et al. (2008) é a escolha do peso utilizado no cálculo do operador laplaciano. Dentre eles, o peso cotangente consegue produzir uma malha contraída com maior fidelidade à geometria e topologia da malha original, embora ocorra falhas como divisões por zero e pesos negativos no decorrer das sucessivas contrações. Como alternativa, foi apresentado o peso tangente que, embora não produz uma malha contraída tão fiel à malha original, o resultado deste

peso é sempre positivo e raramente ocorrerá falhas presentes no peso cotangente, além de representar bem as características geométricas e topologicas da malha original. Foi apresentado também outros artigos que têm como referência a abordagem de Au et al. (2008), como a publicação de Tagliasacchi et al. (2012) e a publicação de Cao et al. (2010).

Foi analisado também o algoritmo de Jiang et al. (2013). Apesar de não ser um dos algoritmos mais referenciados, este é capaz de resolver desvantagens presentes nas abordagens de Tagliasacchi, Zhang e Cohen-Or (2009) e Au et al. (2008). Este algoritmo converte a malha triangular em um grafo, o qual é contraído até formar o esqueleto final. Tal processo é acompanhado de um agrupamento dos vértices da superfície, o qual desloca os vértices do grafo para o centro de massa de cada agrupamento, tornando-o o esqueleto centralizado e fornecendo um mapeamento entre os vértices do esqueleto e os vértices da superfície associados a ele. Esta abordagem também pode ser realizada em nuvem de pontos caso esta possa se converter em um grafo.

Analisando as três abordagens, pode-se concluir que métodos envolvendo contração apresentam melhores resultados. Dentre eles, a abordagem de Jiang et al. (2013) se destaca por ser de simples implementação e por não necessitar de tratamentos adicionais como o peso cotangente utilizado pelo algoritmo de Au et al. (2008). Entretanto, é necessário avaliar a aplicação que utilizará esqueletos de objetos 3D antes de selecionar o algoritmo apropriado. O algoritmo de Au et al. (2008), por exemplo, pode gerar ramos adicionais comparados com o algoritmo de Jiang et al. (2013), o que pode ser útil ou prejudicial para algumas aplicações.

5.2 Difuldades encontradas

Durante o desenvolvimento deste trabalho algumas dificuldades foram encontradas. O primeiro desafio foi a implementação do algoritmo de Tagliasacchi, Zhang e Cohen-Or (2009) ao procurar extrair os vetores normais dos pontos do objeto 3D, uma vez que os algoritmos citados não conseguem corrigir as orientações destes vetores com bastante eficácia.

Outra dificuldade foi ao implementar o algoritmo de Cao et al. (2010), o qual estava causando instabilidades no processo de contração. Uma das avaliações foi analisar a malha que este algoritmo estava buscando construir através de triangulações de Delaunay. Para isto, foi desenvolvido o processo de construção do operador laplaciano e armazenando as faces vizinhas para cada ponto no objeto 3D a fim de construir a tabela de faces da nova malha. O resultado foram as distorções já abordadas neste trabalho.

A maior dificuldade foi a implementação do algoritmo de Au et al. (2008). Para este algoritmo foi realizado testes na contração da malha utilizando a suavização laplaciana extraída pelo método de integração de Euler antes de desenvolver a contração por

resolução do sistema linear. Nesta etapa foi observada as distorções geradas pelo peso cotangente conforme abordado neste trabalho. Ocorreu uma grande dificuldade em encontrar publicações que explicassem como este problema pode ser solucionado, o que levou a alguns meses de pesquisa.

Referências

ALTMAN, Y. M. *Accelerating MATLAB Performance: 1001 Tips to Speed Up MATLAB Programs*. 1st. ed. [S.l.]: Chapman & Hall/CRC, 2014. ISBN 1482211297, 9781482211290. Citado na página 19.

ANDRADE, L. N. de. *Rotações no espaço tridimensional*. 2000. (Data de acesso: 22-11-2016). Disponível em: <http://www.lcad.icmc.usp.br/~rosane/CG/TransfGeomAndersonIcaro.pdf>. Citado 2 vezes nas páginas 29 e 30.

ARUOBA, S. B.; FERNáNDEZ-VILLAVERDE, J. *A Comparison of Programming Languages in Economics*. [S.l.], 2014. (Working Paper Series, 20263). Disponível em: <http://www.nber.org/papers/w20263>. Citado 2 vezes nas páginas 18 e 19.

