Carolina dos Santos Oliveira Viana

Reconstrução Tridimensional de Imagens Transversais para Prototipagem Rápida

Monografia de Graduação

Ciência da Computação

Universidade Estadual do Norte Fluminense - Darcy Ribeiro

Campos dos Goytacazes, 12 de julho de 2018.

Carolina dos Santos Oliveira Viana

Reconstrução Tridimensional de Imagens Transversais para Prototipagem Rápida

Monografia apresentada ao Curso de Ciência da Computação da Universidade Estadual do Norte Fluminense Darcy Ribeiro, como parte dos requisitos para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Luis Antonio Rivera Escriba, DSc

Ciência da Computação

Universidade Estadual do Norte Fluminense Darcy Ribeiro.

Campos dos Goytacazes, 12 de julho de 2018.

Carolina dos Santos Oliveira Viana

Reconstrução Tridimensional de Imagens Transversais para Prototipagem Rápida

Monografia apresentada ao Curso de graduação em Ciência da Computação da Universidade Estadual do Norte Fluminense Darcy Ribeiro, requisito para a obtenção do título de Bacharel em Ciência da Computação.

Aprovada em 12/07/2018

COMISSÃO EXAMINADORA

_____

Prof. Dr. Luis Antonio Rivera Escriba
Orientador – Universidade Estadual do Norte Fluminense Darcy Ribeiro (UENF)

_____

Prof. Dr. Fermín Alfredo Tang Montané
Universidade Estadual do Norte Fluminense Darcy Ribeiro (UENF)

_____

Prof. Dr. Ausberto Silverio Castro Vera
Universidade Estadual do Norte Fluminense Darcy Ribeiro (UENF)

"It took a while, now I understand just where I'm goin'
I know the world and I know who I am
'Bout time I show it"
Beyoncé

Agradecimentos

Agradeço ao meu orientador, o professor Rivera, pelo total apoio e paciência na realização deste trabalho, apoio sem o qual não seria possível a conclusão do mesmo. Agradeço aos meus pais e meus irmãos, que sempre me incentivaram a finalizar a graduação. Ao meu namorado que sempre esteve ao meu lado me incentivando com veemência para terminar este trabalho. Aos meus amigos que já não me cumprimentavam mais com "oi", mas com "cadê a sua monografia?". A todos aqueles que me impulsionaram a chegar até aqui, muito obrigada pelo apoio!

Resumo

Exames médicos como a tomografia computadorizada (TC) e a ressonância magnética (RM) geram imagens fatiadas ou transversais de um objeto, normalmente um corpo, fazendo com que muitas vezes a "costura" dessas imagens, para uma melhor visualização de uma determinada parte deste corpo, se torne um trabalho de imaginação. Isto cria uma demanda que esse trabalho tem como objetivo suprir ao criar um programa que realizará a reconstrução tridimensional de um sólido que poderá ser posteriormente impresso em 3D, tendo como base essas imagens fatiadas. A reconstrução é composta por três etapas: pré-processamento, reconstrução tridimensional e impressão 3D. Na etapa de pré-processamento ocorre um tratamento das imagens adquiridas para remoção de ruídos e criação de pontos. A etapa de reconstrução utiliza um algoritmo de triangulação para criação de uma malha de triângulos que reconstruirá o sólido, e conta com um algoritmo de criação de fecho convexo visando assegurar um bom resultado da triangulação. Na última etapa o sólido reconstruído é preparado para a impressão 3D. Utilizando imagens sintéticas, ao final das três etapas obteve-se um sólido em 3D, posteriormente impresso, e permitindo assim uma visualização completa do objeto em questão.

Abstract

Medical exams such as Computed Tomography (CT) and Magnetic Resonance Imaging (MRI) generate sliced or transverse images of an object, usually parts of the human body, often making the "stitching"of these images, for a better visualization of a certain part of this body, become a work of imagination. This creates a demand that this work aims to supply by creating a program that will make a three-dimensional reconstruction of a 3D object, that can be printed in a 3D printer, based on these sliced images. The reconstruction is composed of three stages: pre-processing, three-dimensional reconstruction and 3D printing. In the pre-processing stage, occurs a treatment of the acquired images for noise removal and points identification. The reconstruction step uses a triangulation algorithm to create a triangle mesh to reconstruct the object, and contains a convex hull algorithm to ensure a good result. The last step of the solid reconstruction will be the preparation for 3D printing. Using synthetic images, at the end of the three steps a solid was obtained in 3D, later printed, and allowing in this way a complete visualization of the object in question.

Sumário

Lista de Figuras

Capítulo 1

Introdução

Existem situações em que um objeto existe como parte de outro, mas não é visível naturalmente, senão entendido por deduções de especialistas que desejam analisar a forma e características do mesmo para as ações desejadas. Isso se dá a partir de informações como imagens de seções transversais: tal como ocorre na área médica ou industrial – na qual o especialista infere a característica do órgão de interesse, por exemplo, mediante uma série de imagens tomográficas ou ressonâncias magnéticas.

Existem casos que o especialista da área gostaria de visualizar ou manipular o órgão íntegro em forma volumétrica, como por exemplo, para o domínio das dimensões exatas para um melhor entendimento de suas propriedades, ou para possivelmente recreação de um protótipo para substituição – caso de cirurgias e transplantes dos órgãos no corpo – ou simplesmente para uma visualização mais exata das partes ocultas do órgão e suas classificações.

A demanda para esse propósito é reconstruir em forma tridimensional o órgão de interesse a partir das placas de imagens de seções transversais. Uma vez tendo o órgão reconstruído como um objeto tridimensional, em função dos atributos geométricos de superfícies e texturas pertinentes, pode ser reproduzido como um objeto sólido através de uma impressora 3D, tecnologia útil nos últimos anos em muitas áreas de modelagens. Essa tecnologia, também conhecida como máquina de prototipagem rápida [5], permite a visualização tridimensional de objetos que é bastante útil em diversas áreas da ciência aplicada atual como uma ferramenta para auxiliar nas pesquisas.

A reconstrução tridimensional de objetos pode ser feita através de diversas técnicas, variando de acordo com a necessidade do modelo, e os recursos disponíveis. Entre alguns exemplos temos: a fotogrametria [10] que criará uma nuvem de pontos baseada em fotografias tiradas de todos os ângulos de um objeto; *Structure from Motion (SfM)* [17] que cria modelos 3D de cenas também baseadas em fotos, mas adicionando o movimento realizado de deslocamento da câmera; scanners 3D [8] dos mais diversos, desde do tipo "faça você mesmo" até os mais complexos como os utilizados na área médica.

Em específico, no caso médico, os modelos tridimensionais podem ser reconstruídos através de uma sequência de imagens de seções transversais ou fatias que representam o objeto de estudo, essas imagens podem ser obtidas através de dois equipamentos, geralmente utilizados na área médica, mas que possuem uma extensa gama de aplicações em outras áreas, conhecidos como tomografia computadorizada (TC) e ressonância magnética(RM). Cada um dos dois equipamentos possuem processos diferentes de geração de imagens, mas em ambos o produto final é um arquivo contendo imagens de seções transversais do objeto examinado. A reconstrução associada a técnica de prototipagem rápida permite que os médicos consigam visualizar detalhes que poderiam ser difíceis de identificar através de uma sequência de imagens.

1.1 Objetivos

Este trabalho tem como principal objetivo a reconstrução tridimensional de um objeto através de imagens de seções transversais, no estilo gerado por imagens tomográficas. Esse modelo reconstruído em três dimensões será então impresso também em três dimensões por uma máquina de prototipagem rápida.

1.2 Motivações

A principal motivação para este trabalho é a de utilizar uma tecnica, relativamente nova [5] de fácil acesso e baixo custo, que é a impressora tridimensional, associada a exames médicos bastante utilizados há muitos anos por médicos do mundo todo, para criar uma ferramenta que dê suporte aos médicos na tomada de decisões para

a realização de diagnósticos dos mais variados possíveis, e aproximar a interação médico-paciente através de uma melhor visualização e entendimento de possíveis procedimentos médicos.

1.3 Metodologia

A sequência seguida neste trabalho é composta por três etapas:pré-processamento, reconstrução e prototipagem rápida. A primeira etapa é a de pré-processamento, na qual é feito o tratamento das imagens semelhantes às de uma TC para iniciar o processo de reconstrução. A segunda etapa é composta pela reconstrução em si, onde as fatias serão "costuradas" uma em cima da outra, formando uma malha de triângulos que representará tridimensionalmente o sólido reconstruído. E para finalizar na terceira etapa é realizada a adaptação do modelo para impressão 3D, e em seguida a impressão de fato.

Na etapa de reconstrução tridimensional é abordada a relação dos componentes do objeto de estudo. Em cada fatia adjacente é selecionada uma área de interesse, conhecida como ROI (*Region Of Interest*), que então com base das teorias de geometria computacional no que diz respeito a métodos de construção de fechos e fechos convexos de pontos de contornos é possível reconstruir o objeto. As implementações podem ser feitas com diversas bibliotecas, como por exemplo: *OpenGL* [9], *Python Photogrammetry toolbox* [10], destaque para o *OpenCV* [8]; [12]. Ou podem ser utilizados programas já consolidados no mercado como o *InVesalius*, *Blender* e *MeshLab* [10].

São utilizados neste trabalho em particular as bibliotecas *OpenCV* para preparação e processamento de imagens tomográficas afim de se obter os contornos do objeto da seção transversal, e *OpenGL* para a reconstrução da superfície unindo pontos correspondentes entre contornos consecutivos e geração da malha de triângulos. A linguagem C e C++ são usadas na programação para produzir aplicativos de manipulação em tempo real.

1.4 Aplicações

Este trabalho pode ser aplicado a todas as situações de reconstrução tridimensional baseadas em superfícies, que utilizem imagens transversais como sua fonte de dados, onde gera-se uma sequência de pontos que possuem uma orientação e estejam dispostos ao longo de contornos presentes em imagens, que ao serem "empilhadas" uma em cima da outra, formem um conjunto de pontos que quando associados a orientação possuam uma estrutura já conhecida que facilitará o processo de geração de uma malha triangular que envolva todos esses pontos e que ao final forme um objeto tridimensional. Dentre algumas áreas que podem se beneficiar desta tecnologia podemos destacar como exemplo: auxiliar em diversas decisões médicas das mais variadas áreas [1], dar suporte a processos cirúrgicos, criação de moldes para próteses, reconstrução forense, maximizar produção industrial produzindo modelos para inspeção de processos [7], arqueologia [10], topografia de terrenos, indústria cinematográfica [8] dentre outros.

1.5 Organização do trabalho

Este trabalho está dividido em 5 capítulos da seguinte forma:

O Capítulo 2 resume a referência teórica base para produção deste trabalho. Explica sucintamente a diferença entre os dois tipos de reconstrução: a volumétrica e a de superfície. Em seguida fala sobre os métodos de obtenção das imagens médicas. É discutida a fase de pré-processamento das imagens e os algoritmos envolvidos. E para finalizar explica o processo de impressão 3D.

O Capítulo 3 mostra a explicação teórica dos processos da fase de reconstrução, no qual é criado o fecho convexo de um contorno exemplo, para em seguida prosseguir para a parte da criação da malha de triângulos.

O Capítulo 4 descreve toda a implementação realizada no programa desenvolvido neste trabalho iniciando com o detalhamento de todo o processo, e incluindo detalhes como a estrutura de dados utilizada, e resultados.

O Capítulo 5 finaliza o trabalho com a conclusão, e sugestões de trabalhos futuros.

Capítulo 2

Reconstrução 3D de Imagens de Seções Transversais

A reconstrução 3D pode ser feita a partir de uma sequência de imagens que correspondem às seções transversais de um sólido capturadas através de tomografias computarizadas (TC) e ressonâncias magnéticas (RM). Tendo como exemplo de aplicação a área médica, utilizam-se imagens tomográficas, para reconstruir o objeto sólido do corpo para uma melhor análise [3]. Tipicamente, um modelo de reconstrução 3D de imagens de seções transversais é composto por 3 fases: pré-processamento, reconstrução e impressão 3D.

Existem diversos métodos para realizar a reconstrução tridimensional de objetos através de imagens de seções transversais, os que se destacam podem ser divididos entre métodos que são baseados no volume do objeto a ser reconstruído, e métodos baseados na superfície do objeto a ser reconstruído [1]. Os métodos de reconstrução baseados em volume não têm como principal objetivo a identificação dos contornos, e sim uma aproximação de sua silhueta [4]. Esses métodos são baseados no empilhamento das imagens, e da interpolação das mesmas para preenchimento dos espaços entre uma imagem e outra na pilha [3]. São utilizadas *octrees* que envolverão todo o objeto de estudo criando uma *bounding box* [4], uma caixa delimitante, e a partir deste ponto é feito o refinamento do objeto. *Octree* é uma árvore na qual o nó inicial representa o objeto dentro de sua *bounding box*, e cada um de seus nós que não são folhas possuem oito nós filhos. Os nós da árvore recebem o nome de *voxel*. Cada *voxel* é dividido em outros 8 sub-*voxels*, que irão receber uma de três possíveis classificações de acordo com

o seu volume em relação ao objeto: dentro, para *voxels* com todo o seu volume dentro do objeto; fora, para *voxels* com o volume fora do objeto; e ambíguo, para *voxels* que possuem parte do seu volume dentro do objeto, e parte do seu volume fora do objeto. Enquanto houverem *voxels* ambíguos são divididos em 8 *voxels*, e repetidos os processos anteriores de classificação. Na Figura 2.1 podemos ver a evolução do processo de refinamento.

Figura 2.1: Reconstrução 3D de uma mão pelo método volumétrico, com 3, 5 e 7 níveis de refinamento da *octree* [4].

Os métodos baseados em superfícies vão extrair das imagens das seções transversais um conjunto de contornos que formam os limites do objeto. Esses contornos representam a superfície externa do objeto, e são representadas por um número finito de curvas fechadas. É distribuída uma série de pontos sobre esses contornos, e a reconstrução da superfície externa é feita a partir da união desse conjunto de pontos entre as fatias adjacentes. Para realizar a união os pontos são unidos de forma a gerarem polígonos, geralmente triângulos, criando um mosaico. Em geral, esses métodos são divididos em dois tipos: ótimos e heurísticos [1]. Os métodos ótimos são baseados em critérios globais, e geram o melhor resultado para criação das faces triangulares através de muito processamento tornando o processo custoso. Já os métodos heurísticos utilizam critérios locais para criar a malha triangular, e geram resultados próximos o suficiente ao ótimo para que sejam aceitos. Na Figura 2.2 podemos ver a malha de triângulos criada no processo de reconstrução tridimensional de uma escapula.

As técnicas de obtenção de imagens tomográficas e as técnicas de prototipagem rápida tem em comum o fato de utilizarem camadas ou seções transversais. Essas similaridades por si só acabam estimulando a união das duas técnicas, o que pode trazer vários benefícios a aplicações nas diversas áreas, como por exemplo a área médica [1].

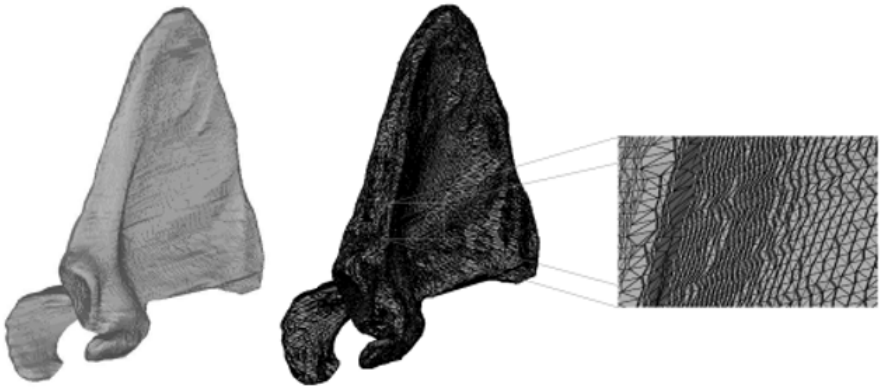


Figura 2.2: Escapula. Visualização da reconstrução e modelo triangulado [1].

As imagens geradas pelo tomógrafo não são automaticamente reconhecidas pela máquina de prototipagem rápida, as espessuras das imagens tomográficas variam de 1mm a 5mm, enquanto as espessuras das camadas na máquina de prototipagem variam de 0,1mm a 0,3mm. Dessa forma é necessário que haja o processamento das imagens de forma a preencher os espaços entre as camadas das imagens transversais, e assim produzir um objeto tridimensional. Além deste fato, o tomógrafo produz arquivos em formato diferente do utilizado na impressora 3D, desta forma é necessário converter as imagens tomográficas para o formato *STL (STereo Litography)*, que é o formato que a máquina de prototipagem reconhece.

Com relação aos métodos de reconstrução apresentados, a desvantagem dos métodos baseados em volume é a grande quantidade de dados necessários para serem processados e armazenados, gerando um grande esforço computacional [2]. A vantagem dos métodos baseados em superfícies é que só é necessária a identificação dos contornos entre as fatias, um conjunto de pontos distribuídos nesses contornos, e polígonos (normalmente triângulos) que façam a união dessas camadas através dos pontos. O método escolhido foi o baseado em superfícies, pois a visualização da triangulação feita pela união dos pontos nas camadas é facilmente convertida em um arquivo *STL*, o mesmo da máquina de prototipagem para sua impressão 3D.

2.1 Imagens de seções transversais

Imagens de seções transversais são aquelas imagens que registram os detalhes de contornos do corte transversal de um objeto. Cada imagem irá representar uma fatia do objeto em estudo. Nessa categoria tem-se cortes de seções transversais de madeiras, rochas, etc. para a identificação de propriedades de formação ou estado dessas matérias. Na medicina, esse tipo de imagens são formadas através de exames conhecidos como TC e RM. A partir dessas imagens, quando geradas em sequencias de seções transversais, é possível realizar a reconstrução tridimensional de um objeto de interesse.

DEFINIÇÃO 2.1: Contorno é uma linha divisória no plano que separa o interior do exterior de um objeto, formando um segmento fechado.

DEFINIÇÃO 2.2: Fatia representa uma imagem transversal de um objeto de estudo. Cada fatia é composta por contornos e sub-contornos que compõem o objeto em análise [16].

A TC é uma técnica baseada em Raios-X [3]. Os Raios-X são emitidos por uma fonte, para que atravessem o corpo ou objeto de estudo. O objeto então acaba absorvendo parte desses raios, e os raios que não foram absorvidos atingem os sensores de Raios-X, que então enviam os dados coletados para um computador que gerará uma imagem em duas dimensões (2D) e em escala de cinza, baseada na absorção desses raios. O equipamento que realiza a TC é chamado de tomógrafo.

Para entender a imagem gerada pelo tomógrafo é importante saber que materiais menos densos absorvem uma menor quantidade de raios, enquanto materiais mais densos absorvem uma quantidade maior desses raios [3]. Dessa forma se o objeto de estudo for o corpo humano, é possível diferenciar os diversos tecidos que o compõe como, por exemplo, pele e ossos como pode ser visto na Figura 2.3.

A TC se diferencia de técnicas mais comuns de Raios-X por permitir que seja feito uma sequência de imagens. Essas imagens são representações bidimensionais da estrutura anatômica de um corpo, ou seja, é como se um corpo fosse fatiado, e cada uma dessas fatias fosse fotografada. As sequências dessas imagens bidimensionais geradas mostram os contornos dos órgãos e o interior do corpo ou objeto em forma de curvas simples ou de polígonos fechados [1] como mostrado na Figura 2.4 podemos

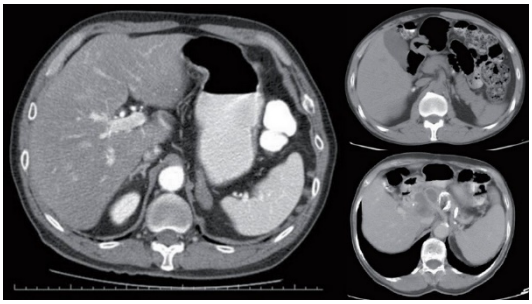


Figura 2.3: Imagem tomográfica do abdômen.

ver a aplicação da TC a área da industria, especificamente neste caso temos imagens tomográficas de um *riser*, que consiste em um tubo que conecta uma plataforma de petróleo a um sistema submarino.

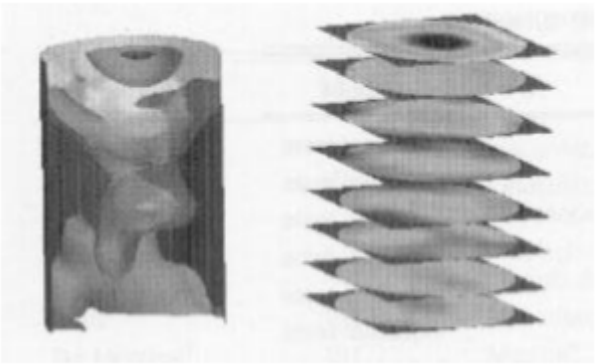


Figura 2.4: Sequência de imagens que formam as seções transversais de um *riser* [7].

Já a RM tem como ideia base a exposição de determinados elementos a um forte campo magnético, que serão então excitados por ondas de rádio em uma determinada frequência (Frequência de Larmor) e serão captados por uma antena e transformados em imagens. Os prótons presentes no núcleo dos elementos possuem uma pequena carga elétrica, que quando excitadas giram sobre seu próprio eixo fazendo um movimento chamado de *spin*. O hidrogênio possui em seu núcleo um único próton tendo a capacidade de produzir o melhor sinal de todos os núcleos estáveis, e devido a abundância de sua presença no corpo humano ele se torna o elemento mais apropriado para realizar o exame [21].

