GERAÇÃO DE IMAGENS 3D DE BIOMICROSCOPIA ULTRASSÔNICA

Luisa Tinoco Carneiro

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Biomédica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Biomédica.

Orientador: João Carlos Machado

Rio de Janeiro

Novembro de 2012

GERAÇÃO DE IMAGENS 3D DE BIOMICROSCOPIA ULTRASSÔNICA

Luisa Tinoco Carneiro

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA BIOMÉDICA

Examinada por:

_____

Prof. João Carlos Machado, Ph. D.

_____

Prof. Alexandre Visintainer Pino, D.Sc.

_____

Prof. Sérgio Shiguemi Furuie, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

NOVEMBRO DE 2012

Carneiro, Luisa Tinoco

Geração de Imagens 3D de Biomicroscopia Ultrassônica / Luisa Tinoco Carneiro. – Rio de Janeiro: UFRJ/COPPE, 2012.

XV, 86 p.: il.; 29,7 cm.

Orientador: João Carlos Machado

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia Biomédica, 2012.

Referências Bibliográficas: p. 66-69.

1. Biomicroscopia Ultrassônica. 2. Phantom. 3 Imagem em 3D. I. Machado, João Carlos. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Biomédica. III. Título.

DEDICATÓRIA

Dedico esse trabalho a minha família.

AGRADECIMENTOS

A minha família, que sempre me incentivou.

Aos professores Claudio Esperança e Ricardo Marroquim do Programa de Engenharia de Sistemas e Computação/COPPE.

Ao meu orientador João Carlos Machado, pela paciência e por estar presente em todos os momentos dessa dissertação.

A todos os meus amigos do LUS e dos outros laboratórios.

Às agências CNPQ e CAPES pelo apoio financeiro.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

GERAÇÃO DE IMAGENS 3D DE BIOMICROSCOPIA ULTRASSÔNICA

Luisa Tinoco Carneiro

Novembro / 2012

Orientador: João Carlos Machado

Programa: Engenharia Biomédica

O ultrassom (US) é amplamente utilizado na medicina para auxiliar os médicos na avaliação de pacientes através da geração de imagens em duas dimensões (2D) de diversos órgãos. Apesar da frequente utilização das imagens de US em 2D, elas possuem algumas limitações quando, por exemplo, é necessária a visualização e a análise da anatomia de um órgão em três dimensões (3D). Tais limitações podem ser resolvidas ao se utilizar um sistema de US com geração de imagens em 3D, cujas vantagens incluem a possibilidade de visualizá-las em volume, de rotacionás-las e de disponibilizá-las em diversos ângulos e tendo-se uma visão que é impossível com imagens em 2D. Com essas imagens em 3D pode-se estimar, por exemplo, o volume de determinados órgãos, e isto pode ser importante em um diagnóstico preciso. Este trabalho relaciona-se com a implementação da geração de imagens 3D de biomicroscopia ultrassônica (BMU) a partir da múltipla aquisição de imagens 2D obtidas com a instrumentação de BMU de varredura setorial. O sistema BMU 3D foi testado na geração de imagens de *phantoms* de tecidos biológicos para BMU operando em 40 MHz. As formas e dimensões relativas das ponteiras utilizadas como molde são coincidentes com as dimensões imagens 3D das regiões dos *phantoms* moldadas pelas ponteiras, concluindo-se que o sistema de BMUs em conjunto com os programas de aquisição de imagens em 2D e de geração de imagens em 3D é capaz de gerar imagens em US em 3D de *phantoms* para BMU, com formas e dimensões relativas calibradas.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

GENERATION OF 3D ULTRASOUND BIOMICROSCOPIC IMAGES

Luisa Tinoco Carneiro

November /2012

Advisor: João Carlos Machado

Department: Biomedical Engineering

Ultrasound (US) is widely used in medicine to aid doctors in the evaluation of patients through two-dimensional (2D) images obtained from several organs. Although the 2D US images are often used, they have some limitations, for example, when visualization and analysis of an organ anatomy, in three dimensions (3D), is required. These limitations can be solved using a 3D US imaging system. The advantages include the ability to visualize the images in volume, to rotate them and to make them available at various angles with a visualization that is impossible with 2D images. The 3D images allow to estimate the volume of certain organs, which may be important for an accurate diagnosis. This work relates the implementation of 3D ultrasound biomicroscopic (UBM) image generation from the acquisition of multiple 2D images obtained with a sector-scan UBM imaging system. The 3D UBM system, operating at 40 MHz, was tested in the generation of images from UBM tissue phantoms. The shape and relative size of the tips used as template coincide with the dimensions of the respective 3D regions of the phantoms molded by the tips, concluding that the system of BMUs in conjunction with the programs of acquisition of 2D images and the 3D image generation is able to generate images of 3D US in phantoms for ultrasonic biomicroscopy, with shapes and relative dimensions calibrated.