ASHBROOK, A. P. et al. Robust recogniton of scaled shapes using pairwise geometric histograms. In: *BMVC*. [s.n.], 1995. p. XX–YY. Disponível em: <http://www.bmva.org/bmvc/1995/bmvc-95-049.pdf>. Citado na página 15.

AU, O. K.-C. et al. Skeleton extraction by mesh contraction. *ACM Trans. Graph.*, ACM, New York, NY, USA, v. 27, n. 3, p. 44:1–44:10, ago. 2008. ISSN 0730-0301. Disponível em: <http://doi.acm.org/10.1145/1360612.1360643>. Citado 20 vezes nas páginas 15, 42, 43, 55, 62, 63, 64, 65, 66, 67, 68, 69, 71, 72, 73, 74, 75, 82, 83 e 84.

AURENHAMMER, F. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 23, n. 3, p. 345–405, set. 1991. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/116873.116880>. Citado na página 32.

AUSTIN, D. *Voronoi Diagrams and a Day at the Beach*. 2006. (Data de acesso: 22-02-2017). Disponível em: <http://www.ams.org/samplings/feature-column/fcarc-voronoi>. Citado na página 32.

AZEVEDO, E.; CONCI, A. *Computação gráfica: teoria e prática*. 1. ed. [S.l.]: Elsevier, 2003. ISBN 9788535223293. Citado 6 vezes nas páginas 34, 35, 36, 37, 53 e 54.

BARRETT, R. et al. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. 2. ed. Philadelphia, PA: SIAM, 1994. ISBN 9780898713282. Citado 2 vezes nas páginas 40 e 41.

BOGOMOLNY, A. *Translation Transform from Interactive Mathematics Miscellany and Puzzles*. 1996. (Data de acesso: 22-11-2016). Disponível em: <http://www.cut-the-knot.org/Curriculum/Geometry/Translation.shtml>. Citado na página 29.

BORTOLI Àlvaro Luiz de et al. *Introdução ao Cálculo Numérico*. 2. ed. [s.n.], 2003. (Data de acesso: 15-07-2015). Disponível em: <https://chasqueweb.ufrgs.br/~carolina.manica/cap6.pdf>. Citado na página 30.

BOTSCH, M. et al. Geometric modeling based on triangle meshes. In: *ACM SIGGRAPH 2006 Courses*. New York, NY, USA: ACM, 2006. (SIGGRAPH '06). ISBN 1-59593-364-6. Disponível em: <http://doi.acm.org/10.1145/1185657.1185839>. Citado na página 20.

BOURKE, P. D. *Object Files (.obj)*. 2016. (Data de acesso: 06-12-2016). Disponível em: <http://paulbourke.net/dataformats/obj>. Citado 4 vezes nas páginas 19, 93, 94 e 95.

CANTERAKIS, N. 3D zernike moments and zernike affine invariants for 3D image analysis and recognition. In: *SCIA*. [S.l.: s.n.], 1999. p. Pattern Recognition I. Citado na página 15.

CAO, J. et al. Point cloud skeletons via laplacian based contraction. In: *Proceedings of the 2010 Shape Modeling International Conference*. Washington, DC, USA: IEEE Computer Society, 2010. (SMI '10), p. 187–197. ISBN 978-0-7695-4072-6. Disponível em: <http://dx.doi.org/10.1109/SMI.2010.25>. Citado 7 vezes nas páginas 42, 43, 55, 73, 74, 83 e 84.

DESBRUN, M. et al. Implicit fairing of irregular meshes using diffusion and curvature flow. In: *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999. (SIGGRAPH '99), p. 317–324. ISBN 0-201-48560-5. Disponível em: <http://dx.doi.org/10.1145/311535.311576>. Citado 5 vezes nas páginas 37, 38, 63, 69 e 70.

DEY, T. K.; GIESEN, J.; HUDSON, J. Delaunay based shape reconstruction from large data. In: *Proceedings of the IEEE 2001 Symposium on Parallel and Large-data Visualization and Graphics*. Piscataway, NJ, USA: IEEE Press, 2001. (PVG '01), p. 19–27. ISBN 0-7803-7223-9. Disponível em: <http://dl.acm.org/citation.cfm?id=502125.502129>. Citado na página 74.

ENGENHARIA, E. *Digitalização 3D e Suas Aplicações*. 2012. (Data de acesso: 15-11-2016). Disponível em: <http://www.esx-engenharia.com.br/digitalizacao-3d-e-suas-aplicacoes>. Citado na página 13.