2.2 Pré-processamento

O pré-processamento das imagens é a primeira fase do processo de reconstrução tridimensional. Essa fase tem como objetivo a preparação das imagens das seções transversais obtidas no tomógrafo. Para a construção do modelo em 3D é necessário que as bordas ou os contornos dos objetos sejam descritas por uma sequência de pontos, que quando conectados entre si através de arestas formem um objeto em três dimensões. Dessa forma é importante que essas bordas sejam corretamente identificadas [1].

2.2.1 Tratamento das Imagens

Nesta primeira fase, as imagens são tratadas através de técnicas de processamento de imagens, tendo como principal objetivo colocar em evidencia os detalhes que permitam a identificação dos conjuntos de pontos em cada fatia que representem as bordas do objeto de estudo. Na Figura 2.5 é mostrado o estado inicial de duas fatias de imagens tomográficas da escápula, um osso do corpo humano que fica localizado na parte superior traseira do tórax.

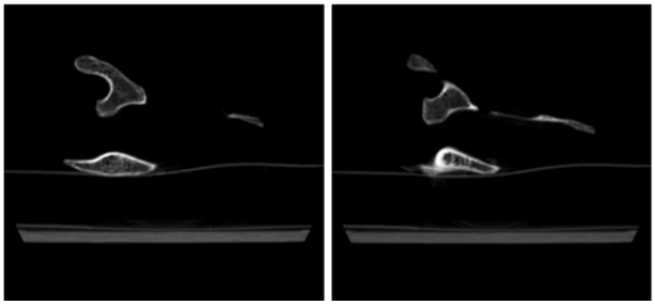


Figura 2.5: Imagens tomográficas da escápula [1].

As imagens passam por 4 processos: segmentação, detecção de bordas e captura de pontos.

Nas imagens no formato bitmap são identificadas as "áreas de interesse" também conhecidas como ROI, e é feito um recorte para que sejam isoladas do restante da imagem de forma a reduzir o ruído que estruturas próximas venham a gerar no processamento e posteriormente converter a imagem em binário conhecido como limiarização. Na Figura 2.6 é mostrada o segmento de ROI da imagem da Figura 2.5.

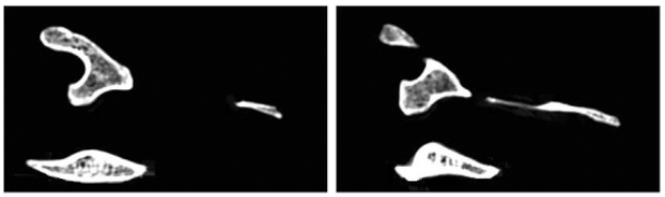


Figura 2.6: Imagens segmentadas [1].

A limiarização é uma técnica de processamento de imagens utilizada para fazer a separação entre o fundo da imagem, e os elementos a serem analisados – regiões de interesse, e regiões de não interesse [14]. Com essa técnica, transforma-se uma imagem em escala de cinza em uma imagem monocromática (preto e branco). Para tal, é necessário levar em consideração o limiar (*threshold*) ou também conhecido como ponto de corte.

Representando uma imagem digital como uma função $f(x, y)$, pixels com intensidade menor ou igual ao ponto de corte passam a ser pretos (0) e pixels com intensidade maior que o ponto de corte passam a ser brancos (255), a binarização é dada pela função $g(x, y)$ [14].

$$g(x,y) = \begin{cases} R_1, sef(x,y) \leq T \\ R_2, sef(x,y) > T \end{cases} \tag{2.1}$$

Considerando que T é o ponto de corte; R_1, 0 (preto); R_2, 255 (branco).

O objetivo dos algoritmos de limiarização é descobrir automaticamente o ponto ideal T de forma a capturar somente os elementos que façam parte do objeto [14]. Uma ferramenta importante utilizada para auxiliar nesse processo é o histograma da imagem. O histograma é um gráfico da distribuição de frequência das cores presentes na imagem, quanto mais bimodal for a distribuição, mais fácil é a localização do ponto de corte. Na Figura 2.7 são mostrados os resultados do processamento das imagens da Figura 2.6 após o processo de limiarização.

2.2.2 Detecção de bordas

Uma borda é definida por uma variação brusca, altas frequências, de nível de cinza [15], ou também pode ser definida como os limites entre um objeto e o fundo da

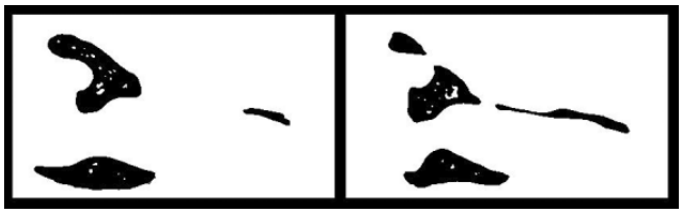


Figura 2.7: Imagens limiarizadas [1].

imagem. As identificações das bordas são importantes, pois nelas que ficaram os pontos limites do objeto de estudo. Na Figura 2.8, são mostrados os resultados do processamento das imagens da Figura 2.7 após o processo de detecção de bordas.

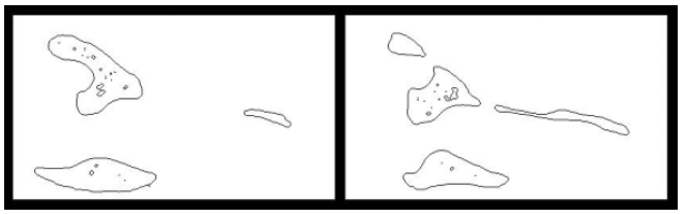


Figura 2.8: Bordas [1].

Sharifi, Fathy e Mahmoud [15] fizeram uma comparação entre diversos métodos de detecção de bordas. Os algoritmos que se saíram melhor foram classificados na seguinte ordem: *ISEF, Canny, MarrHildreth, Kirsch, Sobel, Lapla2* e *Lapla1*.

A captura de ponto de cada contorno é realizada percorrendo a borda do polígono fechado em estudo. Os pontos deverão ser definidos de forma a manter as principais características da imagem, sendo assim os pontos são distribuídos sempre onde há mudança de orientação do contorno [16] como mostrado na Figura 2.9. Souza, Centeno e Pedrini [1] destacam o uso do algoritmo *Chain Code* para realizar a escolha dos pontos.

2.3 Reconstrução

Nessa fase é feita a reconstrução de fato. O processo se divide em 3 etapas [16]: primeiro é feita a correspondência, conhecido também como definição das arestas, entre os pontos dos contornos nas fatias adjacentes; em seguida é tratado o problema das ramificações, quando existir; por fim, é gerada uma malha triangular conectando esses contornos adjacentes.

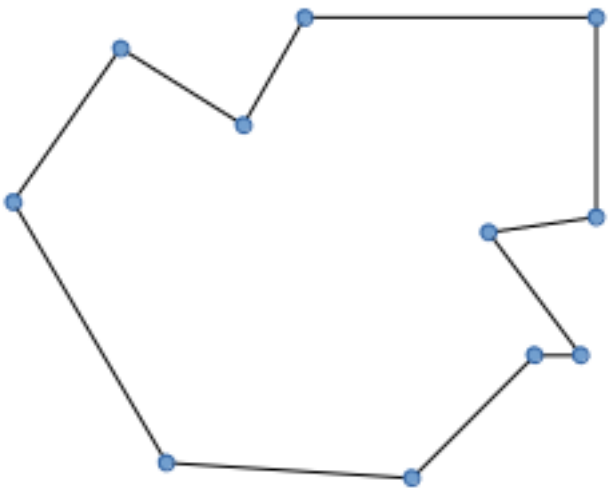


Figura 2.9: Exemplo de contorno com pontos distribuídos.

Na etapa de correspondência, é realizado o mapeamento entre os contornos de fatias adjacentes [16]. É nessa etapa que também é estabelecida a topologia da superfície, e as possíveis conexões entre os contornos das fatias. Necessário encontrar uma boa solução para o problema, pois existem inúmeras possíveis variações, ver exemplos na Figura 2.10.

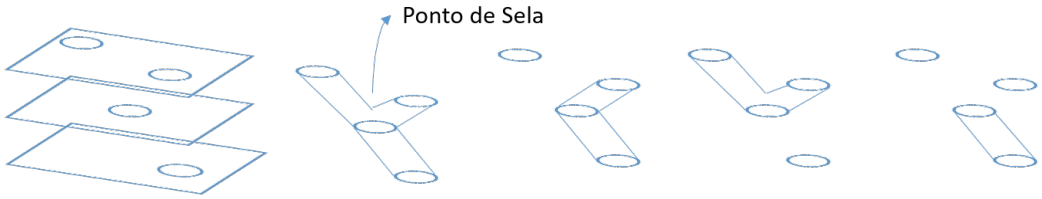


Figura 2.10: Pontos distribuídos em fatias e suas possíveis conexões.

Nas ramificações, quando houver um contorno em uma fatia e mais de um contorno correspondente na fatia adjacente [16], são geralmente considerados dois possíveis casos de tratamento desse problema como mostrados na Figura 2.10 a) quando há contornos correspondentes é necessário calcular uma bifurcação por meio de interpolação de um ponto de sela; b) quando não há contornos correspondentes na fatia adjacente é característico de buracos na superfície, ou a presença de outros objetos dentro do objeto de estudo.

Na geração de malha, os contornos são conectados por interpolação e a superfície entre eles são reconstruídas [16]. É definido quais vértices de um contorno se conectaram

aos contornos adjacentes correspondentes. Existem duas possíveis abordagens para gerar a malha:

- Considerar os pontos distribuídos no espaço sem nenhum tipo de conexão, uma nuvem de pontos [2] formando uma solução mais genérica, e tornando a solução computacional mais difícil de se obter;
- Considerar os pontos em planos paralelos, com ligações conhecidas previamente [16]. Essa abordagem ainda pode se dividir em resolução por métodos ótimos e por métodos heurísticos.
- Os métodos ótimos geram uma malha melhor, eles são baseados em critérios globais (maximização de volume, minimização de superfícies, etc.) e são mais caros de processar computacionalmente [1].
- Os métodos heurísticos são baseados em critérios locais, o que permite aproximações admissíveis ao ótimo, e são computacionalmente mais baratos.

2.4 Impressão 3D

Atualmente existem diversas técnicas de impressão 3D [5], mas de forma geral todas elas partem do mesmo princípio: fatiar o objeto 3D em camadas muito finas (da ordem de 0,1 mm a 0,5 mm dependendo do nível de precisão dos detalhes da impressão); e em seguida realizar a impressão de cada camada, uma em cima da outra, através do processo de deposição de materiais. Dessa forma a peça final é construída. O material utilizado pela impressora para realizar a impressão pode variar entre plásticos, metais, madeiras, vidros, argila, papel, dentre outros [6]. O material mais comum é o plástico.

A impressão em três dimensões de objetos é uma tecnologia relativamente nova, que no começo de seu desenvolvimento era restrita somente a grandes empresas devido ao seu alto custo, mas que agora está se popularizando e se tornando mais acessível devido a adaptações da técnica de impressão a projetos do tipo "faça você mesmo" (*Do It Yourself – DIY*), no qual basta procurar um projeto de construção na internet, comprar as peças necessárias e com um pouco de conhecimento de eletrônica e programação montar uma impressora em casa [6]. O projeto mais conhecido nessa área é o *RepRap*.

Entre as técnicas mais comuns de impressão se destaca a deposição de material fundido (*Fused Deposition Modeling – FDM*), essa técnica consiste em esquentar o material a ser impresso a ponto de ficar quase líquido e fazer a extrusão (fundir) desse material através de uma cabeça de impressão, formando um fino cordão que será depositado em uma base que seja aderente de forma a "grudar" completamente a primeira camada sem se movimentar durante todo o processo, e que seja fácil de "desgrudar" o objeto assim que terminar a impressão da peça. Após a impressão da primeira camada, a cabeça de impressão é elevada a altura da segunda camada e então realiza a impressão dessa camada. O processo se repete até que todas as camadas estejam impressas uma em cima da outra [5]. Essa é a técnica de impressão mais utilizada atualmente pois é a técnica implementada pelo projeto *RepRap*, que será descrito mais à frente.

2.4.1 Projeto RepRap

O projeto *RepRap* (*Replicating Rapid-prototyper*) iniciou a revolução das impressões 3D. O objetivo principal desse projeto é o de criar impressoras 3D que possam imprimir as partes necessárias para montar outras impressoras 3D, ou seja, a *RepRap* é uma máquina de prototipagem rápida auto replicável [5] feita em casa. Na Figura 2.11 um exemplo de impressora *RepRap*.

Os modelos de impressoras desenvolvidas pelo projeto podem imprimir em diversos tipos de materiais, sendo os mais comuns ABS (*Acrylonitrile Butadiene Styrene*) e PLA (*Polylactic Acid*), basicamente um tipo de plástico [5]. Sendo assim, atualmente o projeto chegou num estágio em que grande parte das peças mecânicas pode ser impressas por outras impressoras, e as peças restantes (parafusos, porcas, barras metálicas, entre outras) podem ser encontradas facilmente em lojas de materiais de construção, ou recicladas de impressoras antigas. Já as partes eletrônicas e motores podem ser encontradas na internet devido à grande popularização desse projeto.

2.4.2 Processo de impressão

O processo de impressão de objetos 3D é bem simples e pode ser resumido em 5 passos [5]:

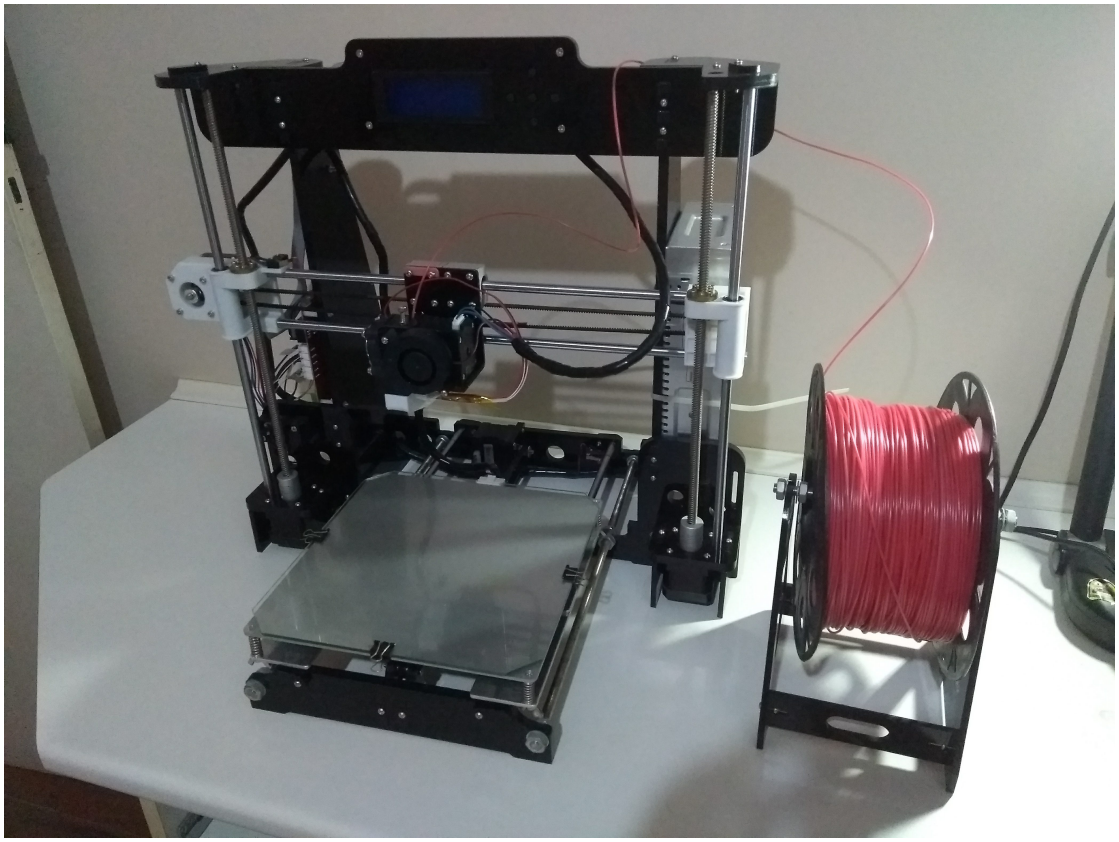


Figura 2.11: Impressora modelo Anet A8.

1. Passo 1. Obter objeto 3D - um objeto em três dimensões pode ser obtido de diversas formas, por exemplo:

    - O objeto pode ser desenhado através de diversos programas do tipo CAD que existem no mercado (*AutoCAD 3D, SolidWorks, Google Sketchup*, etc.);
    - O objeto pode ser obtido através de *downloads* em redes sociais de compartilhamento de modelos em 3D;
    - O objeto pode ser reconstruído através das diversas técnicas de reconstrução existentes;

2. Passo 2. O modelo deve ser convertido em um formato que a impressora reconheça (formato *STL*)

3. Passo 3. O modelo deve ser fatiado em camadas finas.

4. Passo 4. Cada camada obtida deve ter as coordenadas para sua impressão,

descritas na linguagem gcode que controla o movimento dos motores da impressora. Exemplo, caminhar a uma velocidade de Xm/s das coordenadas A até B com a cabeça de impressão ligada. Esse comando faz com que seja impresso uma linha entre o ponto A e B

5. Passo 5. A impressora interpreta o gcode, e imprime o objeto.

2.5 Trabalhos na área

Souza, Centeno e Pedrini [1] descrevem um sistema que integra a reconstrução tridimensional a partir de imagens tomográficas para criação de modelos médicos que são impressos em uma máquina de prototipagem rápida (impressora 3D) em quatro fases: Na fase de pré-processamento é feita uma segmentação das imagens com a redução do número de cores e recorte da imagem, depois a limiarização com algoritmo de *Otsu*, detecção de bordas com o algoritmo de *Kirsch*, afinamento de bordas com o algoritmo de *Stentiford*, rotulação de contornos, e a seleção dos pontos com o algoritmo *Chain Code*. Na fase 2, é feita a reconstrução das superfícies através do mapeamento dos contornos para a identificação de quais contornos devem ser conectados através de suas sobreposições, e assim realizar a união das seções transversais. A fase 3 é a geração de um arquivo que seja reconhecido pela impressora 3D, e a última fase é a fabricação dos modelos.

Azevedo, Tavares, e Vaz [4] estudam formas de fazer a reconstrução volumétrica de objetos a partir de imagens 2D de forma automática, rápida e precisa. As imagens são adquiridas por uma câmera calibrada pelo método de calibração *Zhang*. A reconstrução é feita em duas partes, a primeira é a de pré-processamento onde as imagens são segmentadas, a limiarização é feita por *threshold* e são utilizadas operações morfológicas. Na segunda parte foram utilizados três métodos de reconstrução volumétrica para comparação. Os dois primeiros são baseados em silhuetas, e o terceiro é baseado em silhuetas e critérios de foto-consistência.

Centeno [7] analisa o *FCC* (craqueamento catalítico fluído), um processo utilizado no refinamento de petróleo, conhecido por produzir gasolina. A contribuição do artigo está em tentar maximizar a produção de gasolina, diesel e olefinas leves, além de reduzir a geração de efluentes. O autor utiliza tomógrafos industriais para adquirir

imagens das seções transversais de um *riser* (reator de leito fluidizado de transporte em fase diluída). Durante o pré-processamento das imagens adquiridas é utilizado um filtro para eliminar ruídos na imagem quando necessário, e em seguida utiliza-se o filtro da mediana para observar a variação da densidade dos fluidos dentro do *riser*, a limiarização é feita com o algoritmo de *Otsu*, detecção de contornos com o algoritmo de *Stentiford*, e finaliza com a rotulação dos contornos. A reconstrução é feita pela construção de faces triangulares entre dois contornos situados em fatias diferentes.

Fernandes [8] desenvolve um scanner 3D de luz estruturada econômico e eficaz. O autor utiliza a biblioteca *OpenCV* para reconhecer e formar as nuvens de pontos. As nuvens passam por um pré-processamento para remoção de ruídos, e são reconstruídas pelo método de superfícies por um algoritmo de triangulação.

Rebouças [9] utiliza imagens de tomografia computadorizada do tórax para fazer a reconstrução volumétrica de pulmões para ajudar o diagnóstico de doenças que atingem os pulmões como DPCO e fibrose. A técnica de reconstrução utilizada é o método de Crescimento de Regiões 3D (CR 3D), onde inicia-se com um voxel semente dentro da região de interesse, e nas iterações seguintes são adicionados voxels vizinhos respeitando a regra de pertencer a região de interesse. O desenvolvimento do algoritmo foi feito utilizando a biblioteca *OpenGL*.

Moraes, Dias e Melani [10] utilizam a fotogrametria – método para extrair de fotos as dimensões, posições e métricas de objetos – para realizar a reconstrução de superfície aplicada a medicina forense. Através do método de fotogrametria são geradas nuvens de pontos que são trianguladas pelo algoritmo de *Poission*, um algoritmo que calcula uma função implícita através das normais dos pontos da nuvem.

Amorim [11] descreve o funcionamento do software livre *InVesalius* para tratamento de imagens médicas. As imagens provenientes de tomografia computadorizada e ressonância magnética são obtidas no formato *DICOM*, que é baseado em *voxels*, e pelo algoritmo de *Marching Cubes* é gerada uma superfície poligonal constituída principalmente por triângulos.

Capítulo 3

Reconstrução

O processo de reconstrução inicia-se com a verificação de convexidade dos contornos. O algoritmo de triangulação escolhido para implementação neste trabalho aceita somente contornos convexos como dados de entrada, caso o contorno não seja convexo é necessário construir o seu fecho convexo para então realizar a triangulação [22].

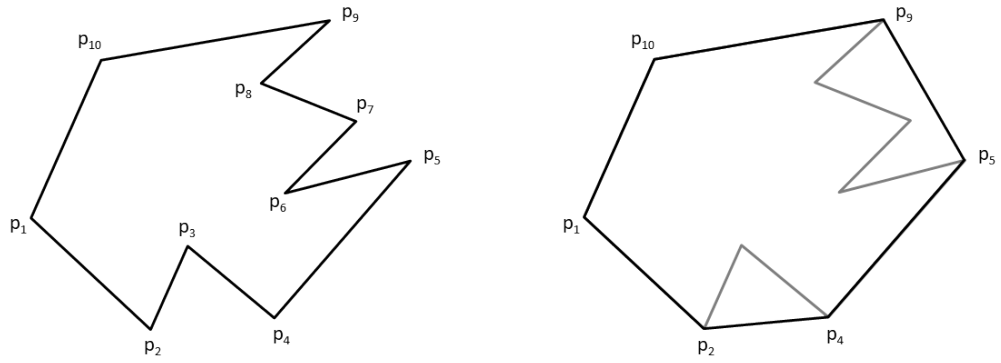
Cada imagem da sequência de imagens do conjunto recebe o nome de fatia. Cada fatia pode possuir um ou mais contornos, mas para simplificação neste trabalho é limitado somente para um contorno em cada fatia. Um contorno é composto por uma sequência de pontos do tipo (x, y, z) onde o ponto (x, y) representa a posição na fatia em duas dimensões e o z a altura da fatia na pilha de fatias, definido externamente como um constante de separação entre as fatias adjacentes. Os contornos são fechados, logo o último ponto tem como ponto vizinho o primeiro ponto, e vice-versa.

Logo após a definição de fechos convexos e tratamentos dos elementos de concavidades, ou transformação, de todos os contornos, começa o processo de triangulação [16]. Basicamente o algoritmo conecta o contorno de uma fatia como o contorno da fatia adjacente através da criação de arestas, que são segmentos de reta unindo dois pontos. Este processamento gera como dado de saída um conjunto de triângulos que serão então utilizados para realizar duas tarefas: a primeira é renderizar o sólido reconstruído na tela para o usuário analisar visualmente o resultado final, e a segunda função deste conjunto de triângulos é gerar um arquivo do formato *".stl"* para ser impresso na impressora 3D.

3.1 Fecho Convexo

O fecho convexo de um contorno pode ser definido, de forma informal, da seguinte forma [24]: supondo que um plano de duas dimensões seja representado por uma tábua de madeira onde pregos são fixados na posição exata onde ficam os pontos do contorno. O fecho convexo deste contorno será definido pelo formato adquirido por um elástico de borracha que foi esticado em volta desses pregos. Se o formato do elástico for exatamente igual ao formato do contorno, ele é então classificado como um contorno convexo. O caso contrário, no qual o formato do contorno não é igual e existe algum prego mais interno que não seja tocado pelo elástico de borracha, esse contorno é classificado com côncavo ou não-convexo. A Figura 3.1 ilustra um contorno não-convexo 3.1(a) e seu fecho convexo 3.1(b), os pontos p_3, p_6, p_7, p_8 são os vértices que definem que o contorno não seja convexo.

O algoritmo de triangulação utilizado neste trabalho tem como exigência de bom desempenho a utilização de somente contornos convexos, desta forma é necessário calcular o fecho convexo de todos os contornos classificados como côncavos. O objetivo da transformação de um contorno côncavo para um contorno convexo é não perder nenhuma informação do contorno original, desta forma todos os pontos que não fazem parte do fecho convexo devem ser projetados em uma posição equivalente no fecho.



(a) (b)

Figura 3.1: Contorno poligonal: (a) fecho; (b) fecho convexo.

Seguindo a definição formal deste problema clássico da geometria computacional [20] temos que um contorno fechado (C) é convexo se todos os pontos que definem este contorno também definem o seu fecho convexo $E(C)$. Ou seja, somente se $E(C) = (C)$. Caso contrário, o fecho (C) é não-convexo se ele difere de seu fecho convexo

$E(C)$.

Seja um contorno fechado e ordenado (C) definido por uma sequência de pontos ordenados $p_i, 1 \leq i \leq M$ de forma que $(C)^0 = (C)$, e seu fecho convexo é formado por uma subsequencia de pontos do contorno $(C)^0 - (E)^0 = p_k/k \in K$ onde K é um subconjunto ordenado de inteiros em $[1, M]$. Portanto se $(E)^0 \neq (C)^0$, ao menos existem dois pontos $p_{(j_1)}$ e $p_{(i_1)}$ sucessores em $(E)^0$ mas não em $(C)^0$, e o conjunto p_k não é vazio. Sendo assim o contorno $(C)^0$ não é convexo, e existe pelo menos uma concavidade de primeira ordem $(C)^1_{(i_1,j_1)}$. Existem vários métodos de gerar um fecho convexo a partir de um conjunto de pontos, neste caso seguimos o algoritmo de Graham tratado em detalhe, entre outros algoritmos, no livro de Figueiredo e Carvalho [20].

3.1.1 Algoritmo de Graham

O algoritmo de Graham foi o primeiro artigo publicado na área de computação geométrica, ele foi publicado no ano de 1972. Na época Graham trabalhava no Bell Laboratories, e precisava calcular o fecho convexo de um problema com aproximadamente $n = 10.000$ pontos, o melhor algoritmo da época tinha uma complexidade de $O(n^2)$, o que dificultava resolver o problema em tempo hábil. Ele teve então a ideia de desenvolver este algoritmo que calcula o fecho com uma complexidade $O(n \log n)$, bem mais eficiente e apropriado para resolver o seu grande problema [24].

O algoritmo tem como estrutura de dados uma pilha, e a principal estratégia para o algoritmo ter sucesso se dá por uma perfeita escolha do primeiro ponto a entrar nessa pilha. É essencial que a escolha seja feita de forma que o ponto esteja presente no fecho convexo. Para tal, é necessário garantir que o primeiro ponto seja o ponto mais "baixo" do conjunto de pontos no plano, ou seja, o ponto $P(X, Y)$ escolhido é o com a menor coordenada Y e menor das abscissas X.

Os pontos seguintes devem ser reordenados de acordo com o ângulo em relação ao primeiro ponto. Após o conjunto ordenado, cria-se uma função para remoção de pontos com ângulos repetidos, mantendo no conjunto o ponto que seja mais distante do primeiro ponto. Utilizando o mesmo exemplo das seções anteriores, podemos ver na Figura 3.2 como funciona a reordenação dos pontos do contorno.

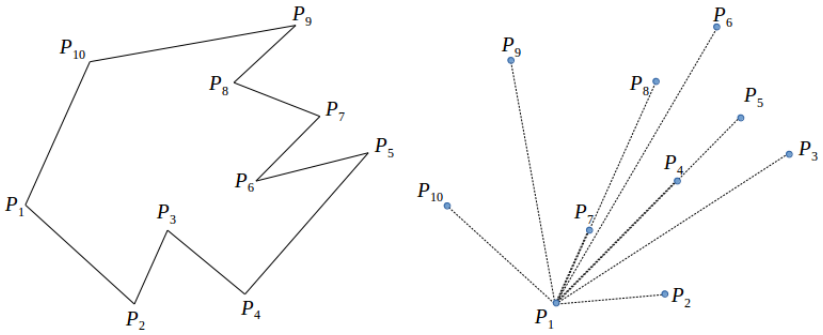


Figura 3.2: Algoritmo de Graham

Na Figura 3.2 podemos ver os pontos ordenados segundo o ângulo em relação ao ponto p_2, que apartir de agora se torna o ponto p_1. Nesta Figura é possivel identificar dois pares de pontos com o mesmo ângulo em relação ao ponto inicial, são eles: p_4 e p_5; p_7 e p_8. Sendo assim os pontos mais próximos ao primeiro ponto (p_4 e p_7 respectivamente) são removidos.

A próxima etapa consiste em verificar a convexidade de cada ponto individualmente através de uma função de verificação de convexidade. Os pontos são analisados na ordem estabelecida na reordenação dos pontos. A função analisará os pontos da seguinte maneira: tendo dois pontos como referência (ponto um e ponto dois) é traçada uma reta entre eles. Verifica-se o terceiro ponto (ponto três), o ponto é analisado de forma a descobrir se ele está a esquerda ou a direita desta reta. Como estamos percorrendo o contorno no sentido anti-horário, todos os pontos à esquerda fazem parte do fecho convexo. Caso o ponto três esteja à esquerda, este ponto faz parte do fecho e é acrescentado na pilha. Caso esteja à direita o ponto dois é removido da pilha, o ponto um se torna o ponto dois, e o ponto anterior ao antigo ponto um se torna o ponto um. Verifica-se novamente o ponto três, se estiver a esquerda ele entra na pilha, se estiver a direita ele é ignorado, não entra na pilha e a análise passa para o ponto seguinte. O algoritmo continua assim até que todos os pontos tenham sido analisados [24].

Devido aos pontos de cada fatia, especificamente neste trabalho, estarem organizados seguindo uma direção anti-horária, para evitar a reorganização desnecessária dos pontos, o algoritmo de fecho convexo não seguiu estritamente o algoritmo de Graham, e sim um algoritmo inspirado nele.

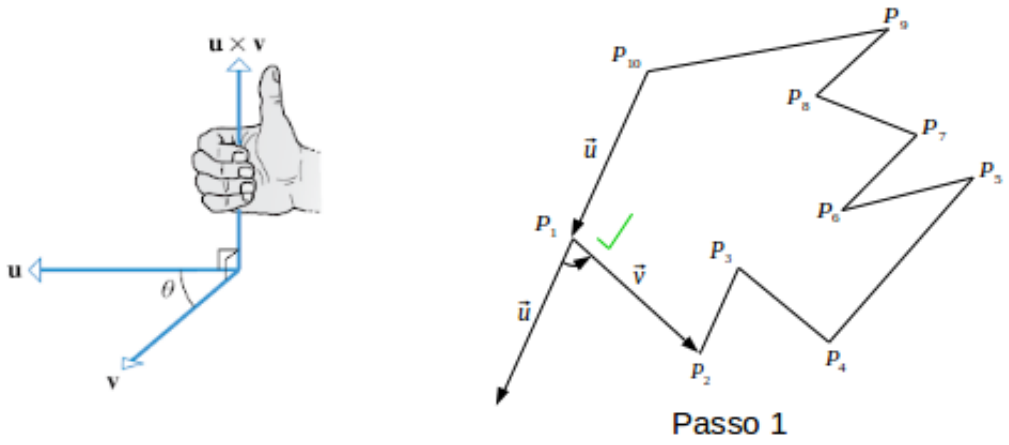


Figura 3.3: Regra da mão direita [19] e análise do primeiro ponto do contorno

3.1.2 Algoritmo implementado neste trabalho

O algoritmo implementado neste trabalho tem como dado de entrada uma lista encadeada contendo um contorno fechado. Iniciando do primeiro ponto, ele deverá percorrer esta lista verificando a convexidade de cada ponto através de uma função local de verificação de convexidade. Se o ponto for côncavo ele é removido da lista e a análise volta ao último ponto que passou no teste de convexidade. Quando o algoritmo terminar os pontos que ficarem na lista formaram o fecho convexo.

A função de verificação de convexidade tem como base o cálculo do produto vetorial entre dois vetores [19]. Em posse de três pontos, o ponto a ser analisado, o seu ponto anterior e o seu ponto seguinte, cria-se então dois vetores: o primeiro com a subtração entre o ponto atual e o ponto anterior ($\overrightarrow{u}$), e o segundo com a subtração do ponto seguinte com o ponto atual ($\overrightarrow{v}$). Através do produto vetorial entre eles cria-se um vetor ortogonal $\overrightarrow{u \times v}$. Dependendo de sua orientação identificamos a convexidade do contorno. Utilizando a regra da mão direita, se o vetor ortogonal apontar grosseiramente para cima ele tem uma orientação positiva e significa que o ponto analisado é possivelmente convexo, caso contrário, se apontar para baixo ele terá uma orientação negativa, tornando o ponto analisado côncavo.

Como pode ser visto na Figura 3.3, utilizando a regra da mão direita, quando os dedos estão em posição de concha e vão se fechando no sentido do vetor $\overrightarrow{u}$ ao $\overrightarrow{v}$ o polegar aponta para cima. Portanto analisando o primeiro ponto do contorno dos exemplos anteriores chegamos a conclusão de que ele é convexo.

Cada ponto da lista precisa ter uma variável de controle associada para garantir que o algoritmo percorra cada ponto exatamente por duas vezes. Na primeira execução será criado o fecho convexo. Já a segunda execução serve para eliminar qualquer falha que por algum motivo tenha passado no processo. Um bom exemplo de falha que pode ocorrer é com o primeiro ponto da lista, se ele estiver dentro de uma concavidade não vai ser possível identificar somente pela primeira análise, pois a função de verificação de convexidade realiza os seus testes somente com três pontos, um ponto anterior e um ponto seguinte ao ponto analisado, não possuindo assim uma visão global da lista. Neste caso em específico, se este ponto anterior também fizer parte da concavidade vai parecer para a função de verificação que os dois pontos fazem parte do fecho convexo, na segunda execução esta concavidade será identificada e logo eliminada.

Prosseguindo a construção iniciada na Figura 3.3, seguindo os treze passos no exemplo a seguir. Na primeira parte desse processo, correspondendo aos passos 1 a 7, é ilustrada Figura 3.4.

Após verificado o ponto p_1 no passo 1 da Figura 3.3, passamos ao p_2 no passo 2 da Figura 3.4. Cria-se o vetor $\overrightarrow{u}$ entre $\overrightarrow{p_2p_1}$, e o vetor $\overrightarrow{v}$ entre $\overrightarrow{p_3p_2}$. Pela regra da mão direita, a concha formada pela mão irá se fechar no sentido $\overrightarrow{u \times v}$, apontando o polegar para cima. Logo o ponto p_2 é convexo.

No passo 3 (Figura 3.4), analisamos o ponto p_3. Temos o vetor $\overrightarrow{u}$ entre $\overrightarrow{p_3p_2}$ e o vetor $\overrightarrow{v}$ entre $\overrightarrow{p_4p_3}$. Neste caso a regra da mão direita ao fechar a concha da mão no sentido do produto $\overrightarrow{u \times v}$ fica com o polegar apontando para baixo. Logo o ponto p_3 é côncavo, desta forma ele é eliminado da lista. Com a eliminação do ponto p_3 no passo 3, no passo 4 (Figura 3.4) o ponto analisado se torna o último ponto convexo registrado, no caso o ponto p_2. O vetor $\overrightarrow{u}$ será entre $\overrightarrow{p_2p_1}$, e o vetor $\overrightarrow{v}$ será entre $\overrightarrow{p_4p_2}$. A regra da mão direita mostra um vetor ortogonal apontando para cima, portanto o p_2 continua sendo convexo.

No passo 5, tal como ilustra a Figura 3.4, o vetor $\overrightarrow{u}$ será entre $\overrightarrow{p_4p_2}$, e o vetor $\overrightarrow{v}$ entre $\overrightarrow{p_5p_4}$. Regra da mão direita apontando para cima, vetor ortogonal positivo, igual a ponto convexo. No passo 6, temos o mesmo caso. Vetor $\overrightarrow{u}$ entre $\overrightarrow{p_5p_4}$, vetor $\overrightarrow{v}$ entre $\overrightarrow{p_6p_5}$. Regra da mão direita pra cima, ponto convexo. No passo 7, o ponto p_6 em análise está dentro de uma concavidade. O vetor $\overrightarrow{u}$ será $\overrightarrow{p_6p_5}$ e o vetor $\overrightarrow{v}$ será $\overrightarrow{p_7p_6}$, pela regra da mão direita o produto $\overrightarrow{u \times v}$ vai gerar um vetor apontando pra baixo, logo o ponto é côncavo e deve ser removido da lista.

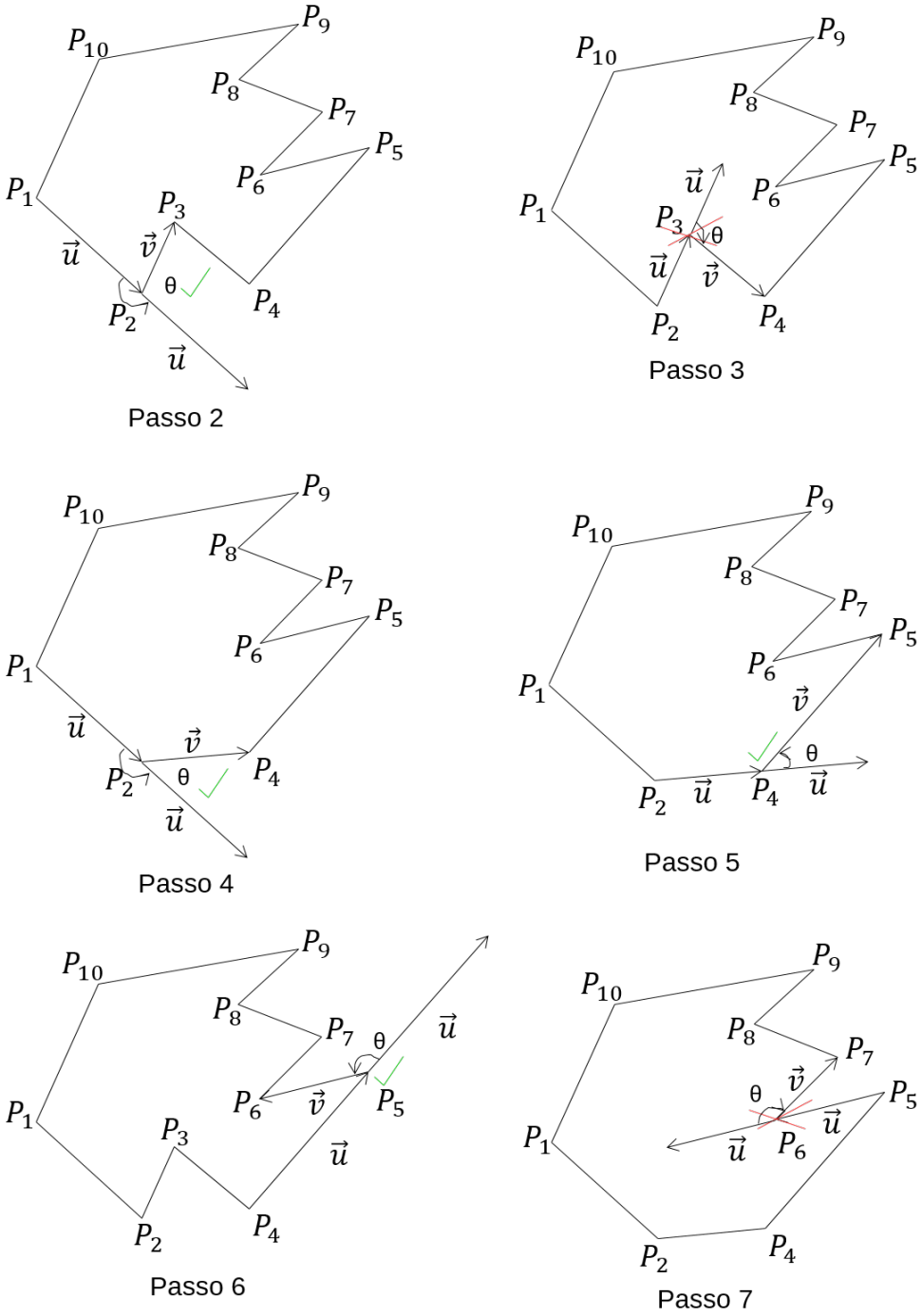


Figura 3.4: Primeira parte do processamento do fecho convexo.

Os procedimentos dos passos 8 a 13 são ilustrados pela Figura 3.5. Assim, no passo 8, após a eliminação do ponto p_6, a análise volta ao ponto p_5 para reconfirmar a sua convexidade. O vetor $\overrightarrow{u}$ será $\overrightarrow{p_5p_4}$ e o vetor $\overrightarrow{v}$ será $\overrightarrow{p_7p_5}$. A regra da mão direita mostrará um vetor ortogonal positivo, logo p_5 é convexo. No passo 9 verificará o ponto p_7. O vetor $\overrightarrow{u}$ será $\overrightarrow{p_7p_5}$ e o vetor $\overrightarrow{v}$ será $\overrightarrow{p_8p_7}$. Apesar de estar dentro de uma concavidade, analisando somente os pontos que a função de verificação tem acesso, o produto vetorial tem um vetor positivo e o ponto é convexo.