ERSOY, I. *How to use BoundingBox?* 2010. (Data de acesso: 11-01-2017). Disponível em: <http://www.c-sharpcorner.com/uploadfile/iersoy/how-to-use-boundingbox/>. Citado na página 22.

FISHER, M. et al. An algorithm for the construction of intrinsic delaunay triangulations with applications to digital geometry processing. In: *ACM SIGGRAPH 2006 Courses*. New York, NY, USA: ACM, 2006. (SIGGRAPH '06), p. 69–74. ISBN 1-59593-364-6. Disponível em: <http://doi.acm.org/10.1145/1185657.1185668>. Citado na página 71.

FLOATER, M. S. Mean value coordinates. *Comput. Aided Geom. Des.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 20, n. 1, p. 19–27, mar. 2003. ISSN 0167-8396. Disponível em: <http://dx.doi.org/10.1016/S0167-8396(02)00002-5>. Citado na página 71.

FLORES, J. M. *Simple 2D and 3D Shape Descriptions*. 2012. (Data de acesso: 16-11-2016). Disponível em: <http://www.esx-engenharia.com.br/digitalizacao-3d-e-suas-aplicacoes>. Citado na página 14.

FRANçA, R. M. de. *Conheça mais sobre a nuvem de pontos*. 2013. (Data de acesso: 20-07-2015). Disponível em: <http://mundogeo.com/blog/2013/07/15/conheca-mais-sobre-a-nuvem-de-pontos/>. Citado na página 20.

FUNKHOUSER, T. *Solid Modeling*. 2002. (Data de acesso: 15-06-2015). Disponível em: <https://www.cs.princeton.edu/courses/archive/fall01/cs426/lectures/17.pdf>. Citado 2 vezes nas páginas 21 e 22.

GUIDI, L. F. *Notas da disciplina Cálculo Numérico*. 2014. (Data de acesso: 24-11-2016). Disponível em: <http://www.mat.ufrgs.br/~guidi/grad/MAT01169/calculo_numerico.pdf>. Citado 2 vezes nas páginas 18 e 37.

HOPPE, H. et al. Surface reconstruction from unorganized points. *SIGGRAPH Comput. Graph.*, ACM, New York, NY, USA, v. 26, n. 2, p. 71–78, jul. 1992. ISSN 0097-8930. Disponível em: <http://doi.acm.org/10.1145/142920.134011>. Citado 9 vezes nas páginas 34, 43, 48, 49, 50, 51, 54, 73 e 83.

HORN, B. K. P. Extended gaussian images. *Proceedings of the IEEE*, v. 72, n. 12, p. 1671–1686, Dec 1984. ISSN 0018-9219. Citado na página 15.

IEZZI, G. *Fundamentos de matemática elementar*. 2. ed. [S.l.]: Atual Editora, 1978. v. 3. ISBN 9788535704570. Citado 4 vezes nas páginas 25, 26, 39 e 71.

JIANG, W. et al. Curve skeleton extraction by coupled graph contraction and surface clustering. *Graph. Models*, Academic Press Professional, Inc., San Diego, CA, USA, v. 75, n. 3, p. 137–148, maio 2013. ISSN 1524-0703. Disponível em: <http://dx.doi.org/10.1016/j.gmod.2012.10.005>. Citado 13 vezes nas páginas 15, 42, 43, 56, 75, 76, 77, 78, 80, 81, 82, 83 e 84.

KOBBELT, L. et al. Interactive multi-resolution modeling on arbitrary meshes. In: *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM, 1998. (SIGGRAPH '98), p. 105–114. ISBN 0-89791-999-8. Disponível em: <http://doi.acm.org/10.1145/280814.280831>. Citado na página 38.

KOLONIAS, I. et al. Fast content-based search of vrml models based on shape descriptors. *IEEE Transactions on Multimedia*, v. 7, n. 1, p. 114–126, Feb 2005. ISSN 1520-9210. Citado na página 14.

KÖNIG, S.; GUMHOLD, S. Consistent propagation of normal orientations in point clouds. In: MAGNOR, M. A.; ROSENHAHN, B.; THEISEL, H. (Ed.). *VMV*. [S.l.]: DNB, 2009. p. 83–92. ISBN 978-3-9804874-8-1. Citado 7 vezes nas páginas 50, 51, 52, 53, 54, 55 e 83.