No passo 10 o algoritmo começa a identificar a concavidade. O vetor $\overrightarrow{u}$ será $\overrightarrow{p_8p_7}$ e o vetor $\overrightarrow{v}$ será $\overrightarrow{p_9p_8}$. O vetor ortogonal $\overrightarrow{u \times v}$ é voltado para baixo, e o ponto é côncavo. O ponto será eliminado da lista, e a análise volta ao ponto anterior. No passo 11 a análise volta ao ponto p_7 para confirmar a sua convexidade. O vetor $\overrightarrow{u}$ será $\overrightarrow{p_7p_5}$ e o vetor $\overrightarrow{v}$ será $\overrightarrow{p_9p_7}$. O produto $\overrightarrow{u \times v}$ tem um vetor com orientação negativa, logo o ponto p_7 é côncavo e deve ser eliminado da lista. No passo 12 a análise volta ao ponto convexo anterior p_5 para confirmação de sua convexidade. O vetor $\overrightarrow{u}$ é $\overrightarrow{p_5p_4}$ e o vetor $\overrightarrow{v}$ é $\overrightarrow{p_9p_5}$. O produto $\overrightarrow{u \times v}$ tem uma orientação positiva, e o ponto p_5 é convexo.

O Passo 13 analisará o ponto seguinte p_9, tendo como vetor $\overrightarrow{u}$ $\overrightarrow{p_9p_5}$ e vetor $\overrightarrow{v}$ $\overrightarrow{p_{10}p_9}$. O produto $\overrightarrow{u \times v}$ tem um vetor com orientação positiva, e o ponto é convexo. Para finalizar a primeira execução do algoritmo, analisa-se o ponto p_{10}. Este ponto tem o vetor $\overrightarrow{u}$ $\overrightarrow{p_{10}p_9}$ e o vetor $\overrightarrow{v}$ $\overrightarrow{p_1p_{10}}$. O produto vetorial $\overrightarrow{u \times v}$ gera um vetor positivo, sendo assim o ponto é convexo.

3.2 Concavidades

Concavidades são os segmentos de contornos formados pelos conjuntos ordenados de pontos $\{p_k/k = i_1 j_1\}$ (o subscrito k decresce de j_1 a i_1 para manter a orientação do contorno) que não se encontra no fecho convexo. Cada concavidade pode ser decomposta em outras concavidades menores até que seja encontrada a concavidade elementar. Uma concavidade elementar pode ser definida de duas formas:

1. Ela pode ser uma concavidade que forma um fecho convexo,
2. Ela pode ser uma concavidade que não pode ser reduzida pela pouca quantidade de pontos restantes no conjunto.

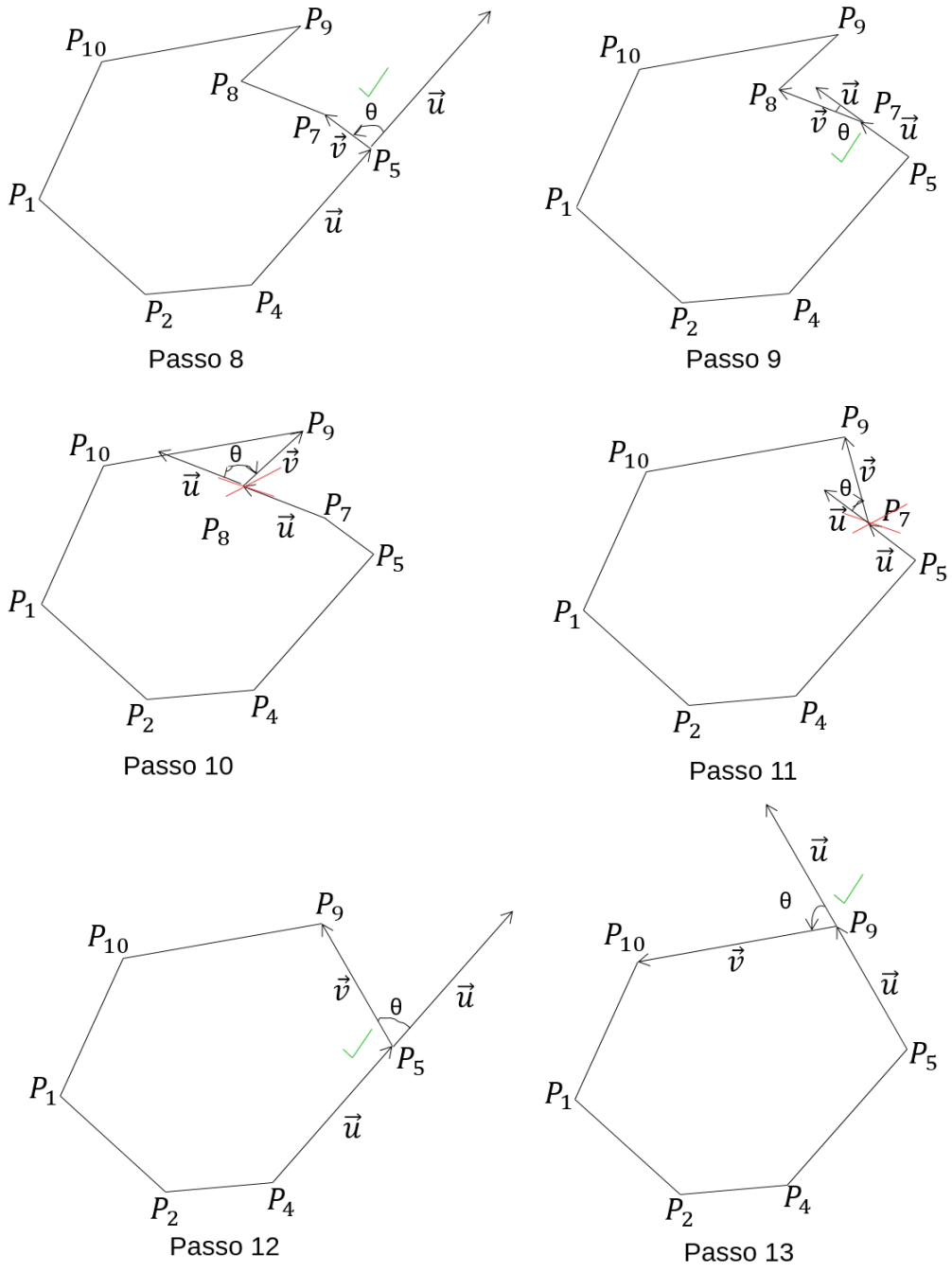


Figura 3.5: Segunda parte do processamento do fecho convexo.

A quantidade mínima de pontos em um conjunto é de três pontos, dois pontos que fazem parte do fecho convexo e formam a aresta de projeção e um ponto côncavo intermediário. A única forma geométrica capaz de ser representada por essa quantidade de pontos é um triângulo, e um triângulo é sempre convexo.

Uma concavidade é identificada através do indice dos seus subscritos. É necessário subtrair o indice do ponto atual, com o indice do ponto anterior. Se o valor encontrado for maior do que 1 (um) existe ali uma concavidade.

Seja a Figura 3.1(a) um contorno fechado $(C)^0$, ou um fecho, tal que p_i para $1 \leq i \leq 10$. O seu fecho convexo é então definido por $(E)^0 = (p_1, p_2, p_4, p_5, p_9, p_{10})$, ilustrado pela Figura 3.1(b). Nesse exemplo podemos identificar duas concavidades de primeira ordem: $(C)^1{}_{(4,2)} = (p_4, p_3, p_2)$ e $(C)^1{}_{(9,5)} = (p_9, p_8, p_7, p_6, p_5)$. Cada concavidade de primeira ordem é definido um novo contorno fechado, e por tanto é necessário definir seu fecho convexo. Esse novo contorno passará então pelo processo de identificação de concavidades. Esse processo deve continuar até que se chegue a concavidade elementar. Na Figura 3.6 ilustra duas concavidades de primeira ordem a partir do fecho convexo do exemplo Figura 3.1(b).

Analisando cada concavidade de primeira ordem, descobre-se que a concavidade $(C)^1{}_{(4,2)}$ forma um contorno convexo, pois seu fecho convexo $(E)^1{}_{(4,2)}$ satisfaz $(E)^1{}_{(4,2)} = (C)^1{}_{(4,2)}$ e é, portanto, uma concavidade elementar. Já a concavidade $(C)^1{}_{(9,5)}$ possui o fecho convexo $(E)^1{}_{(4,2)}$, que é diferente de seu contorno. Essa concavidade pode ser decomposta em uma concavidade menor de segunda ordem: $(C)^2{}_{(9,5)(6,8)} = (p_6, p_7, p_8)$, tal como ilustra a Figura 3.7, como uma árvore hierárquica de decomposição e projeções. Dando sequência a decomposição, analisa-se a concavidade de segunda ordem. Ela é convexa, sendo assim é também uma concavidade elementar.

A decomposição das concavidades forma uma estrutura de árvore com toda a hierarquia de concavidades do contorno, de forma que a raiz da árvore é o contorno $(C)^0$, o primeiro nível representa as concavidades de primeira ordem $(C)^1{}_{(i_1,j_1)}$, o nível n representa as concavidades de ene-ésima ordem $(C)^n{}_{(i_1,j_1)\ldots(i_n,j_n)}$, e as folhas são concavidades elementares. Concavidade de primeira ordem sempre é em relação ao seu nó pai da árvore hierárquica.

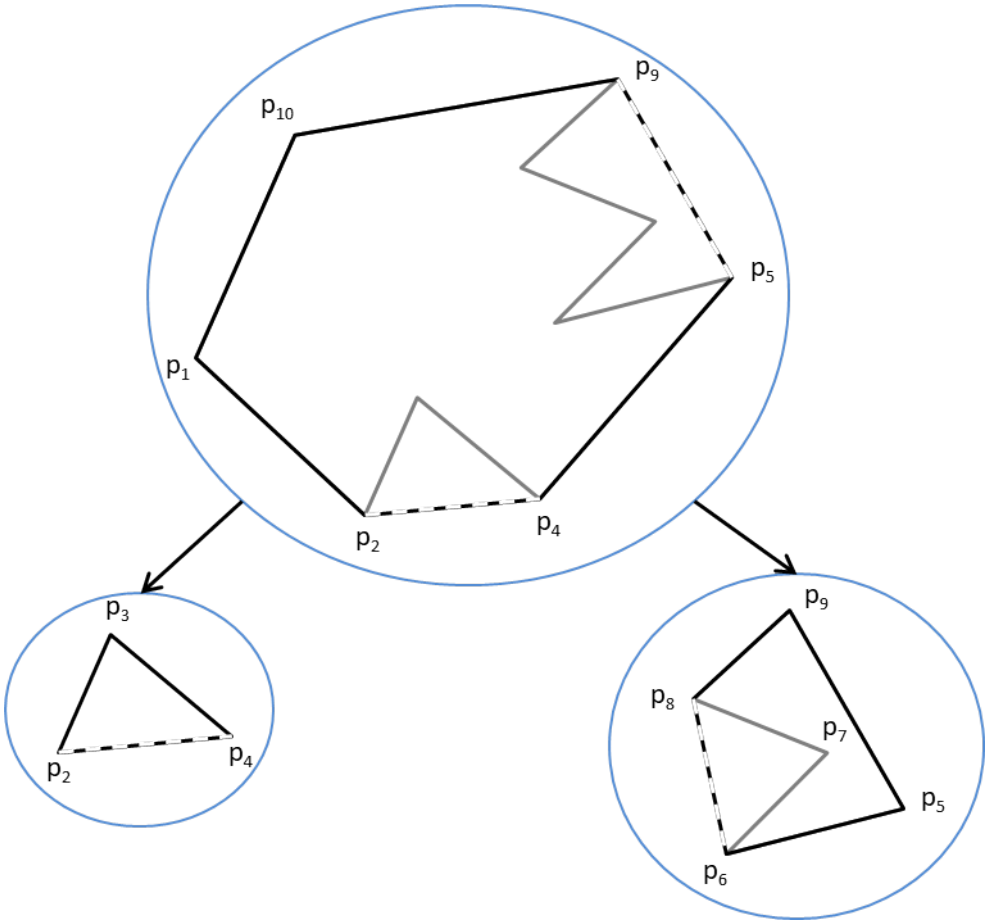


Figura 3.6: Concavidades de primeira ordem.

3.3 Transformação do contorno

Após a construção do fecho convexo temos dois contornos: o original e o fecho convexo. No escopo deste trabalho, ainda não podemos considerar estes contornos como sendo equivalentes. Consideramos o fecho convexo equivalente ao seu contorno original somente quando ele possui a mesma quantidade de pontos. Desta forma é necessário realizar uma transformação do contorno original para seu respectivo fecho convexo. Essa transformação consiste em projetar os pontos do conjunto $(C)^0 - (E)^0$ no fecho, ou seja, todos os pontos do contorno que não estão no fecho convexo devem ser projetados nele.

Para este propósito, utiliza-se a árvore hierárquica associada a $(C)^0$. Ela deverá, aos poucos, ser reduzida a um simples nó que representará o fecho convexo criado com todos os pontos projetados já incluídos. O processo de eliminação dos nós se

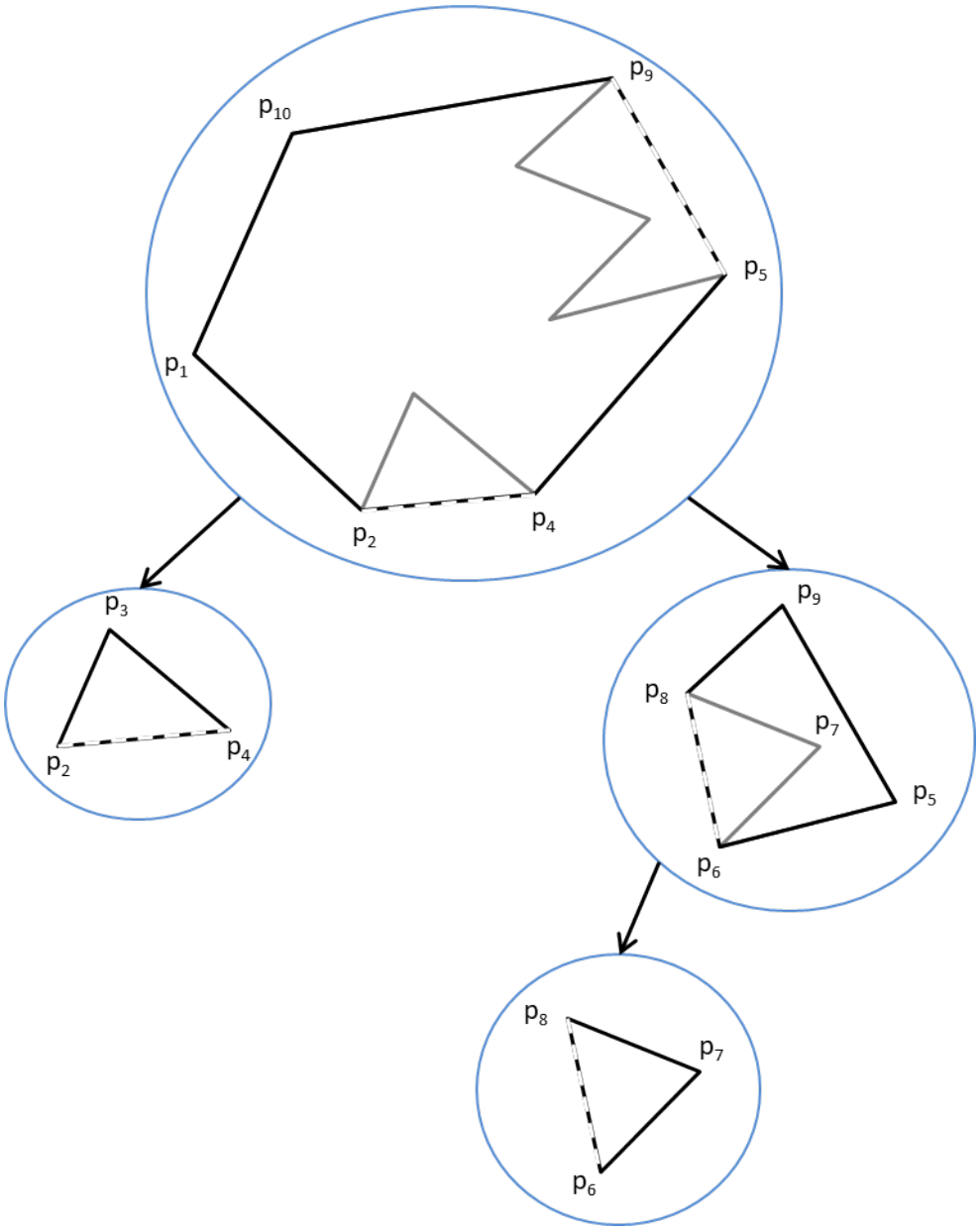


Figura 3.7: Árvore hierárquica.

dá de forma ascendente iniciando pelas folhas, e eliminando sucessivamente todos os nós terminais correspondentes as concavidades elementares. Um de cada vez, os nós terminais são eliminados da árvore e projetados na concavidade elementar correspondente ao fecho convexo do contorno definido por seu nó pai. A transformação termina quando todas as ramificações da árvore tenham sido eliminadas, dando origem ao contorno transformado (C').

Formalmente a transformação é implementada por recursividade e ocorre da seguinte forma: seja uma concavidade elementar de enésima ordem $(C)^n{}_{(i_1,j_1)\ldots(i_n,j_n)}$. Cada ponto p_i em $i \in (i_n, j_n)$ deve ser projetado na aresta de projeção $\overrightarrow{p_{in}p_{jn}}$ através da transformação $T(p_i) = p'_i$, de forma que $T := p_i + R_i(p_{jn} - p_{in})$ onde R_i é a fração do comprimento da origem da concavidade ao ponto de análise em relação ao comprimento das arestas de origem ao final da concavidade, tal como mostra a expressão R_i.

$$R_i = \frac{\sum_{k=i_n}^{i-1} d(p_k, p_{k+1})}{\sum_{k=i_n}^{j_n-1} d(p_k, p_{k+1})} \tag{3.1}$$

R_i tem como objetivo nesta função exercer um papel de fator de ponderação normalizado, que vai levar em conta no seu calculo a distância sobre o contorno entre p_i e p_{in} (respectivamente p_{jn}), assim mantendo uma distância relativa entre os pontos na concavidade. Quando o contorno C é processado, seu contorno convexo (C') obtido tem o mesmo número de pontos de forma que cada ponto p_i de (C) é associado com um único ponto $p'_i = T(p_i)$ em (C').

Seguindo o exemplo apresentado na seção anterior, temos na Figura 3.8 a seguir a primeira iteração da transformação, onde é analisada a concavidade elementar de segunda ordem $(C)^2{}_{(9,5)(6,8)}$, folha da árvore hierárquica.

Os pontos das folhas que não estão presentes no fecho convexo de seu nó pai ($(E)^1{}_{(9,5)}$) são projetadas na aresta de projeção $\overrightarrow{p_{in}p_{jn}}$ correspondente ($\overrightarrow{p_6p_8}$), e a folha é excluída da árvore. O nó pai agora se torna folha da árvore.

Na segunda iteração as novas folhas são as concavidades de primeira ordem $(C)^1{}_{(4,2)}$ e $(C)^1{}_{(9,5)}$, acrescidas das projeções da iteração anterior. Estas concavidades devem

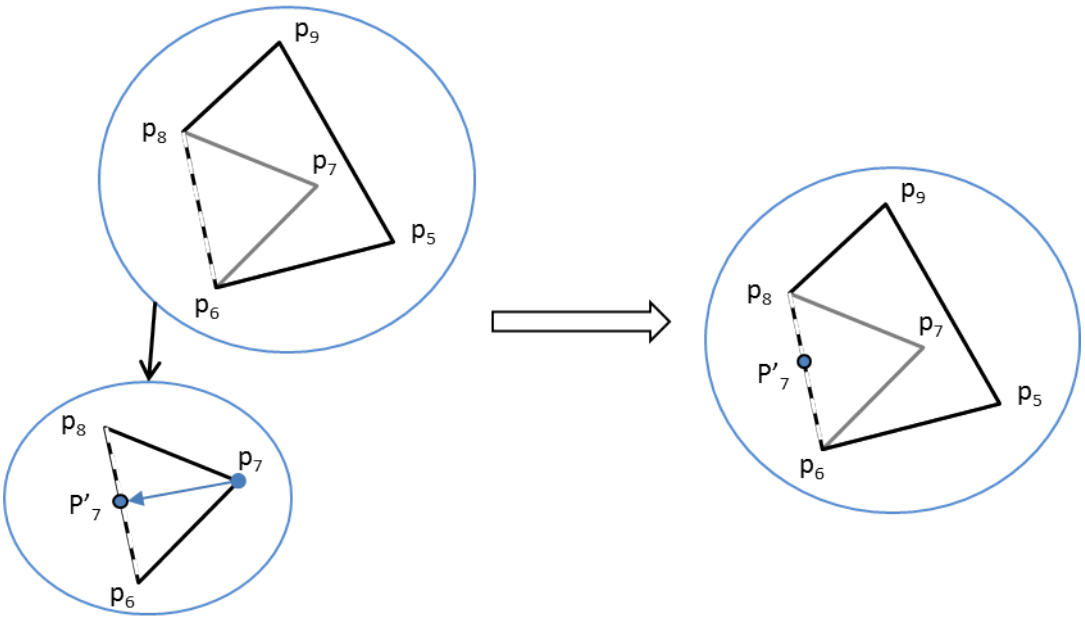


Figura 3.8: Transformação da concavidade elementar de segunda ordem.

ser projetadas na aresta de projeção $\overrightarrow{p_{in}p_{jn}}$ correspondente em seu nó pai $(C)^0$ ($\overrightarrow{p_4p_2}$ e $\overrightarrow{p_9p_5}$, respectivamente). Na Figura 3.9 é mostrada uma representação desta iteração.

Ao fim das projeções temos o novo contorno (C') que é um contorno fechado, convexo, e que possui a mesma quantidade de pontos que o contorno $(C)^0$. Na Figura 3.10 temos a comparação entre os contornos.

Na etapa de triangulação o algoritmo utilizado levará em conta nos seus calculos os pontos projetados, mas a real conexão será feita aos pontos originais de forma que não se altere o formato real do objeto a ser reconstruido. A projeção é necessária para que não ocorram triangulações erradas entre pontos internos ao objeto que pertençam a lados opostos do mesmo, desta forma os pontos sempre estarão projetados em uma região bem próxima a real.

3.4 Triangulação

Sejam $(C)^{k-i}$ e $(C)^k$ dois contornos fechados e não-convexos nos níveis $k-1$ e k respectivamente. Os seus contornos convexos associados, obtidos na transformação anterior, são $(C')^{k-1}$ e $(C')^k$.

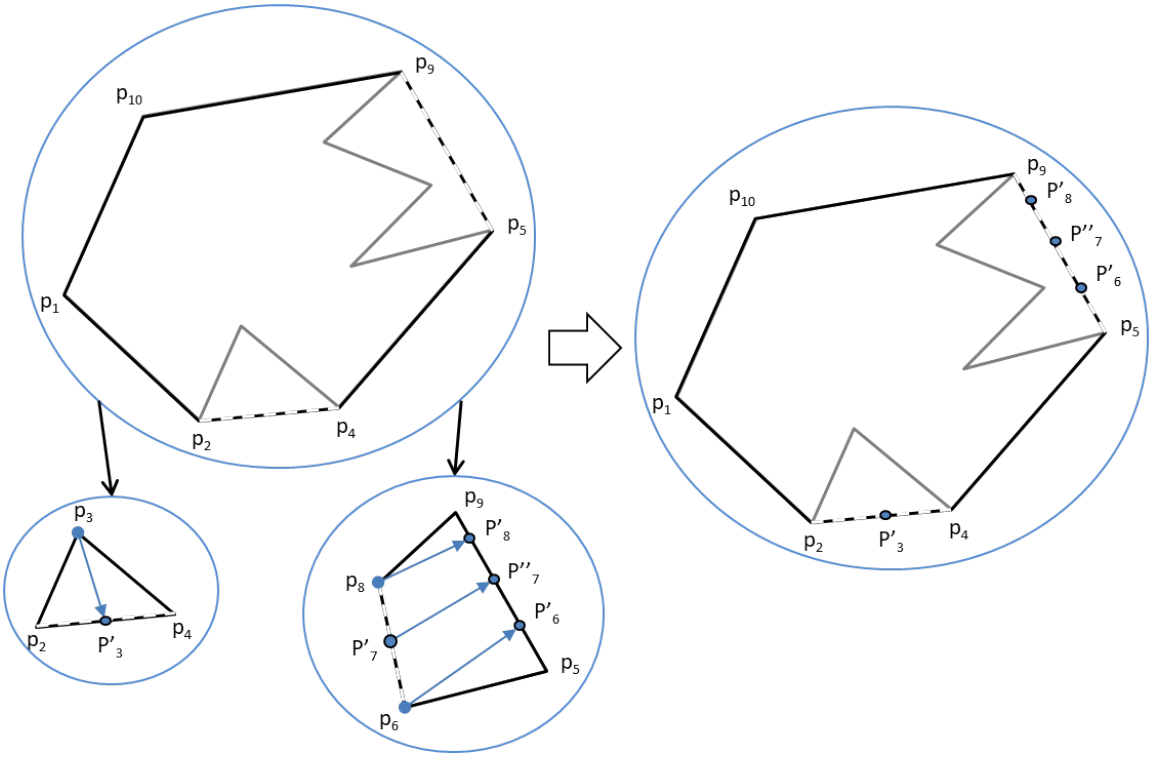


Figura 3.9: Transformação das cavidades elementares de primeira ordem.

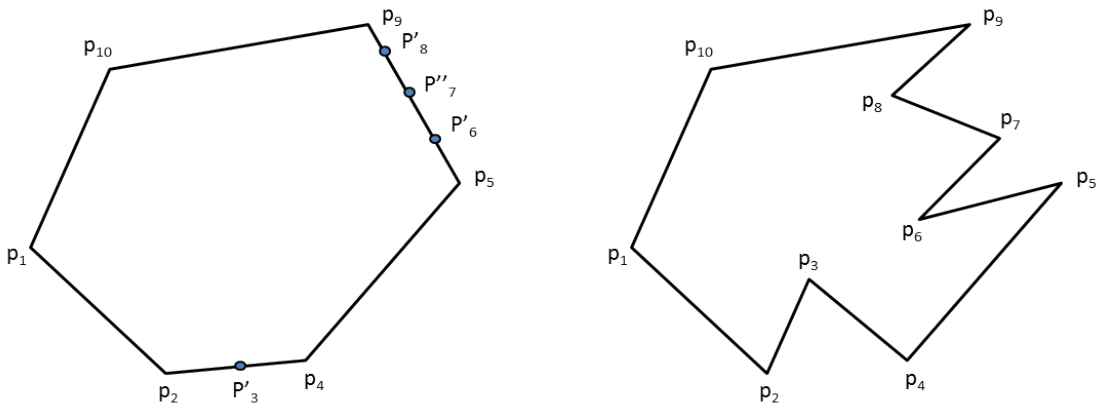


Figura 3.10: Comparação entre contornos.

O processo de triangulação utiliza somente contornos convexos, logo, pela relação de equivalência estabelecida pela transformação T que os contornos não convexos passaram, é possível afirmar que uma aresta (p'_i, q'_j) entre $(C')^{(k-1)}$ e $(C')^k$ é equivalente a uma aresta (p_i, q_j) entre $(C)^{(k-1)}$ e $(C)^k$. Desta forma, realiza-se a triangulação normalmente entre os contornos transformados, e ao final do processo é necessário somente criar as arestas equivalentes [22].

Vários algoritmos para gerar triangulação já foram propostos na literatura para resolver este tipo de problema. Em geral, esses algoritmos utilizam como valores de entrada pontos dispersos no espaço e sem nenhum tipo de organização reconhecível, muito diferente do presente caso onde os pontos estão em uma ordem bem definida, dispostos ao longo de contornos presentes em fatias, com uma distância fixa entre elas. Partindo deste fato, o algoritmo de triangulação pode ser reduzido a conectar duas fatias de cada vez.

Existem alguns algoritmos que foram desenvolvidos para resolver o problema de conectar fatias, dentre eles dois serão descritos a seguir, o algoritmo de Christiansen e Sederberg [23] e o algoritmo de Ekoule, Peyrin e Odet [22] que será utilizado neste trabalho.

3.4.1 Algoritmo de Christiansen

Este algoritmo tem como base a escolha das arestas mínimas entre contornos adjacentes. Dado o contorno superior T (*top*) e o inferior B (*bottom*) como mostrada na Figura 3.11.

O algoritmo escolhe como primeiro ponto de cada contorno os pontos mais próximos das duas fatias para a triangulação ter sucesso. Duas regras são seguidas:

- Para dois pontos no mesmo contorno fazerem parte do mesmo triângulo eles devem ser vizinhos;
- Não podem ser escolhidos mais do que dois pontos no mesmo contorno.

A conectividade dos contornos deve seguir a mesma direção, e os primeiros pontos devem ser próximos.

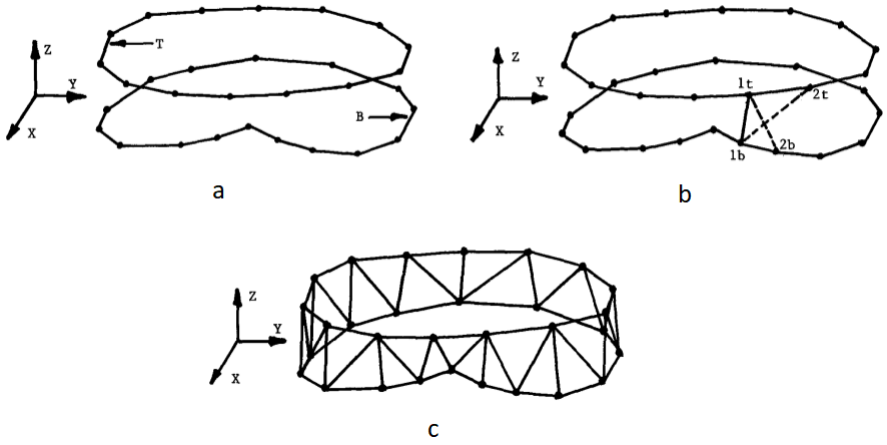


Figura 3.11: Algoritmo de Christiansen: (a) Contornos T e B; (b) As duas possibilidades de segunda conexão; (c)Contorno ao final do algoritmo [23].

Passo 1: Escolher dois pontos iniciais de cada contorno.

Passo 2: Criar aresta entre os dois pontos escolhidos de cada contorno (1t e 1b na Figura 3.11). Escolher a mesma direção de busca em ambos contornos

Passo 3: O próximo ponto é escolhido através da menor diagonal até os pontos vizinhos imediatos (2t ou 2b na Figura 3.11).

Passo 4: Criar aresta conectando o ponto escolhido no passo 3 formando assim um triângulo.

Passo 5: Repetir o passo 3 e 4 para o resto do contorno (Figura 3.11c).

Este algoritmo funciona muito bem para fatias que possuam aproximadamente a mesma quantidade de pontos, mas quando é necessário conectar fatias, onde uma fatia possui mais pontos do que a outra, ele possui um desempenho insatisfatório, deixando alguns pontos sem conectar. Os contornos também precisam ter formatos próximos e precisam ser centrados para que a distância entre os pontos seja a mais próxima possível. Existem artifícios que podem ser utilizados para contornar os problemas de quando os contornos têm formatos diferentes e de quando as fatias não são centradas, mas é impossível contornar o problema de quando há uma quantidade diferente de pontos.

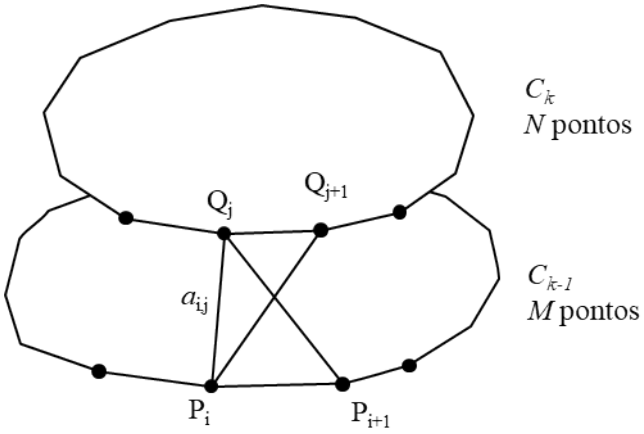


Figura 3.12: Contornos P_i e Q_j

3.4.2 Algoritmo de Ekoule

O algoritmo de Ekoule [22] é baseado em um método heurístico com um critério de escolha de uma aresta de comprimento mínimo. Dado z_1 , ..., z_k uma pilha de fatias ordenadas, Ekoule demonstrou como realizar a conexão do contorno (C_{k-1}) da fatia no nível z_{k-1} ao contorno (C_k) da fatia no nível z_k. O contorno (C_{k-1}) e (C_k) devem ser contornos fechados e convexos, e (P_i) para $1 \leq i \leq M$ é a sequência de pontos que representa o contorno (C_{k-1}) e respectivamente (Q_j) para $1 \leq j \leq N$ é a sequência de pontos que representa C_k. Em ambos os contornos os pontos devem estar dispostos na ordem anti-horária.

As seguintes regras são definidas para o funcionamento do algoritmo:

- Cada aresta deverá ter um ponto no contorno (C_{k-1}) e o outro ponto no contorno (C_k);
- Duas arestas consecutivas da lista solução devem possuir somente um ponto em comum de modo que se forme um triângulo (P_i, P_{i+1}, Q_j) ou (P_i, Q_j, Q_{j+1}) como na Figura 3.12.

Dados os contornos P e Q, tal que M é número de pontos de Q e N número de pontos de P, com $M \leq N$, como mostra a Figura 3.13 a, dois contornos em fatias paralelas, o algoritmo de Ekoule compreende em duas etapas: a primeira etapa realiza uma conexão global entre os contornos, criando as arestas base; a segunda etapa realiza o complemento da primeira etapa, em conectar pontos não conectados na primeira

etapa e formar triângulos. A conexão global se dá de forma que cada ponto do menor contorno (P_i) seja conectado ao ponto mais próximo da fatia com o maior contorno (Q_j), formando assim as arestas base como mostrado na Figura 3.14 b. Na etapa complementar se conectam os pontos que pertencem ao conjunto $S_i = \{Q_j / j \in [j_{(i)}, j_{(i+1)}]\}$ a P_i ou a P_{i+1} de forma que sejam feitos triângulos, exemplificado na Figura 3.13 c. Se Q_q é um ponto que pertence a S_i que está localizado entre $Q_{j(i)}$ e $Q_{j(i+1)}$, de forma que $d(P_i, Q_q) > d(P_{i+1}, Q_q)$. Os pontos livres Q_k anteriores a Q_q que ainda não foram conectados, serão conectados a P_i, e os pontos Q_k restantes e posteriores a Q_q serão conectados a P_{i+1}.

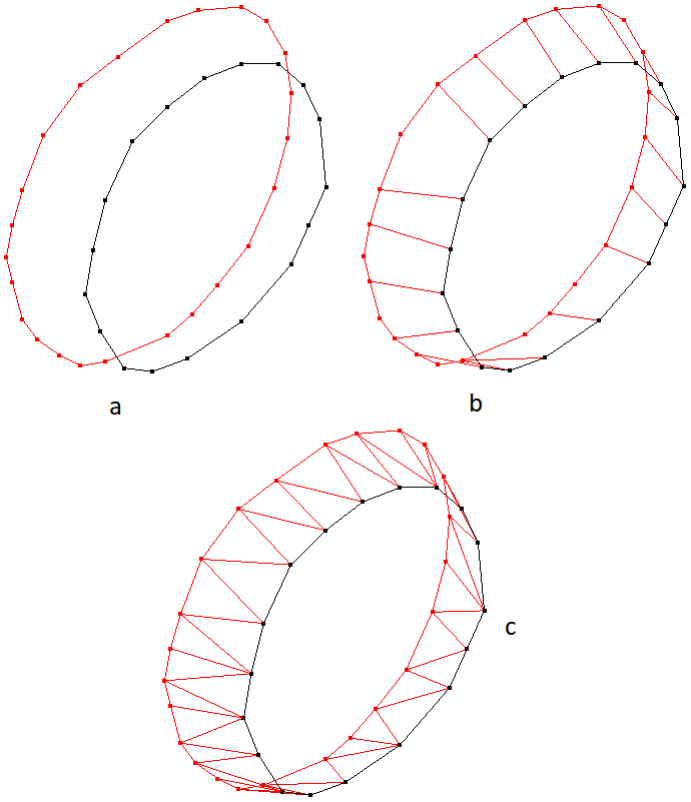


Figura 3.13: Processo de triangulação: (a) Contornos; (b) Arestas-base; (c) Contornos triangulados.

Para efeitos de implementação, são considerados os contornos A e B de forma a mantermos A como o contorno da fatia atual em análise e B como o contorno da fatia imediatamente seguinte, sem importar a relação entre seus números de pontos. Enquanto os símbolos P e Q são associados para contornos de menor e maior número de pontos, respectivamente. São mostrados duas situações dessas relações de representação de contornos na Figura 3.14, o contorno A representado pelo contorno

da cor preta e o contorno B representado pela cor vermelha.

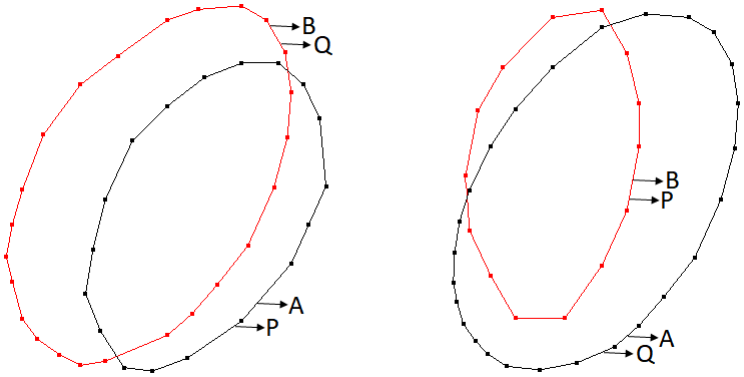


Figura 3.14: Dois exemplos de contornos.

A primeira etapa do algoritmo é a criação das arestas base, partindo do contorno com menos pontos (P) para o contorno com mais pontos (Q). A primeira iteração do algoritmo verifica se os contornos estão alinhados ou não e se os primeiros pontos são realmente os mais próximos entre os contornos.

A comparação do comprimento das arestas é iniciada a partir do primeiro ponto do contorno P com os pontos vizinhos anteriores ao ponto inicial do contorno Q, e continua à medida que o comprimento da aresta vai reduzindo. Assim que o comprimento aumenta a comparação volta ao ponto inicial e seus vizinhos posteriores. A primeira aresta é formada entre o primeiro ponto da fatia P com o ponto encontrado nesta busca no qual a aresta formada possui o menor comprimento. Nesta etapa do algoritmo o ponto selecionado no contorno Q não pode se conectar a nenhum outro ponto do contorno P. As arestas seguintes são criadas através de buscas somente nos vizinhos posteriores, pois os anteriores já não estão mais disponíveis para conexão.

A primeira etapa se completa quando cada ponto do contorno P se conecta a um ponto diferente do contorno Q, formado assim as arestas base. Como o contorno Q possui mais pontos que o contorno P, ainda ficam alguns pontos intermediários sem nenhuma conexão com a outra fatia. Por tanto, é realizada a segunda etapa tendo como principal objetivo criar arestas entre as arestas criadas na primeira etapa, estas arestas servem para conectar os pontos intermediários e dar forma à malha de triângulos. Foram identificados dois casos distintos de possíveis tipos de conexões de arestas nesta segunda etapa:

Caso 1: Este caso é o mais simples. Nele não há pontos intermediários entre as

arestas base, e junto com o traçado dos contornos elas formam um quadrilátero. Como o objetivo do algoritmo é criar uma malha de triângulos, é necessário criar uma aresta conectando as duas já existentes, para assim dividir o quadrilátero ao meio formando dois triângulos. Só existem duas possíveis arestas diagonais, e a escolha é bem simples: a aresta com menor comprimento. Como pode ser visto na Figura 3.15.

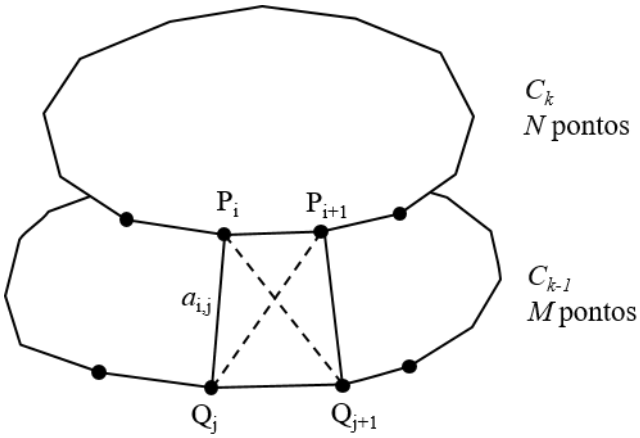


Figura 3.15: Caso 1.

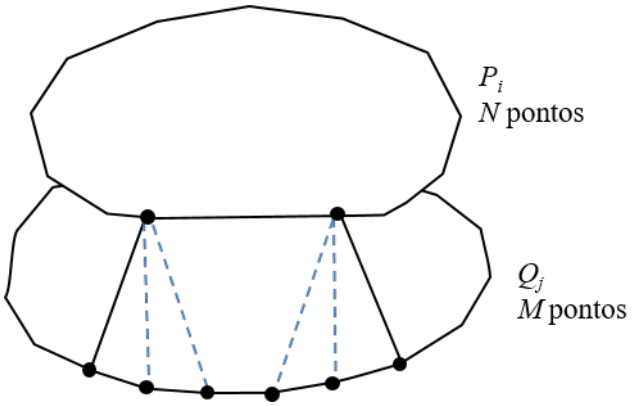


Figura 3.16: Caso 2.

Caso 2: No segundo caso existem pontos intermediários entre as arestas existentes, eles devem ser conectados a P_i ou a $P_(i+1)$. A seleção de qual ponto intermediário irá se conectar a cada um dos dois pontos disponíveis também é baseada na criação de uma aresta com o menor comprimento possível, e é feito mediante o Algoritmo 3.1:

Algoritmo 3.1: Triangulação

```
1 Enquanto existirem pontos intermediarios livres:
2   Inicio
```

```
3     Seleciona o primeiro ponto intermediario livre;
4     Verifica se a aresta de menor comprimento e com o ponto i ou com o
          ponto i+1;
5     Se for com o ponto i:
6       Cria a aresta com o ponto i;
7     Se for com o ponto i+1:
8       Na primeira iteracao desta opcao:
9         Cria aresta entre o ponto intermediario e i+1;
10        Cria uma aresta diagonal entre as duas ultimas
11        arestas criadas (caso 1);
12      Iteracoes seguintes:
13        Cria aresta entre o ponto intermediario e i+1;
14  Fim
```

Uma vez gerada a lista de triângulos anexa à fatia A em relação à fatia B, são calculados os vetores unitários normais de cada aresta. Esse vetor normal sempre aponta para lado exterior do objeto, obtendo-se pelo produto vetorial entre duas arestas de origem comum do triângulo. Esse vetor normal é usado para renderização do objeto e para sua impressão 3D.