LAINE, S.; KARRAS, T. *Efficient Sparse Voxel Octrees – Analysis, Extensions, and Implementation*. [S.l.], 2010. Citado na página 21.

LEE, I.-K. Curve reconstruction from unorganized points. *Computer Aided Geometric Design*, v. 17, n. 2, p. 161–177, 1999. Disponível em: <http://dx.doi.org/10.1016/S0167-8396(99)00044-8>. Citado 2 vezes nas páginas 46 e 47.

LEHTINEN, J. et al. A meshless hierarchical representation for light transport. *ACM Trans. Graph.*, ACM, New York, NY, USA, v. 27, n. 3, p. 37:1–37:9, ago. 2008. ISSN 0730-0301. Disponível em: <http://doi.acm.org/10.1145/1360612.1360636>. Citado na página 44.

LEVOY, M. et al. The digital michelangelo project: 3D scanning of large statues. In: AKELEY, K. (Ed.). *Siggraph 2000, Computer Graphics Proceedings*. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000. (Annual Conference Series), p. 131–144. Disponível em: <http://visinfo.zib.de/EVlib/Show?EVL-2000-49>. Citado na página 13.

MADARAS, M. Extraction of skinning data by mesh contraction with collada 1.5 support. aug 2013. Disponível em: <http://www.cescg.org/CESCG-2010/papers/MadarasMartin.pdf>. Citado na página 15.

MAMOU, K.; GHORBEL, F. A simple and efficient approach for 3d mesh approximate convex decomposition. In: *ICIP*. IEEE, 2009. p. 3501–3504. ISBN 978-1-4244-5654-3. Disponível em: <http://dblp.uni-trier.de/db/conf/icip/icip2009.html#MamouG09>. Citado na página 65.

MATHWORKS. *Estimate normals for point cloud - MATLAB pcnormals*. 2016. (Data de acesso: 30-01-2017). Disponível em: <https://www.mathworks.com/help/vision/ref/pcnormals.html>. Citado na página 48.

MATHWORKS. *MEX File Creation API*. 2016. (Data de acesso: 24-11-2016). Disponível em: <https://www.mathworks.com/help/matlab/call-mex-files-1.html>. Citado na página 19.

MATHWORKS. *mexCallMATLABWithTrap (C and Fortran)*. 2016. (Data de acesso: 24-11-2016). Disponível em: <https://www.mathworks.com/help/matlab/apiref/mexcallmatlabwithtrap.html>. Citado na página 19.

MATHWORKS. *Not-a-Number - MATLAB NaN*. 2016. (Data de acesso: 24-05-2017). Disponível em: <https://www.mathworks.com/help/matlab/ref/nan.html>. Citado na página 72.

MATHWORKS. *Tecniques to Improve Performance*. 2016. (Data de acesso: 24-11-2016). Disponível em: <https://www.mathworks.com/help/matlab/matlab_prog/techniques-for-improving-performance.html>. Citado na página 18.

MATHWORKS. *Triangular surface plot - MATLAB trisurf*. 2016. (Data de acesso: 10-01-2017). Disponível em: <https://www.mathworks.com/help/matlab/ref/trisurf.html>. Citado na página 95.

MATHWORKS. *Using Vectorization*. 2016. (Data de acesso: 24-11-2016). Disponível em: <https://www.mathworks.com/help/matlab/matlab_prog/vectorization.html>. Citado na página 18.

MELO, A. L. *Baricentro do triângulo - O que é e como determinar suas coordenadas*. 2015. (Data de acesso: 26-02-2017). Disponível em: <http://www.estudopratico.com.br/baricentro-do-triangulo-o-que-e-e-como-determinar-suas-coordenadas/>. Citado na página 58.

MEYER, M. et al. Discrete differential-geometry operators for triangulated 2-manifolds. *Visualization and mathematics*, Citeseer, v. 3, n. 7, p. 34–57, 2002. Citado na página 63.

MODERNA, B. E. E. *CAD, CAE e CAM: qual a diferença?* 2016. (Data de acesso: 15-11-2016). Disponível em: <http://www.cim-team.com.br/blog-engenharia-eletrica-moderna/cad-cae-e-cam-qual-a-diferenca>. Citado na página 13.

NEALEN, A. et al. Laplacian mesh optimization. In: *Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia*. New York, NY, USA: ACM, 2006. (GRAPHITE '06), p. 381–389. ISBN 1-59593-564-9. Disponível em: <http://doi.acm.org/10.1145/1174429.1174494>. Citado 2 vezes nas páginas 38 e 70.