Capítulo 4

Implementação

A implementação de reconstrução 3D tem como entrada um conjunto de arquivos descrevendo os n contornos. Por tanto, cada arquivo representa uma fatia. Em condições não limitadas a um contorno e uma fatia; ou seja, para caso genérico de uma fatia com vários contornos, em cada arquivo teria a descrição de todos os contornos que contem cada fatia. A descrição dos contornos nesta etapa é o número de pontos e as suas coordenadas (x, y), por se tratar cada fatia em um plano. Esses arquivos de formato texto (*.txt*) são gerados como consequência do processo de pre-processamento e detecção dos contornos a partir de imagens de seções transversais. Como saída é gerado arquivo de triângulos e seus respectivos vetores normais para posterior impressão 3D.

Em resumo, o processo implementado consiste em três etapas, tal como ilustra a Figura 4.1, e cada etapa é executada separadamente. Na primeira etapa, são gerados os arquivos "*.txt*" de pontos dos contornos. Em condições normais, as coordenadas de pontos de contornos são geradas por programa em C usando a Biblioteca *OpenCV* para processamento de imagens. No entanto, foram geradas coordenadas de pontos de contornos sintéticos para os respectivos testes. Na segunda etapa, a reconstrução que é o foco do trabalho, se implementa na linguagem C e usando biblioteca *OpenGL* para criação do fecho convexo, triangulação e permite eficientemente a manipulação de primitivas geométricas em 3D para exibições na janela. Na terceira parte, como complementar, usa-se o arquivo gerado pela etapa 2 para a respectiva impressão em sólido 3D.

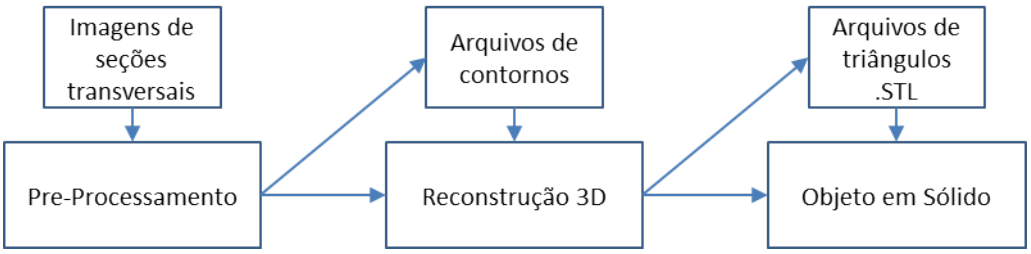


Figura 4.1: Sequência da implementação.

A Figura 4.2 ilustra a etapa de reconstrução 3D, etapa principal deste trabalho. Os arquivos de contornos são capturados pelo módulo Carrega e alocados na estrutura de dados como lista de vetores. A partir dos respectivos vetores são computados os fechos convexos e armazenados como tal na estrutura das fatias correspondentes. Na sequência seguinte, em função dos fechos convexos e suas projeções são geradas as arestas unindo pontos de pares adjacentes de contornos, definindo nesta parte uma lista de arestas. As arestas são fundamentais para o estabelecimento da lista de triângulos com seus atributos normais. Listas de triângulos são elementos para a geração de objetos 3D e sua impressão como sólido.

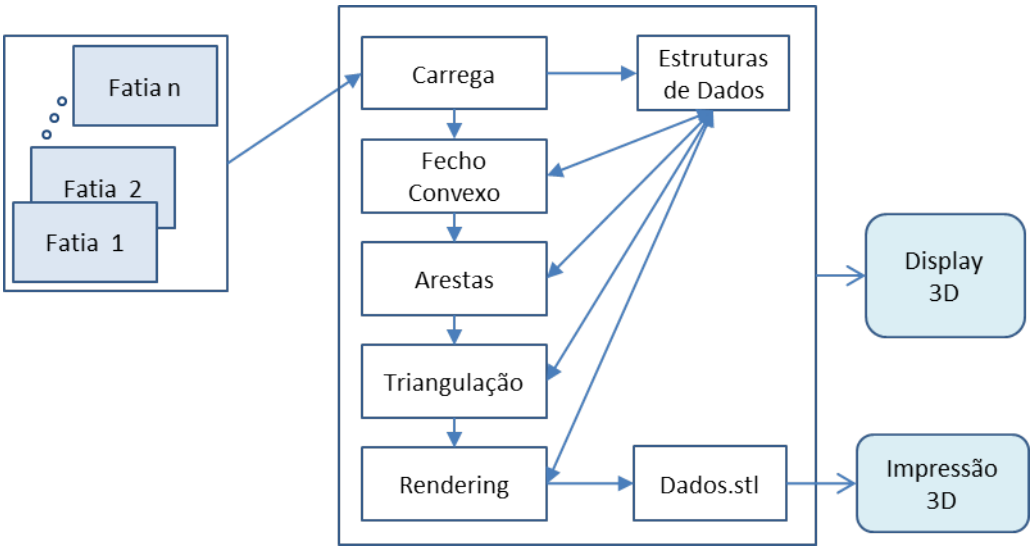


Figura 4.2: Estrutura do programa que implementa a reconstrução.

4.1 Estrutura de dados

Para a geração do objeto 3D é criada uma variável do tipo objeto. Essa estrutura possui uma variável indicando a quantidade de fatias que irão compor o sólido e dois ponteiros, um para a primeira fatia da lista, e outro para a última fatia da lista. Cada fatia possui uma lista que será carregada com todos os pontos da fatia, uma lista para armazenar o fecho convexo dos contornos presentes na fatia, uma lista que é utilizada pela função de triangulação para criação de arestas entre a fatia em uso e a seguinte, e em seguida uma lista para armazenar todos os triângulos entre a fatia em uso e a seguinte que serão criados com essas arestas. Na Figura 4.3 a seguir uma representação gráfica desta estrutura.

Um conceito muito importante para a implementação deste programa é o de contornos fechados. O último ponto da lista é conectado ao primeiro, ou seja, a lista é circular. Se uma função está percorrendo toda a lista sempre deve haver um mecanismo de conexão entre os dois extremos da lista.

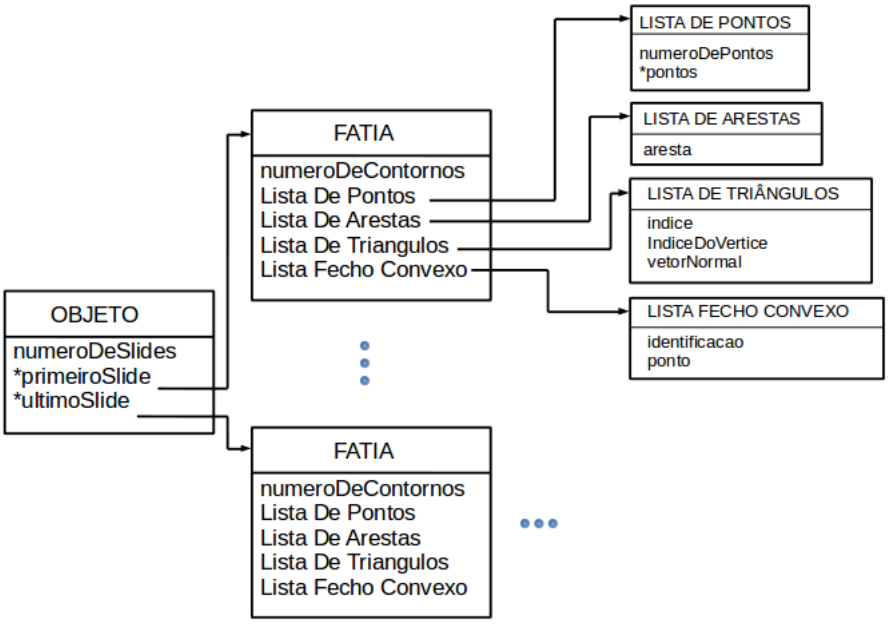


Figura 4.3: Estrutura de dados.

4.2 Fluxo de processamento

O programa se inicia com todos os processos da biblioteca *GLUT* (do *OpenGL*) para controle do sistema operacional, como por exemplo a criação da janela que mostrará o progresso da reconstrução, assim como a configuração de todos os parâmetros necessários para o perfeito funcionamento da biblioteca.

Ao clicar com o botão direito do mouse na janela criada é mostrado um menu com as opções disponíveis: carrega de dados, fecho convexo, criação de arestas, triangulação e renderização em 3D, tal como ilustrada pela Figura 4.4.

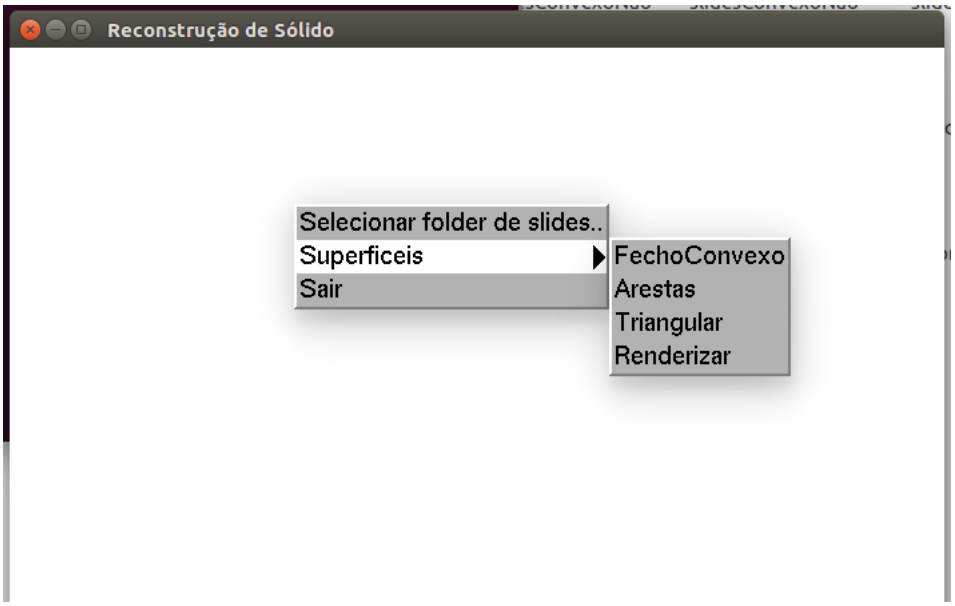


Figura 4.4: Menu do programa.

Ao selecionar a opção carregamento de dados é necessário indicar uma pasta específica no computador onde se encontram todos os arquivos referentes às fatias do sólido a ser reconstruído. Esses arquivos devem seguir uma regra de nomeação da seguinte forma, ele deve ser nomeado "contorno-xx.txt" onde xx representa a posição da fatia no sólido, essa posição começa obrigatoriamente em 01 e vai até 99. Dentro de cada arquivo, deve conter na seguinte ordem as informações: número de identificação do contorno, pois é possível que exista mais de um contorno em cada fatia apesar da limitação deste trabalho; na linha seguinte o número de pontos no contorno; e nas linhas seguintes as coordenadas de cada ponto deste contorno; caso exista um outro contorno nesta fatia, ele deve ser descrito em seguida seguindo as mesmas regras. A Figura 4.5 a seguir traz um arquivo como exemplo.

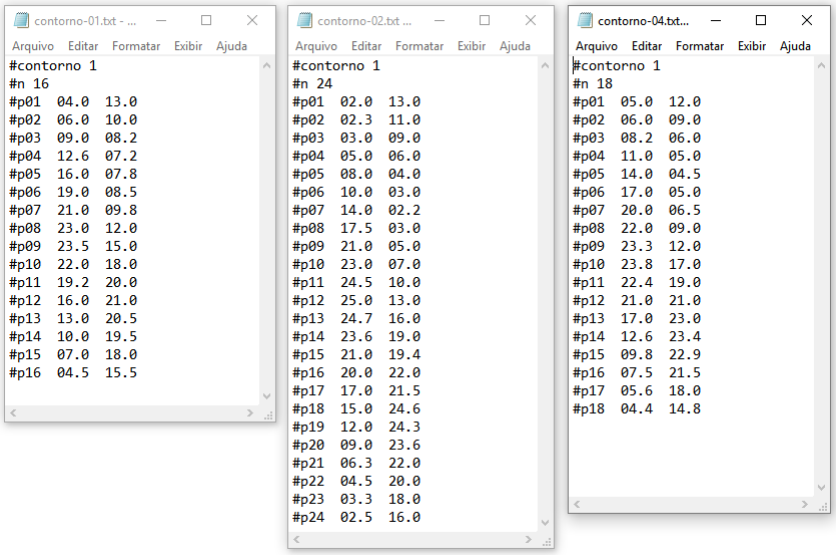


Figura 4.5: Três exemplos de arquivo de inserção de dados.

Após o carregamento dos dados, os contornos são exibidos na tela para que o usuário acompanhe o processamento. É necessário selecionar a opção fecho convexo para iniciar a próxima fase do processamento (esse processo será detalhado nas próximas sessões). O resultado é mostrado na tela. Com a opção arestas, criam-se as arestas-base entre todos os contornos e dar início ao processo de triangulação, e novamente os resultados são exibidos na tela. O próximo passo é a parte da triangulação em si, onde são criados os triângulos de fato, são inseridas as arestas diagonais entre as arestas que ligam as fatias, os resultados são mostrados na tela. Neste estágio o sólido já está completo, mas somente por arestas, parecendo uma estrutura de aramado. A opção renderização em 3D criará a "superfície" do sólido para melhorar a sua visualização.

4.3 Fecho convexo e transformação

Dado um vetor de pontos de um contorno, o objetivo nesta parte é construir o fecho convexo desses pontos com as respectivas projeções sobre ele dos pontos do contorno que não pertencem ao seu fecho convexo simples. Isso significa que construiremos o fecho simples, e em forma hierárquica iremos descompondo suas concavidades de primeira ordem e projetá-las nas arestas de concavidades do fecho convexo correspondente, tal como explicado no Capítulo 3 em suas seções correspondentes a convexidades,

concavidades e transformações.

Esse processo está associado a uma estrutura hierárquica de árvore recursiva, neste caso em busca em profundidade – ascendente de esquerda para a direita. Significa que transforma, com dados posição inicial e final dos pontos do vetor de contorno (ou sub-contorno) em análise, será construído seu fecho e construído seu respectivo fecho convexo e continuar analisando por fecho convexo. O Algoritmo 4.1 mostra a lógica seguida na implementação.

Algoritmo 4.1: Transformação do fecho convexo

```
1  ListaFechoConvexo transforma(vetorPontos)
2    criaFecho(fecho) //copia lista de pontos originais para a lista do
       fecho convexo
3    criaFechoConvexo (fechoConvexo) //cria um fecho convexo do contorno
4    se (fecho != fechoConvexo)
5      concavidade = procuraConcavidade1Ordem(fechoConvexo)
6      enquando existe concavidade
7        transforma(concavidade)
8        projeta(concavidade, fechoConvexo)
9        procuraConcavidade1Ordem(pontosSeguintes)
10       libera(concavidade)
11     fim enquanto
12   fim se
13   retorna fechoConvexo
14 fim
```

Cria-se uma lista de fecho convexo, que será uma cópia da lista de todos os pontos do contorno para que não se perca a referência dos pontos originais. Esta lista é do tipo duplamente encadeada e o seu final é conectado ao início. Com essa lista pronta, cria-se um fecho convexo. Se o fecho convexo for igual ao contorno original, então o contorno é convexo e a função termina ai. Se o fecho for diferente do fecho convexo significa que o contorno é côncavo, e existe pelo menos uma concavidade de primeira ordem. Sendo assim começamos a construir a árvore de concavidades. Localiza-se a primeira concavidade, e como a função é recursiva ela é chamada novamente até que não exista mais concavidades dentro desta primeira concavidade. Desta forma realiza-se uma busca de concavidades em profundidade, ao invés do sugerido por Ekoule, Peyrin e Odet [22] que fizeram uma busca em largura. Quando esta busca atinge a folha da árvore de concavidades, inicia-se o processo de projeção dos pontos

côncavos. O ponto côncavo é projetado na aresta de projeção, e a concavidade é liberada da árvore, esse processo se repete até voltar a primeira chamada da função, que segue procurando concavidades pelo resto do contorno.

Da mesma forma que ocorre na representação da teoria onde os subscritos das concavidades decrescem para manter a orientação, ocorre também no algoritmo. O fecho convexo, em sua criação, tem uma orientação padrão anti-horária na qual ao ser percorrida a aresta, o interior do contorno estará sempre a esquerda. A medida que se encontram novas concavidades, e a função de transformação do fecho convexo é novamente chamada, a orientação se mantém anti-horária, mas como a referência de interior do contorno muda a ordem de percorrer os pontos também muda invertendo os subscritos como pode ser visto na Figura 4.6.

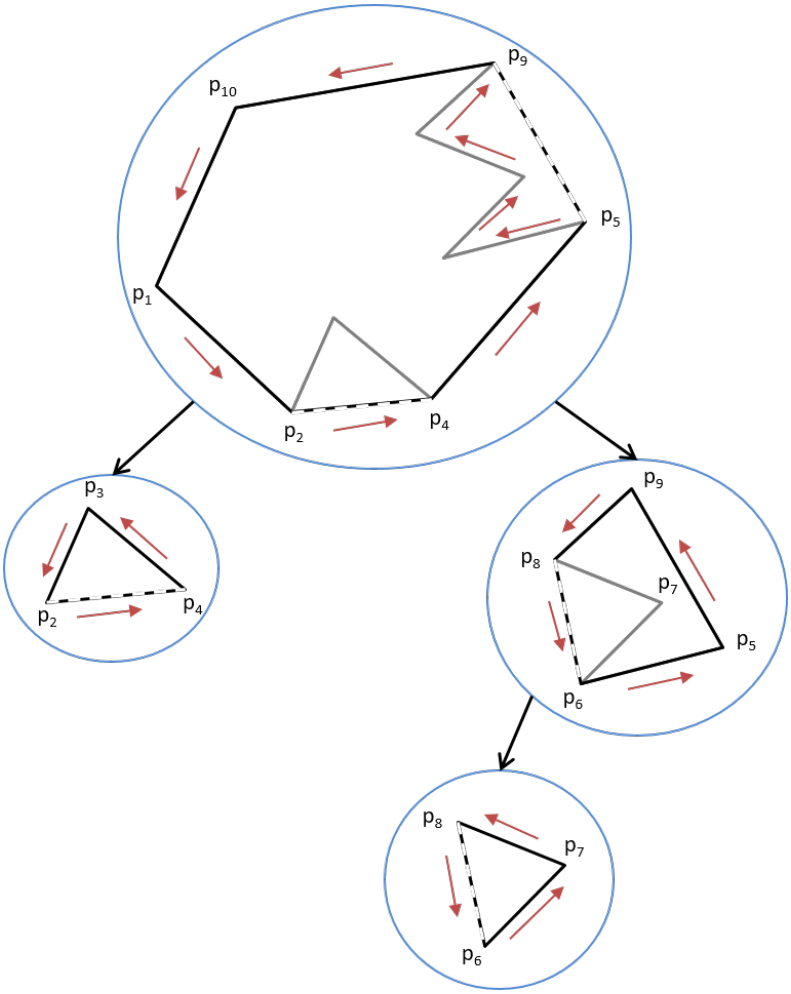


Figura 4.6: Orientação dos contornos.

A função para criação de fecho convexo tem uma logica de acordo com o Algoritmo

4.2:

Algoritmo 4.2: Criação de fecho convexo

```
1  ListaFechoConvexo criaFechoConvexo(fechoConvexo)
2    ponto = fechoConvexo
3    enquanto(ponto->w < 2)    // cada ponto e verificado duas vezes
4      fc = verificaPontoConvexo(ponto) // verifica se o ponto e convexo
5      se(!fc) //se nao convexo
6        removePontoNaoConvexo (fechoConvexo, ponto)
7        ponto->w = ponto->w  - 1
8      senao
9        ponto->w = ponto->w + 1
10       ponto = ponto->proximo
11     fim se
12   fim enquanto
13
14   ponto->w = 3          // para identificar o inicio da lista
15   ponto->anterior->w = 4   // para identificar o fim da lista
16   retorna(fechoConvexo)
17 fim
```

O fecho convexo é criado da seguinte forma: cada ponto da lista duplamente encadeada de fecho convexo possui uma variável w associada, que indicará quantas vezes cada ponto foi verificado. Cada ponto deve ter a sua convexidade verificada duas vezes para verificar falhas. A função inicia com um loop do tipo enquanto que verifica quantas vezes cada variável foi verificada. Assim que encontra a primeira variável que foi verificada duas vezes ele se encerra. Dentro deste loop é feita uma chamada para a função de verificação de convexidade, essa função tem como parâmetro de entrada somente um ponto da lista de fecho convexo. Se o ponto não for convexo, ele é removido da lista, caso contrário a variável w é incrementada de uma unidade. Após o término do loop enquanto, a primeira variável encontrada com a variável w com valor dois será o primeiro ponto da lista. Sendo assim a sua variável w associada recebe o valor três para indicar o início da lista, e o ponto anterior recebe o valor 4 para indicar o final.

A função de verificação de convexidade é descrita no Algoritmo 4.3, e recebe como parâmetro um ponto, e a partir deste ponto são calculados dois vetores: um partindo do ponto anterior ao ponto atual, e o segundo partindo do ponto atual até o ponto seguinte. Calcula-se então o produto vetorial entre os dois vetores, se o vetor

resultante tiver uma orientação positiva o ponto de análise é convexo, caso contrário se o vetor tiver uma orientação negativa o ponto não é convexo.

Algoritmo 4.3: Verificação de pontos convexos

```
int verificaPontoConvexo(ponto)
  subtrai(ponto, ponto->anterior, va) //criando o vetor a
  subtrai(ponto->proximo, ponto, vb) //criando o vetor b
  s = produtoVetorial(va, vb)
  se (s>0)
    convexo = 0;
  senao
    convexo = 1;
  fim se
  retorna (convexo)
fim
```

A função de localização de concavidades é bem simples e está descrita no Algoritmo 4.4, ela localizará a concavidade pelos índices faltantes. Basta percorrer a lista de pontos verificando a diferença entre os índices, diminui o valor do próximo índice pelo valor do índice atual, se a diferença for maior do que 1 (um) está faltando pelo menos um ponto entre esses dois pontos. Ou seja, existe uma concavidade entre eles.

Algoritmo 4.4: Identificação de concavidades

```
ListaFechoConvexo concavidade1Ordem (fechoConvexo)
  ponto = fechoConvexo
  enquanto (ponto->w != 4) // enquanto nao e o ponto final
    se (ponto->proximo->id - ponto->id > 1)
      termina //tem concavidade
    senao
      ponto = ponto->proximo
    fim se
  fim enquanto
fim
```

A função de projeção de pontos, Algoritmo 4.5, recebe como dado de entrada os pontos da concavidade que formam a aresta de projeção, e que também são o primeiro e último ponto da concavidade.

Algoritmo 4.5: Projeção

```
void projeta (deste, para)
```

```
2   criaVetorProjecao(deste, para, vetorProjecao)
3   distanciaArestas = somaArestas(concavidade) //somatorio do comprimento
        das arestas
4   distanciaAcumulada = 0.0f
5   enquanto (ponto->w < 4)                  //enquanto nao e o ultimo ponto
        da concavidade
6     distancia = ponto->xyz  - ponto->anterior->xyz
7     distanciaAcumulada = distanciaAcumulada + distancia
8     distanciaFinal = distanciaAcumulada / distanciaArestas
9     novoPonto = criaPontoProjetado()
10    calculaCoordenadas(ponto, vetorProjecao,novoPonto)
11    inserePontoVetorProjecao(novoPonto)ponto = ponto->proximo
12  fim enquanto
13 fim
```

O primeiro passo do algoritmo é fazer a criação do vetor de projeção. Esse vetor deve possuir uma orientação oposta à aresta de projeção, que serve como indicativo da sequência certa de inserção dos pontos, do ponto deste para o ponto para.

Em seguida inicia-se o processo de calculo da projeção do ponto. A função Ri é definida pela divisão entre a posição relativa do ponto na concavidade (distância acumulada) e o somatório do comprimento de todas as arestas. Cria-se então um ponto, que terá as novas coordenadas na aresta de projeção. As novas coordenadas são calculadas com base em Ri, e o ponto é inserido na orientação do vetor projeção, e passa para o próximo ponto. Se existir mais pontos, calcula-se novamente a distância acumulada, o Ri, e as novas coordenadas. O próximo ponto será inserido ao final do vetor projeção. Este processo continua até que se acabem todos os pontos a serem projetados. Na Figura 4.7 temos um exemplo de projeção do ponto P3.

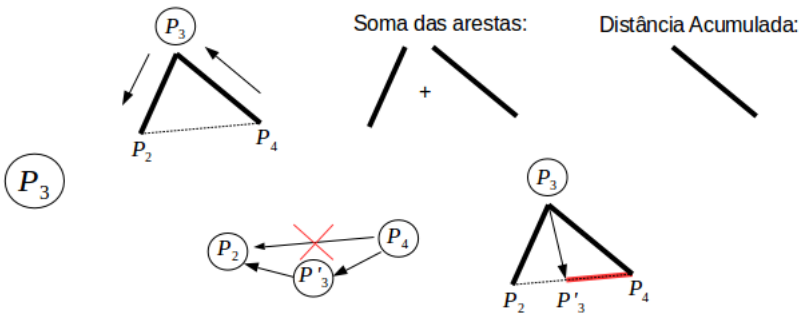


Figura 4.7: Projeção do ponto P3

4.4 Triangulação

O processo de triangulação é dividido em duas etapas, na primeira todos os pontos do menor contorno serão conectados por uma aresta ao contorno com maior número de pontos. Cada ponto de um contorno será conectado a um ponto diferente do outro contorno, não podendo ter mais de um ponto conectado a outro em nenhuma dos dois contornos. A segunda etapa do processo de triangulação conectará todos os pontos que sobraram entre as arestas criadas, escolhendo a mais próxima para realizar a conexão.

A primeira etapa do processo de triangulação se inicia com a definição de qual contorno será o contorno P, que possui menor número de pontos, e qual será o contorno Q, que possui maior número de pontos. A função defineArestaBase() descrita no Algoritmo 4.6 gerencia esta seleção. Em seguida ela cria uma aresta nova através da função novaAresta() entre os dois contornos, e insere na lista de arestas na posição correta. A posição correta será no final da lista de arestas do contorno A, a aresta tendo sentido do ponto do contorno a para o ponto do contorno B. Este processo de criação de novas arestas se repete até que todos os pontos do contorno P tenham um ponto correspondente no contorno Q.

Algoritmo 4.6: Criação de arestas base

```
1  listAresta defineArestaBase(listSlide a, listSlide b)
2    se (a->tamanho < b->tamanho)
3      p = a          // menor numero de pontos
4      q = b          // maior numero de pontos
5      trocaOrdem = 0;       // ordem nao troca
6    senao
7      q = a          // maior numero de pontos
8      p = b          // menor numero de pontos
9      trocaOrdem = 1;      // ordem troca
10   fim se
11
12   para i=0 ate p->tamanho faca //i e a posicao do ponto em analise no
         contorno p
13     j = novaAresta(p,i,q,j) //j e a posicao do ponto em analise no
           contorno q
14     se (trocaOrdem ==0)
15       insereAresta(i,j)
16     senao
```

```
17        insereAresta(j,i)
18      fim se
19    fim para
20
21    //finalizando, a ultima aresta recebe como proximo a primeira
22    ultimaAresta->proximo = primeiraAresta
23  fim
```

A função novaAresta(), Algoritmo 4.7, recebe como parâmetro os dois contornos (P e Q) e os pontos indicando a posição em que eles estão sendo analisados. A função tem como principal objetivo calcular uma aresta de menor comprimento possível entre P e Q partindo do ponto dado em P.

Na primeira iteração, verifica-se os pontos anteriores ao j_0 para verificar se as fatias estão desalinhadas, em seguida verifica-se do ponto j_0 em diante. A menor distância entre os pontos será a aresta selecionada. Nas iterações seguintes verifica-se somente os pontos seguintes, pois os anteriores já estão conectados.

Algoritmo 4.7: Nova aresta

```
1  int novaAresta(int i, listPontos p, int j, listPontos q)
2    se (j<0) entao: // primeira iteracao
3      h = q->n
4      d = distancia (p->ponto[i], q->ponto[h])
5      dmin = d + 0.5f
6      enquanto (d<dmin)      //verifica pontos anteriores ao ponto 0
7        dmin = d
8        d = distancia (p->ponto[i], q->ponto[h])
9        h = h - 1
10     fim enquanto
11     // verificando pontos posteriores a 0
12     h = 0
13     d = distancia (p->ponto[i], q->ponto[h])
14     enquanto (d<dmin) {
15       dmin = d
16       d = distancia (p->ponto[i], q->ponto[h])
17       h = h + 1
18     fim enquanto
19   senao //iteracoes seguintes verifica somente pontos seguintes
20     h = j
21     d = distancia (p->ponto[i], q->ponto[h])
22     dmin = d + 0.5f
```

```
23      enquanto (d<dmin)
24        dmin = d
25        h = h + 1
26        d = distancia (p->ponto[i], q->ponto[h])
27      fim enquanto
28    fim se
29    retorna (h)
30  fim
```

A segunda etapa do processo de triangulação é feita pela função criaArestasEntreAeB(). Dado duas arestas A e B, vizinhas da etapa anterior, esta função calculará as arestas necessarias para conectar todos os pontos livres existentes entre elas, e formar triângulos ao invés dos atuais quadriláteros. Existem dois casos possíveis de conexão entre as arestas, ambos compreendidos no Algoritmo 4.8. O primeiro caso é se não houver nenhum ponto livre entre as arestas A e B, para processamento basta calcular a menor diagonal entre as arestas e inserir na lista.

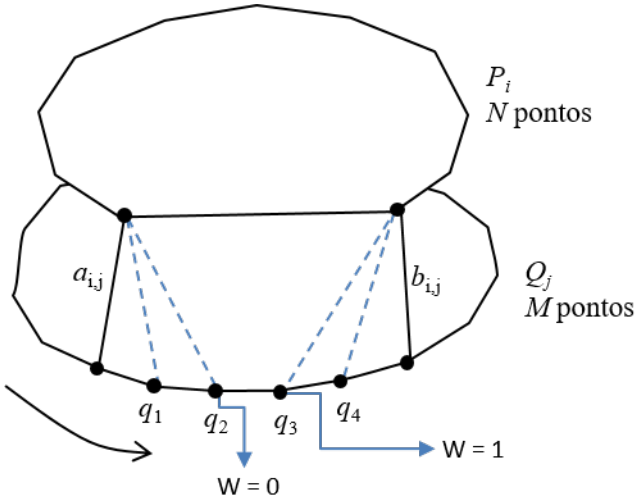


Figura 4.8: Caso 2.

No segundo caso, Figura 4.8 existem pontos entre as arestas, e o processamento é feito da seguinte forma: A conexão dos pontos livres, por questão de controle, inicia com a aresta A. Cria-se uma variável de controle w para verificar quando a conexão passa para a aresta B. O algoritmo inicia verificando qual é a aresta mais próxima, se for a aresta A, a variável w mantem o valor 0, e é realizada a conexão entre o ponto livre com o extremo da aresta A pertencente ao contorno P. Essa verificação continua até que seja encontrado o primeiro ponto que esteja mais próximo a aresta

B. Quando isto ocorre, a variável w recebe o valor 1, e é criada a primeira aresta entre o ponto livre e o extremo da aresta B pertencente ao contorno P. Logo após, é criada uma aresta diagonal entre as duas últimas arestas criadas, a ultima aresta conectada a aresta A, e a primeira conectada a aresta B. A partir do momento que um ponto é conectado a aresta B, todos os pontos seguintes também são conectados. Para finalizar o algoritmo ocorre uma última verificação, se todos os pontos tiverem sido conectados a aresta A, ocorre o caso 1 (um) entre a última aresta criada e a aresta B, sendo assim, é necessário criar uma aresta diagonal para que o poligono quadrilátero criado se tornem dois triângulos.

Algoritmo 4.8: Criação de arestas entre arestas base adjacentes

```
1  listAresta criaArestasEntreAeB(listAresta a, listAresta b, contorno)
2
3    k = q->ponto->proximo
4    se(k == b)//se nao existem pontos entre a e b
5      diagonalMenor(a, b) //caso 1, cria a aresta com a menor diagonal
6      insereArestaLista()
7    senao //caso 2, existem pontos entre a e b
8      w = 0 // indica que o ponto vai conectar a aresta a
9      enquanto (k != b)//enquanto existirem pontos
10       se (w == 0) //se w=0, se conecta a aresta a
11         distanciaMenor(a, k, b, p, q)
12         se (k esta mais proximo de a)
13           insereArestaLista()
14         senao //k conecta por primeira vez a b
15           w = 1 // indica que aconteceu uma conexao para b
16           insereArestaLista()
17           //cria uma diagonal entre as ultimas arestas criadas
18           diagonalMenor()
19           insereArestaLista()
20         fim se
21       senao //conecta os pontos restantes a aresta b
22         insereArestaLista()
23       fim se
24     fim enquanto
25   fim se
26   se (w == 0) //se todos os pontos forem conectados a aresta a
27     diagonalMenor(a, b)
28     insereArestaLista()
29   fim se
```

```
30 fim
```

4.5 Impressão 3D

O processo de geração do arquivo no formato que a impressora 3D reconhece é bem simples. O formato de arquivo STL, que também é conhecido como "Standard Triangle Language" (língua padrão de triângulos), representa o modelo em 3D através de vários polígonos planares de três lados (triângulos). Este arquivo pode ser definido de duas formas: a primeira e mais simples é a ASCII, enquanto a segunda é pela forma binária. O arquivo feito a partir da técnica de ASCII gera um arquivo de tamanho maior, mas é um arquivo que pode ser facilmente lido por editores de textos comuns, pois é uma grande string ASCII, ao contrário da binaria que não pode ser manipulado por esse tipo de software.

A criação do arquivo segue duas regras básicas: cada triângulo precisa de um vetor normal apontando para fora do sólido, e os triângulos devem ser descritos em ordem, eles precisam ser adjacentes.

O arquivo deve seguir a estrutura descrita no Algoritmo 4.9 [18]

Algoritmo 4.9: Estrutura de um arquivo STL

```
1  solid name       // o nome [opcional] depois de solid
2      // inicio do triangulo. Esta parte ira se repetir para todos os
           triangulos
3    facet normal ni nj nk          // vetor normal <ni, nj, e nk>
4      outer loop
5        vertex v1x v1y v1z //coordenada do vertice 1 do triangulo
6        vertex v2x v2y v2z //coordenada do vertice 2 do triangulo
7        vertex v3x v3y v3z //coordenada do vertice 3 do triangulo
8      endloop
9    endfacet //fim do triangulo
10 endsolid name // fim do arquivo
```

4.6 Resultados

O resultado obtido com a implementação foram os seguintes: A Figura 4.9 mostra a visão inicial do programa ao carregar todos os contornos, neste caso contornos criados exclusivamente para teste do programa. Para uma melhor visualização, cada fatia gera contornos de uma cor diferente. A lista possui sete cores, que ao se esgotarem iram se repetir de acordo com a quantidade de fatias, seguindo a ordem de cores temos:

- Fatia 1: contorno preto
- Fatia 2: contorno vermelho
- Fatia 3: contorno verde
- Fatia 4: contorno azul
- Fatia 5: contorno laranja
- Fatia 6: contorno ciano
- Fatia 7: contorno fúcsia

Ao selecionar a opção "Fecho Convexo", são criados todos os fechos convexos dos contornos. Na Figura 4.10 é possível ver mudanças em alguns contornos. Como por exemplo na segunda camada da direita para a esquerda, ela possuía dois pequenos "dentes" em sua parte inferior que foram "cobertos" pelo fecho convexo.

Em seguida é selecionada a opção "Arestas", que criará todas as arestas-base entre os contornos. Podemos ver na Figura 4.11, que neste ponto o sólido já está tomando forma, mas que ainda não é composto por uma malha de triângulos, e sim por polígonos quadriláteros.

A penúltima etapa do algorítimo ocorre após a seleção da opção "Triangular" que dividirá os quadriláteros da etapa anterior em triângulos, e criará enfim o sólido como pode ser visto na Figura 4.12.

Para finalizar, é necessário selecionar a última opção: "Renderizar". Que deixará o sólido com um visual 3D, como pode ser visto na Figura 4.13, e no processo criará o arquivo .stl que será posteriormente impresso.

Ao abrir o arquivo .stl gerado no programa *Repetier Host* versão 2.0.5, um programa para controle de impressão 3D, é a seguinte visão que temos na Figura 4.14.

Selecionando no software a opção "Fatiar", temos o sólido finalmente pronto para impressão, conforme Figura 4.15. É possível ver nas estatísticas de impressão geradas pelo programa que esse solido possui 40 fatias, e um tempo estimado de impressão de 4 minutos e 31 segundos. Ao ser impresso com um filamento do tipo plástico ABS com aproximadamente 1,75mm de diamentro, será necessário 15 centímetros deste filamento para completar a impressão.

Caso o objeto seja muito pequeno como este gerado, é possível alterar o seu tamanho na escala do próprio programa de impressão, como pode ser visto na Figura 4.16.

Passando pelo mesmo processo de fatiamento, temos que este sólido ampliado possui 121 camadas e será impresso em 15 minutos e 51 segundos. Utilizando o mesmo filamento que no exemplo anterior, será necessário 1,56 metros deste filamento

Utilizando uma impressora 3D do modelo Anet A8, com filamento de diametro 1,75mm do tipo plástico ABS na cor vermelho goiaba, foi possível imprimir o sólido reconstruido conforme mostrado na Figura 4.18.

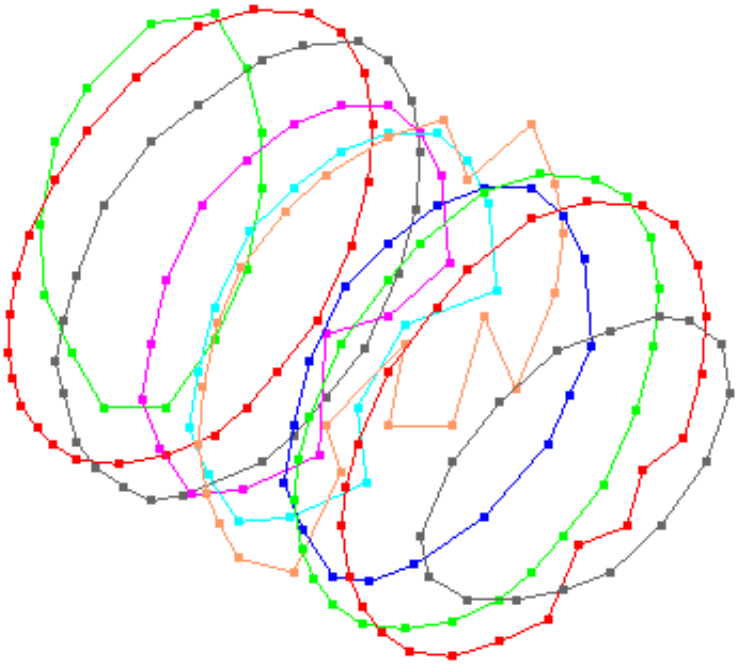


Figura 4.9: Contornos.

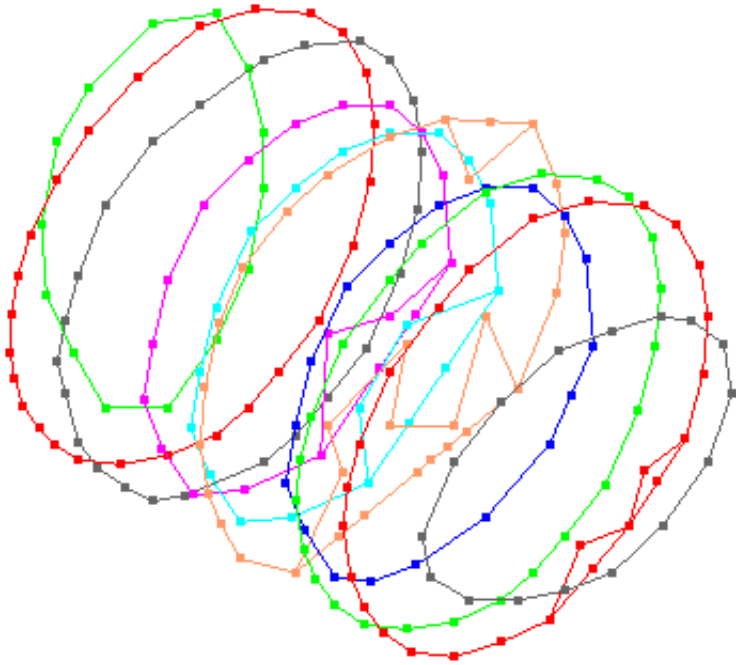


Figura 4.10: Fecho convexo.

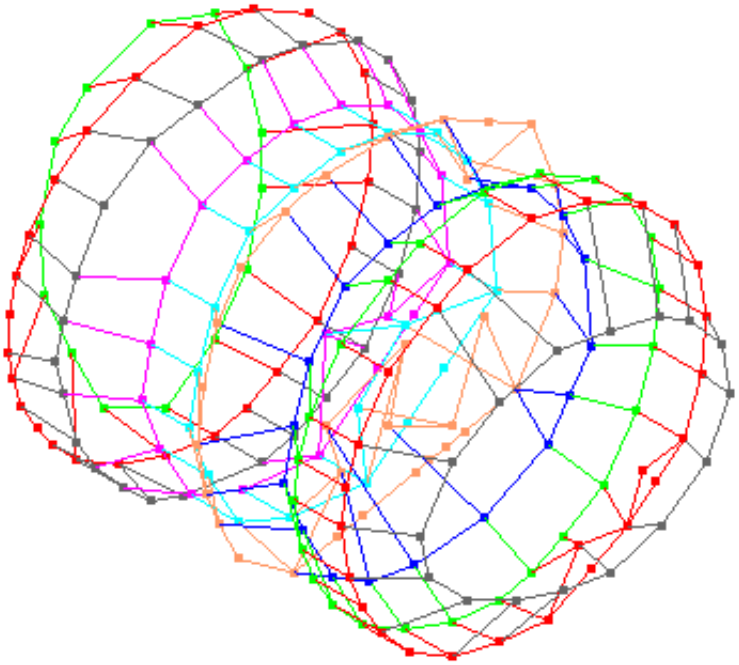


Figura 4.11: Arestas-base.