NEHAB, D. et al. Efficiently combining positions and normals for precise 3D geometry. *ACM Trans. Graph*, v. 24, n. 3, p. 536–543, 2005. Disponível em: <http://doi.acm.org/10.1145/1073204.1073226>. Citado na página 43.

OLIVEIRA, L. E. S. *Reconhecimento de Padrões: PCA*. 2010. (Data de acesso: 15-07-2015). Disponível em: <http://www.inf.ufpr.br/lesoliveira/padroes/pca.pdf>. Citado na página 31.

OLIVEIRA, S. R. de M. *Métodos usados para redução e sintetização de dados*. 2012. (Data de acesso: 27-11-2015). Disponível em: <http://www.ime.unicamp.br/~wanderson/Aulas/MT803_Aula3_Reducao_Sintetizacao_Dados.pdf>. Citado na página 31.

OVSJANIKOV, M. et al. Exploration of continuous variability in collections of 3D shapes. *ACM Transactions on Graphics*, v. 30, n. 4, p. 33:1–33:??, jul 2011. ISSN 0730-0301 (print), 1557-7368 (electronic). Citado na página 13.

OWEN, G. S. *Polygon Decomposition into Triangles*. 1998. (Data de acesso: 15-07-2017). Disponível em: <https://www.siggraph.org/education/materials/HyperGraph/scanline/outprims/polygon1.htm>. Citado 2 vezes nas páginas 94 e 95.

PAPADAKIS, P. et al. 3d object retrieval using an efficient and compact hybrid shape descriptor. In: PERANTONIS, S. J. et al. (Ed.). *3DOR*. Eurographics Association, 2008. p. 9–16. ISBN 978-3-905674-05-7. Disponível em: <http://dx.doi.org/10.2312/3DOR/3DOR08/009-016>. Citado 2 vezes nas páginas 13 e 17.

PEIRCE, A. *Lecture 24: Laplace's Equation*. 2014. (Data de acesso: 21-05-2017). Disponível em: <https://www.math.ubc.ca/~peirce/M257_316_2012_Lecture_24.pdf>. Citado na página 37.

PITERI, M. A. et al. Triangulação de delaunay e o princípio de inserção randomizado. *II Simpósio Brasileiro de Geomática - V Colóquio Brasileiro de Ciências Geodésicas*, Presidente Prudente - SP, p. 9, 2007. Citado na página 33.

POINTCLOUD.ORG. 2011. (Data de acesso: 15-06-2015). Disponível em: <http://pointclouds.org/about/>. Citado na página 20.

POWER, K. *3D Object Representations*. 2012. (Data de acesso: 24-11-2016). Disponível em: <http://glasnost.itcarlow.ie/~powerk/GeneralGraphicsNotes/meshes/polygon_meshes.html>. Citado na página 21.

SHAPEQUEST. 1999. (Data de acesso: 22-11-2016). Disponível em: <http://www.shapecapture.com/SC_Poly.htm>. Citado na página 20.

SMITH, L. I. *A tutorial on Principal Components Analysis*. 2002. (Data de acesso: 15-07-2015). Disponível em: <http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf>. Citado na página 30.

SORKINE, O. Laplacian Mesh Processing. In: CHRYSANTHOU, Y.; MAGNOR, M. (Ed.). *Eurographics 2005 - State of the Art Reports*. [S.l.]: The Eurographics Association, 2005. Citado 2 vezes nas páginas 39 e 69.

STEINBRUCH, A.; WINTERLE, P. *Geometria analítica*. 2. ed. [S.l.]: Pearson Education, 1987. ISBN 9780074504093. Citado 10 vezes nas páginas 23, 24, 25, 26, 27, 28, 45, 46, 58 e 71.

STEINBRUCH, A.; WINTERLE, P. *Álgebra linear*. 2. ed. [S.l.]: Pearson Makron Books, 1987. ISBN 9780074504123. Citado na página 29.

SUZUKI, M. T.; KATO, T.; OTSU, N. A similarity retrieval of 3d polygonal models using rotation invariant shape descriptors. In: *IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC2000*. [S.l.: s.n.], 2000. p. 2946–2952. Citado na página 14.

SUZUKI, M. T.; YAGINUMA, Y.; SUGIMOTO, Y. Y. A 3d model retrieval system for cellular phones. In: *Systems, Man and Cybernetics, 2003. IEEE International Conference on*. [S.l.: s.n.], 2003. v. 4, p. 3846–3851 vol.4. ISSN 1062-922X. Citado na página 14.

TAGLIASACCHI, A. et al. Mean curvature skeletons. *Comput. Graph. Forum*, The Eurographs Association & John Wiley & Sons, Ltd., Chichester, UK, v. 31, n. 5, p. 1735–1744, ago. 2012. ISSN 0167-7055. Disponível em: <http://dx.doi.org/10.1111/j.1467-8659.2012.03178.x>. Citado 5 vezes nas páginas 42, 43, 72, 83 e 84.

TAGLIASACCHI, A.; ZHANG, H.; COHEN-OR, D. Curve skeleton extraction from incomplete point cloud. *ACM Trans. Graph.*, ACM, New York, NY, USA, v. 28, n. 3, p. 71:1–71:9, jul. 2009. ISSN 0730-0301. Disponível em: <http://doi.acm.org/10.1145/1531326.1531377>. Citado 11 vezes nas páginas 15, 42, 43, 44, 45, 46, 47, 48, 82, 83 e 84.

TANGELDER, J. W. H.; VELTKAMP, R. C. A survey of content based 3d shape retrieval methods. *Multimedia Tools Appl*, v. 39, n. 3, p. 441–471, 2008. Disponível em: <http://dx.doi.org/10.1007/s11042-007-0181-0>. Citado na página 16.

TAUBIN, G. A signal processing approach to fair surface design. In: *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM, 1995. (SIGGRAPH '95), p. 351–358. ISBN 0-89791-701-4. Disponível em: <http://doi.acm.org/10.1145/218380.218473>. Citado na página 38.

TECMUNDO. *7 áreas que poderão ser beneficiadas por impressoras 3D*. 2013. (Data de acesso: 15-11-2016). Disponível em: <http://www.tecmundo.com.br/impressora-3d/44064-7-areas-que-poderao-ser-beneficiadas-por-impressoras-3d.htm>. Citado na página 13.

VALETTE, S.; CHASSERY, J.-M. Approximated Centroidal Voronoi Diagrams for Uniform Polygonal Mesh Coarsening. *Computer Graphics Forum*, The Eurographics Association and Blackwell Publishing, Inc, 2004. ISSN 1467-8659. Citado 11 vezes nas páginas 15, 32, 56, 57, 58, 59, 60, 61, 75, 80 e 83.

VARELLA, C. A. A. *Analise dos Componentes Principais*. 2008. (Data de acesso: 20-07-2015). Disponível em: <http://www.ufrrj.br/institutos/it/deng/varella/Downloads/multivariada%20aplicada%20as%20ciencias%20agrarias/Aulas/analise%20de%20componentes%20principais.pdf>. Citado na página 31.

VENTURI, J. J. *Álgebra Vetorial e Geometria analítica*. 10. ed. [S.l.]: Livrarias Curitiba, 2015. ISBN 8585132485. Citado 6 vezes nas páginas 22, 23, 24, 26, 27 e 28.

VRANIC, D. 3D model retrieval. In: *Ph.D.* [S.l.: s.n.], 2004. Citado 2 vezes nas páginas 14 e 15.

WEISSTEIN, E. W. *Perpendicular Vector – from Wolfram MathWorld*. 1994. (Data de acesso: 18-02-2017). Disponível em: <http://mathworld.wolfram.com/PerpendicularVector.html>. Citado na página 53.

XIE, H. et al. Piecewise c1 continuous surface reconstruction of noisy point clouds via local implicit quadric regression. In: *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*. Washington, DC, USA: IEEE Computer Society, 2003. (VIS '03), p. 13–. ISBN 0-7695-2030-8. Disponível em: <http://dx.doi.org/10.1109/VISUAL.2003.1250359>. Citado 5 vezes nas páginas 50, 51, 52, 54 e 83.

XIE, W. J.; THOMPSON, R. P.; PERUCCHIO, R. A topology-preserving parallel 3D thinning algorithm for extracting the curve skeleton. *Pattern Recognition*, v. 36, n. 7, p. 1529–1544, jul. 2003. Disponível em: <http://www.sciencedirect.com/science/article/B6V14-484V86D-1/2/843029e113a3781bd5feb1f7b82f916b>. Citado na página 21.