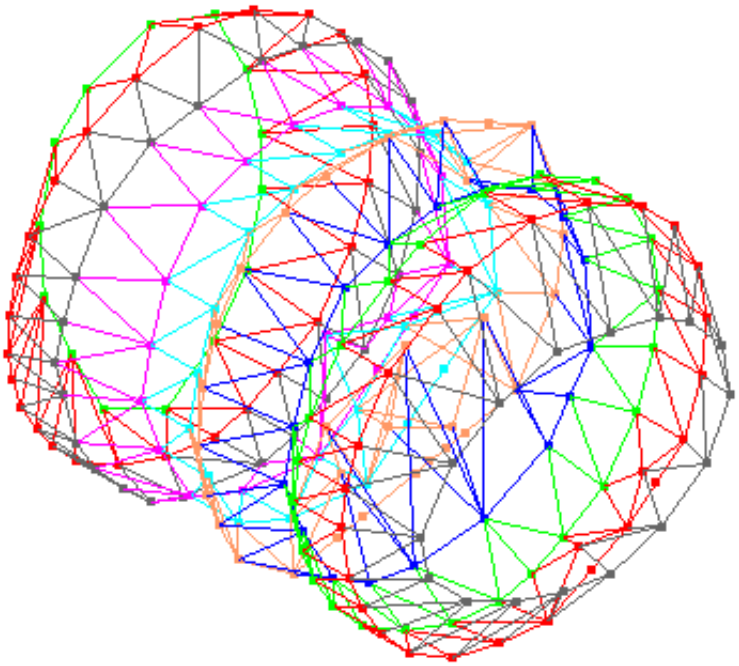


Figura 4.12: Triangulação.

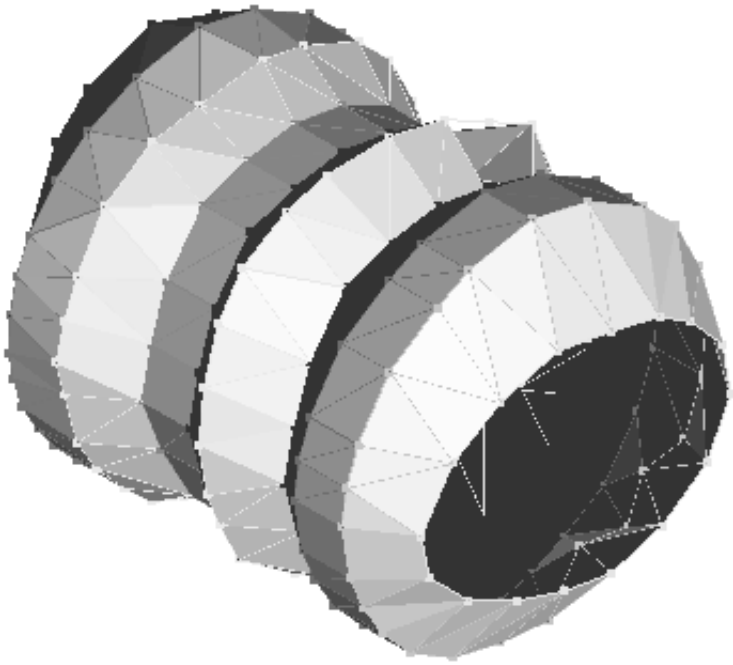


Figura 4.13: Renderizar.

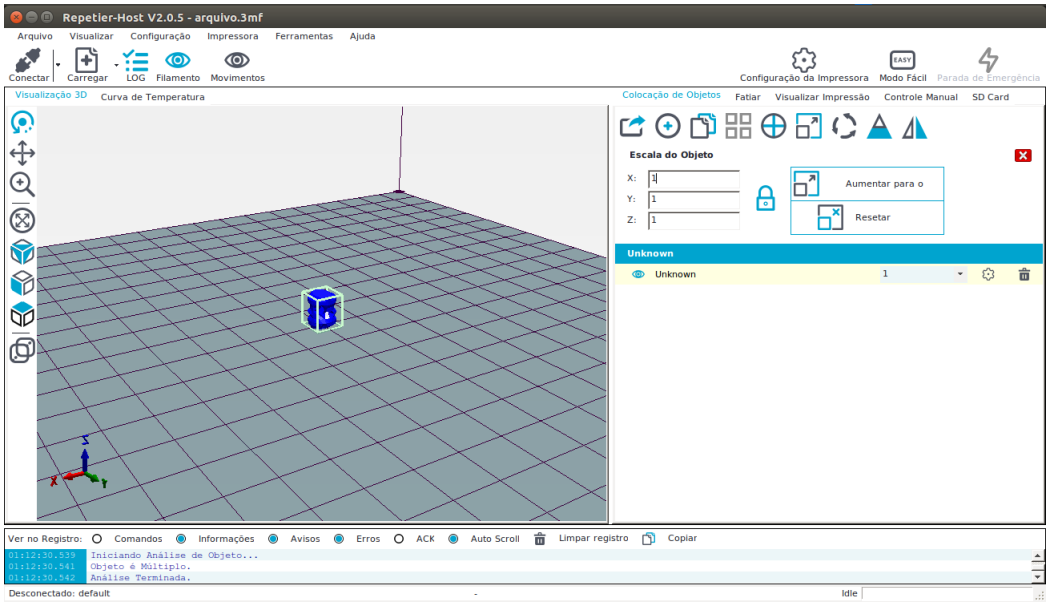


Figura 4.14: Programa de gerenciamento de impressora 3D.

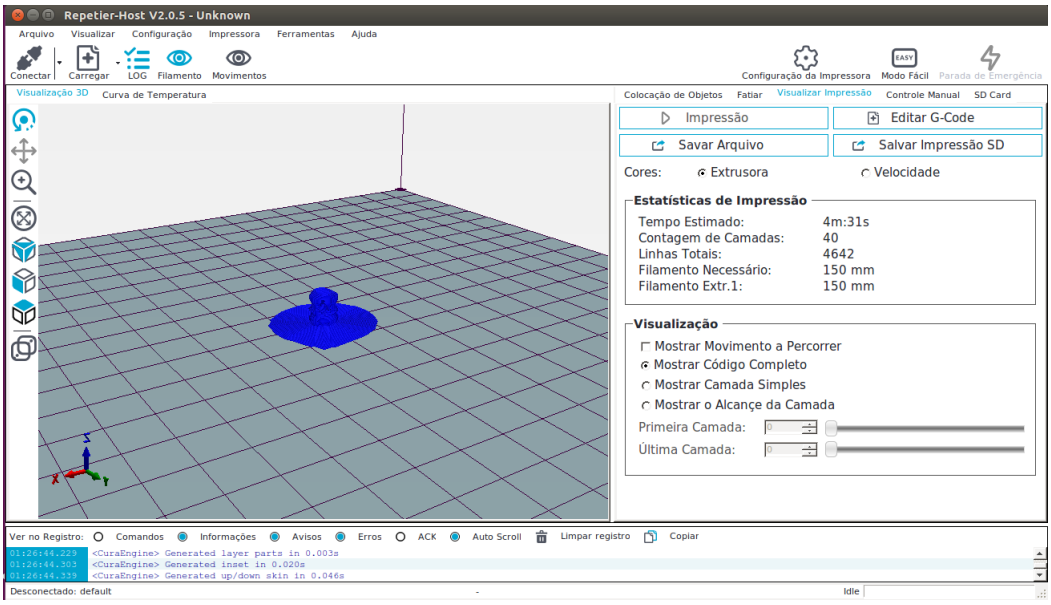


Figura 4.15: Sólido fatiado.

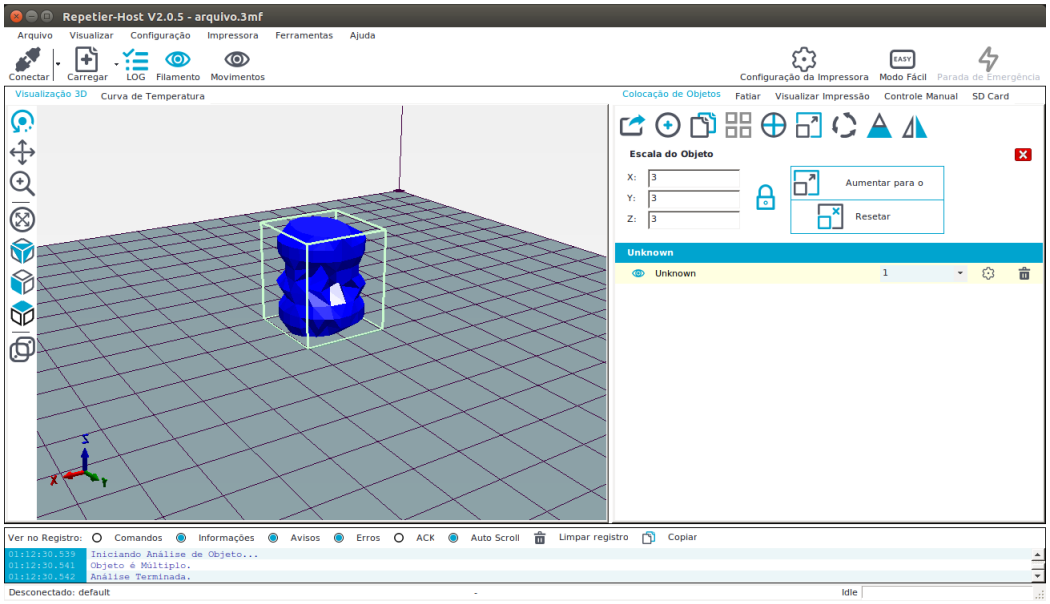


Figura 4.16: Sólido ampliado três vezes.

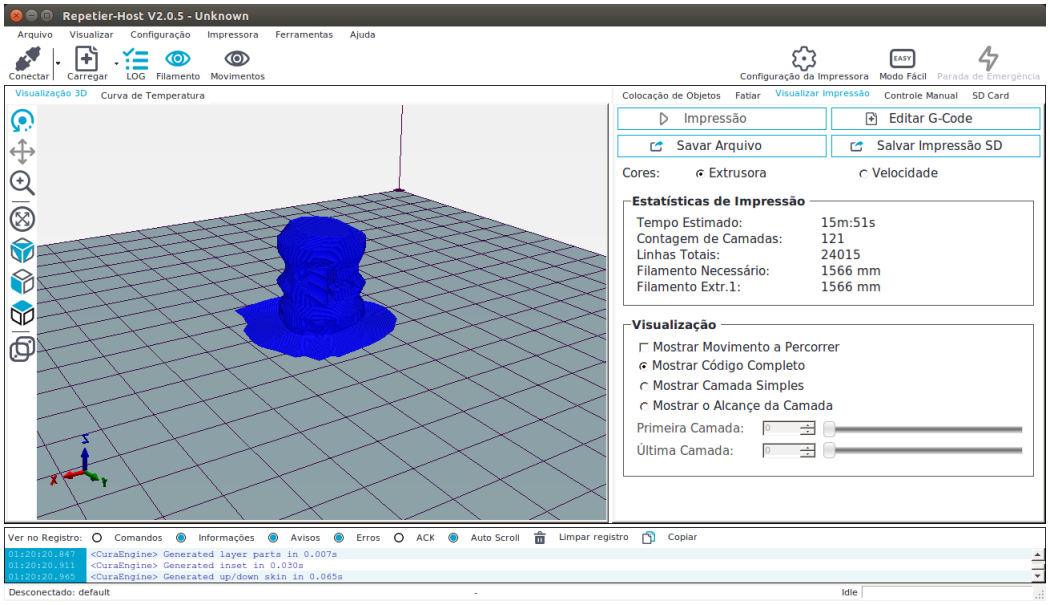


Figura 4.17: Sólido ampliado três vezes.

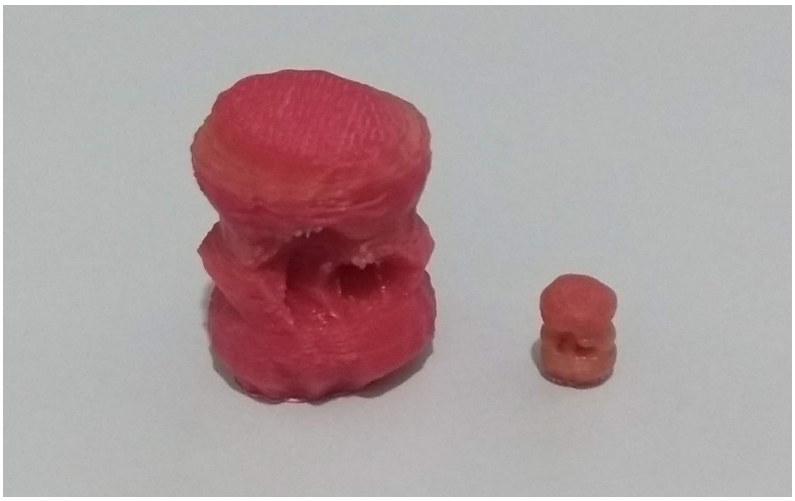


Figura 4.18: Sólidos impressos.

Capítulo 5

Conclusão

Este trabalho apresentou um modelo de reconstrução tridimensional baseado em superfícies de imagens transversais que funciona como uma ferramenta de suporte em tomada de decisões médicas.

Como principal objetivo, este trabalho tinha a função de fazer uma pesquisa e posterior desenvolvimento de um modelo de reconstrução tridimensional de um sólido que se mostrou possível de realizar de acordo com o esperado.

Pelo caráter acadêmico da pesquisa, não foram utilizadas imagens reais de exames médicos, apenas imagens sintéticas simulando fatias transversais de objetos.

As bibliotecas *OpenCV* e *OpenGL*, neste trabalho foram utilizadas como complemento ao desenvolvimento utilizando a linguagem de programação C, atenderam completamente a demanda de processamento de acordo com o esperado. A utilização da biblioteca *OpenCV* na fase inicial de pré-processamento foi fundamental por possui todas as funções descritas nesta fase e necessárias para manipulação das imagens obtidas. A biblioteca *OpenGL* foi utilizada com sucesso na segunda etapa do modelo, a etapa de reconstrução, onde era necessário a realização da manipulação dos pontos obtidos na primeira etapa do trabalho.

Os algoritmos discutidos nos capítulos iniciais deste trabalho, para triangulação e também para a criação de fecho convexo, são os algoritmos mais utilizados em pesquisas com temas parecidos, mostrando assim uma grande confiança nos algoritmos escolhidos.

O algoritmo de triangulação dos pontos das fatias foi utilizado sem a necessidade de realizar modificações que se adequassem ao tema de estudo. Mas pelo fato de aceitar somente contornos convexos para realizar a triangulação, foi necessário a utilização de um segundo algoritmo para tornar todos os contornos convexos.

O algoritmo de criação de fecho convexo precisou de algumas adaptações à realidade deste trabalho para ter um bom desempenho, pois este trabalho possui características que por definição seriam capazes de facilitar o processamento dos dados. Ao contrário da necessidade dos casos estudados que lidavam com pontos dispersos no espaço, este trabalho contava com uma estrutura de distribuição de pontos conhecida.

5.1 Trabalhos futuros

Devido à complexidade dos temas pesquisados neste trabalho, não foi possível abordar em sua completa extensão o tema desejado, por este motivo é necessário continuar o desenvolvimento com trabalhos futuros. Entre os quais podemos destacar:

- Expandir a implementação para realizar a triangulação em fatias que possuam mais de um contorno.
- Utilização de imagens médicas de exames reais.
- Aplicação em uma área específica da Medicina e Biologia.

Referências Bibliográficas

[1] SOUZA, Mauren Abreu de, CENTENO, Tania Mezzadri and PEDRINI, Hélio. Integrando reconstrução 3D de imagens tomográficas e prototipagem rápida para fabricação de modelos médicos *Rev. bras. eng. biomed, v. 19, n. 2*, 2003, p. 103-115.

[2] POLIZELLI JUNIOR, Valdecir. Métodos implícitos para a reconstrução de superfícies a partir de nuvens de pontos. *Tese de Doutorado. Universidade de São Paulo.*, 2008.

[3] MORAES, Thiago Franco de, AMORIM, Paulo Henrique Junqueira and MARTINS, Tatiana Al-Chueyr Pereira. Visualização Volumétrica de Imagens Médicas através de Raycasting. *III Seminário em TI do CTI*, 2010.

[4] AZEVEDO, Teresa Cristina de Sousa , TAVARES, João Manuel Ribeiro da Silva and VAZ, Mário Augusto Pires. Obtenção da forma 3D de objectos através de métodos volumétricos. *Congreso de Métodos Numéricos en Ingeniería*, (MetNum2009),2009 .

[5] TAKAGAKI, Luiz Koiti. Tecnologia de Impressão 3D. *RIT-Revista Inovação Tecnológica*v. 2, n. 2, 2013.

[6] JONES, Rhys et al. RepRap–the replicating rapid prototyper. *Robotica* v. 29, n. 01, p. 177-191, 2011.

[7] CENTENO, Tania M. et al. Reconstrução de Imagens Tomográficas Aplicada na Modelagem do Riser de uma Unidade FCC. *2º Congresso Brasileiro de P& D em Petróleo & Gás.* 2003

[8] FERNANDES, Nuno Teófilo Pereira de Araújo. Reconstrução de Superfícies usando um Scanner 3D de Luz Estruturada. , 2012.

[9] REBOUCAS FILHO, Pedro Pedrosa et al. Segmentação, Reconstrução e Visualização 3D dos Pulmões Utilizando Crescimento de Regiões 3D Aplicado em Imagens de Tomografia Computadorizada do Torax. *Simpósio Brasileiro de Automação Inteligente (SBAI)*, 2013.

[10] MORAES, Cícero André da Costa, DIAS, Paulo Eduardo Miamoto and MELANI, Rodolfo Francisco Haltenhoff. Demonstration of protocol for computer-aided forensic facial reconstruction with free software and photogrammetry. *Journal of Research in Dentistry*, v. 2, n. 1, p. p. 77-90, 2014.

[11] AMORIM, Paulo HJ et al. InVesalius: Software Livre de Imagens Médicas. *Centro de Tecnologia da Informação Renato Archer-CTI*, campinas/SP–2011-CSBC2011, 2011.

[12] TEIXEIRA, J. M., SIMOES, F., ROBERTO, R., TEICHRIEB, V. and KELNER, J.. Reconstrução 3d usando luzes estruturadas. *Virtual Reality (SVR)*,(2010, May). 2010 XII Symposium on (pp. 1-33).

[13] KAEHLER, A. and BRADSKI, G.. Learning Opencv First Edition, CA: O'Reilly Media, Inc.,September 2008.

[14] CONCI, A. and MONTEIRO, L. H.. Reconhecimento de placas de veículos por imagem. *III Congresso Nacional de Engenharia CONEM*, 2004.

[15] SHARIFI, Mohsen, FATHY, Mahmoud and MAHMOUDI, Maryam Tayefeh. A classified and comparative study of edge detection algorithms. *Information Technology: Coding and Computing*, Proceedings. International Conference on. IEEE, 2002. p. 117-120.

[16] SOUZA, M. A.. Reconstrução 3D de Imagens Tomográficas Aplicadas à Prototipagem Rápida. *Tese de Doutorado. Dissertação de mestrado*,CETEP-PR, 2002.

[17] MINGUETTI, Guilberto, FURTADO, Karine and AGOSTINI, LC de.. Tomografia computadorizada na agenesia do corpo caloso: achados em 27 casos. *Arq Neuropsiquiatr*, v. 56, n. 3-B, p. 601-4, 1998.

[18] BÁRTOLO, P.. Stereolithography: Materials, Processes and Applications Springer Science and Business Media, 18 de mar de 2011

[19] ANTON, H., BIVENS, I. and DAVIS, S. Cálculo - Volume II 10.ed. Bookman Editora, 1 de set de 2014

[20] FIGUEIREDO, L, CARVALHO, P Introdução à Geometria Computacional, IMPA, 18 Colóquio Brasileiro de Matemática, 1991, pags 111.

[21] HAGE, Maria Cristina Ferrarini Nunes Soares and IWASAKI, Masao Imagem por ressonância magnética: princípios básicos *Ciência Rural* , Santa Maria, v.39, n.4, p.1287-1295, jul, 2009.

[22] EKOULE, A. B., PEYRIN, F. C. and ODET, C. L. A Triangulation Algorithm from Arbitrary Shape Multiple Planar Contours *ACM Transactions on Graphics*, Vol. 10, No. 2, Pages 182-199, April 1991.

[23] CHRISTIANSEN, H. N. and SEDERBERG, T. W. Conversion of Complex Contour Line Definitions into Polygonal Element Mosaics *SIGGRAPH '78 Proceedings of the 5th annual conference on Computer graphics and interactive techniques* Pages 187-192, August 23 - 25, 1978.

[24] O'ROURKE, J.. Computational Geometry in C. Second Edition, Cambridge University Press, October 13, 1998