YANG, X. W. et al. Shape classification based on skeleton path similarity. In: *EMMCVPR*. [s.n.], 2007. p. 375–386. Disponível em: <http://dx.doi.org/10.1007/978-3-540-74198-5_29>. Citado na página 16.

YANG, Y.; LIN, H.; ZHANG, Y. Content-based 3-D model retrieval: A survey. *IEEE Trans. Systems, Man and Cybernetics*, v. 37, n. 6, p. 1081–1098, nov 2007. Disponível em: <http://dx.doi.org/10.1109/TSMCC.2007.905756>. Citado 5 vezes nas páginas 13, 14, 15, 16 e 17.

ZHANG, C.; CHEN, T. Efficient feature extraction for 2d/3d objects in mesh representation. In: *Proceedings 2001 International Conference on Image Processing (Cat. No.01CH37205)*. [S.l.: s.n.], 2001. v. 3, p. 935–938 vol.3. Citado 2 vezes nas páginas 69 e 83.

APÊNDICE A – Arquivos Wavefront Object

Arquivos *Wavefront Object* são utilizados para armazenar informações que representam a geometria de um objeto tridimensional. Tais arquivos estão disponíveis tanto em formato *ASCII* ou em formato binário, sendo definidos pela extensão *obj* e *mod*, respectivamente. Arquivos de extensão *obj* podem incluir informações de vértices, faces, polígonos, linhas e entre outras de um objeto 3D (BOURKE, 2016).

Os **vértices** de um objeto 3D são representados por listas, conforme o tipo de coordenada: vértices geométricos, vértices de textura e normal do vértice. Vértices paramétricos são fornecidos para objetos de forma livre (BOURKE, 2016).

Os **vértices geométricos** especificam a posição geométrica do ponto no espaço, representado pela sintaxe `v x y z w`. A letra `v` indica que trata-se de um vértice geométrico. Os valores para `x`, `y` e `z` informam a posição do vértice nos eixos de coordenadas **X**, **Y** e **Z**. Para superfícies e curvas racionais, é utilizado uma quarta coordenada `w`, denominada peso, cujo valor padrão é 1 (BOURKE, 2016). A linha a seguir representa um exemplo de vértice geométrico:

```
v -1.000000 3.000000 -2.340000
```

Os **vértices de textura** são representado pela sintaxe `vt u v w`. As coordenadas `u v w` informam os valores para a direção horizontal, direção vertical e profundidade da textura, respectivamente. Os valores de `v` e `w` não são obrigatórios, sendo por padrão atribuído a 0 (BOURKE, 2016). A linha a seguir representa um exemplo de vértice de textura:

```
vt -3.000000 5.000000 0.000000
```

Os **vetores normais dos vértices** são representados pela sintaxe `vn i j k`. Os valores `i`, `j` e `k` são as coordenadas canônicas do vetor normal expresso conforme Equação 3.2 (BOURKE, 2016). A linha a seguir representa um exemplo de vetor normal do vértice:

```
vn 0.000000 0.000000 1.000000
```

Cada vértice de cada tipo é enumerado de acordo com a sua sequência no arquivo, do início ao fim, iniciando com 1. Isto significa que cada i-ésimo vértice geométrico representado no arquivo será enumerado com o valor de i. De modo análogo, cada j-ésimo

vértice de textura será enumerado com o valor de j e cada k-ésimo vetor normal do vértice. Estes rótulos podem ser utilizados para referenciar os vértices em elementos como faces ou superfícies (BOURKE, 2016).

As **faces** são representadas por um conjunto de vértices. Cada vértice pertencente a uma face é expresso pelo seu rótulo ou enumeração como sendo seu índice na lista de vértices. Sintaticamente, inicia com a letra `f` seguido com os índices de cada vértice em sequência (BOURKE, 2016). A linha a seguir representa um exemplo de face:

```
f 1 3 4
```

A representação de cada vértice pode incluir vértices de textura e vetores normais casos existirem. Seja `vi`, `vtj` e `vnk` os índices (ou enumerações) para um vértice geométrico, vértice de textura e vetor normal para vértice, respectivamente. Cada vértice de uma face será representado pela sintaxe `vi/vtj/vnk` ao informar os índices dos vértices geométricos, vértices de textura e vetores normais. Caso um vértice não contém informação de textura é dada pela sintaxe `vi//vnk` (BOURKE, 2016). Segue abaixo exemplos de representação sintática de uma face, onde o último deles não apresenta informações de textura:

```
f 2/2/2 8/8/8 9/9/9
f 14//14 17//17 18//18
```

Pode-se extrair uma malha triangular segundo a representação na Tabela 1. Para isto basta construir uma lista contendo os vértices geométricos e uma lista de faces contendo os índices dos três vértices que pertencem a uma face. Caso a face não seja triangular, Owen (1998) apresenta um meio de particionar uma face poligonal em uma face triangular, conforme exemplificado pela Figura 51.

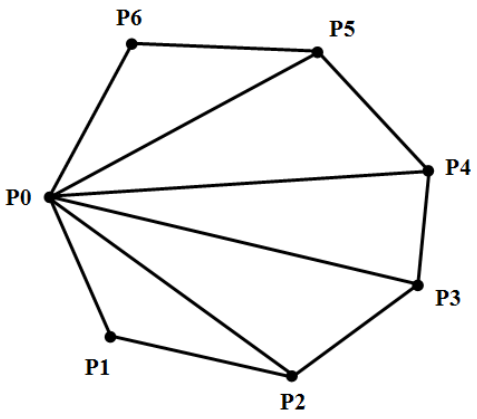

Figura 51 – Convertendo uma face poligonal em uma face triangular
Fonte: Owen (1998) adaptada pelo autor

Considerando o exemplo de um arquivo representando a forma tridimensional de um cubo [1] (Figura 52).

```
v -0.5 -0.5 0.5
v -0.5 -0.5 -0.5
v -0.5 0.5 -0.5
v -0.5 0.5 0.5
v 0.5 -0.5 0.5
v 0.5 -0.5 -0.5
v 0.5 0.5 -0.5
v 0.5 0.5 0.5
f 4 3 2 1
f 2 6 5 1
f 3 7 6 2
f 8 7 3 4
f 5 8 4 1
f 6 7 8 5
# End of file
```

Ao extrair os vértices e faces conforme Bourke (2016) e ao converter as faces quadrangulares em faces triangulares conforme Owen (1998), são obtidas as seguintes listas de vértices e faces apresentadas pela Tabela 6:

Utilizando o MATLAB é possível visualizar o objeto 3D extraído de um arquivo `obj` através do comando `trisurf(F, X, Y, Z)`, onde `F` é uma matriz de m linhas e 3 colunas representando a tabela de faces. As matrizes `X`, `Y` e `Z` são matrizes colunas de tamanho n que representam, respectivamente, as coordenadas X, Y e Z da matriz de vértices (MATHWORKS, 2016f). A Figura 52 mostra o objeto 3D da Tabela 6 visualizado no MATLAB:

[1] Link para arquivo box.obj: http://paulbourke.net/dataformats/obj/box.obj

Tabela de vértices	
1	-0.5, -0.5, 0.5
2	-0.5, -0.5, -0.5
3	-0.5, 0.5, -0.5
4	-0.5, 0.5, 0.5
5	0.5, -0.5, 0.5
6	0.5, -0.5, -0.5
7	0.5, 0.5, -0.5
8	0.5, 0.5, 0.5

(a) Tabela da Vértices

Tabela de faces	
1	4 3 2
2	4 2 1
3	2 6 5
4	2 5 1
5	3 7 6
6	3 6 2
7	8 7 3
8	8 3 4
9	5 8 4
10	5 4 1
11	6 7 8
12	6 8 5

(b) Tabela da Faces

Tabela 6 – Exemplo de tabela de vértices e faces triangulares extraídas de um arquivo obj

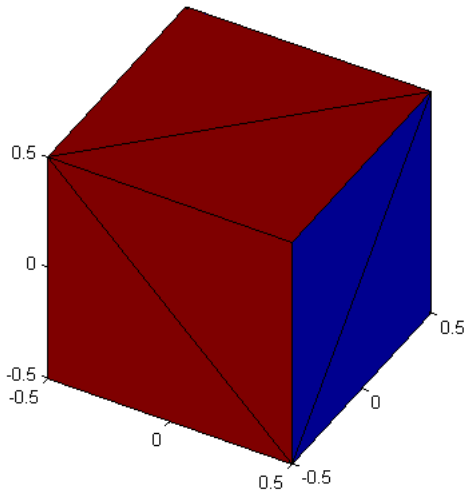

Figura 52 – Exemplo de plotagem de um objeto 3D utilizando trisurf
Fonte: Elaborada pelo autor