Sumário

1 Introdução..... 1
2 Revisão de Literatura..... 3
2.1 Geração de Imagens em 2D..... 3
2.2 Algoritmos Reconstrução Volumétrica..... 4
2.2.1 VBM..... 5
2.2.2 VNN..... 5
2.2.3 VBMI..... 7
2.2.4 PBM..... 7
2.2.5 FBM..... 10
2.3 Geração de Imagens em 3D..... 11
2.3.1 Splatting..... 12
2.3.2 Shear Warp..... 14
2.3.3 Ray casting..... 15
2.3.4 Textura 3D..... 16
2.4 Biomicroscopia Ultrassônica..... 17
3 Materiais e Métodos..... 19
3.1 BMUs..... 19
3.2 Sistema de Posicionamento..... 21
3.3 Aquisição de Imagens em 2D..... 22
3.4 Geometria da Varredura Setorial do Feixe de Ultrassom..... 23
3.5 Conversão de Varredura..... 25
3.5.1 Formação da Imagem..... 26
3.5.1.1 Matriz Final..... 26
3.5.1.2 Translação..... 27
3.5.1.3 Cálculo do Deslocamento..... 29
3.5.1.4 Offset..... 31
3.5.1.5 Cálculo do Offset..... 32
3.5.2 Interpolação..... 33
3.5.2.1 Interpolação Bilinear..... 34
3.5.2.2 DLL..... 34
3.5.2.3 Quadros de Imagem..... 36
3.6 Construção das Imagens em 3D..... 37

3.7 Construção de Phantoms para BMU..... 40
4 Resultados..... 45
4.1 Phantom 1..... 45
4.2 Phantom 2..... 48
4.3 Phantom 3..... 53
5 Discussão..... 54
6 Conclusão..... 65
7 Referências Bibliográficas..... 66
8 Anexos..... 70
8.1 Anexo I..... 70
8.2 Anexo II..... 74

LISTA DE FIGURAS

Figura 2.1: Exemplos de formas de varredura do feixe de ultrassom..... 4
Figura 2.2: Ilustração do método baseado em *voxel* do vizinho mais próximo..... 6
Figura 2.3: Ilustração do método baseado em *pixel* com passo de distribuição..... 8
Figura 2.4: Ilustração do método baseado em *pixel* com passo de preenchimento de espaço..... 9
Figura 2.5: Ilustração da interpolação funcional, visualizado ao longo de 1 dimensão 10
Figura 2.6: Técnica de *splatting*..... 13
Figura 2.7: Algoritmo de *shear-warp*..... 14
Figura 2.8: Técnica de *ray casting*..... 15
Figura 2.9: Visualização direta por mapeamento de texturas..... 16
Figura 2.10: Fatias da visualização direta por mapeamento de textura perpendiculres à direção de observação..... 17
Figura 3.1: Diagrama de blocos do sistema de BMUs..... 19
Figura 3.2: Equipamento controlador de movimento..... 21
Figura 3.3: Sistema de posicionamento em vista superior..... 22
Figura 3.4: Diagrama de blocos com a sequência de operações para a obtenção das imagens 2D..... 23
Figura 3.5: Ilustração da disposição dos dados amostrados dos sinais de eco que compõem a imagem de BMUs..... 24
Figura 3.6: Representação da Matriz Final de dados..... 26
Figura 3.7: Representação da geometria da varredura..... 27
Figura 3.8: Diagrama ilustrando a translação da Matriz Final para gerar a Matriz Intermediária (M_I)..... 28
Figura 3.9: Construção da Matriz Fin Ilustração do cálculo do deslocamento para se transformar a Matriz Final a partir da Matriz Intermediária 29
Figura 3.10: Ilustração do triângulo utilizado para cálculo da altura..... 30
Figura 3.11: Referência angular utilizada na conversão das coordenadas retangulares para polares..... 32
Figura 3.12: Mapeamento dos *pixels*..... 34
Figura 3.13: Região da imagem preenchida durante a execução da DLL..... 36
Figura 3.14: Cortes de imagem 2D para formar o cubo..... 38
Figura 3.15: Formação da imagem 3D a partir dos planos de corte em 2D..... 39

Figura 3.16: Diagrama de blocos do programa de geração de imagens 3D..... 40
Figura 3.17: Ponteira de micropipeta..... 41
Figura 3.18: Ponteira de micropipeta..... 41
Figura 3.19: Foto do *phantom* 1..... 43
Figura 3.20: Foto do *phantom* 2..... 43
Figura 3.21: Foto do *phantom* 3..... 44
Figura 4.1: Imagem de BMUs da seção transversal do *phantom* 1..... 45
Figura 4.2: Imagem de US do *phantom* 1 em 3D com vista lateral..... 46
Figura 4.3: Imagem de US do *phantom* 1 em 3D com vista frontal..... 47
Figura 4.4: Imagem de US do *phantom* 1 em 3D com vista em diagonal..... 48
Figura 4.5: Imagem 2D do *phantom* 2..... 49
Figura 4.6: Imagem 3D de US do *phantom* 2 com vista lateral..... 50
Figura 4.7: Imagem 3D de US do *phantom* 2 com vista frontal..... 51
Figura 4.8: Imagem 3D de US do *phantom* 2 com vista lateral..... 52
Figura 4.9: Imagem 3D de US do *phantom* 2 com vista lateral..... 53
Figura 4.10: Imagem de BMUs da seção transversal do *phantom* 3..... 54
Figura 4.11: Imagem 3D de US do *phantom* 3 com vista lateral..... 55
Figura 4.12: Imagem 3D de US do *phantom* 3 com vista frontal..... 56
Figura 4.13: Imagem 3D de US do *phantom* 3 com vista diagonal..... 57
Figura 5.1: Imagem do *phantom* 1 com sobreposição do contorno da micropipeta..... 60
Figura 5.2: Imagem do *phantom* 2 com sobreposição do contorno da micropipeta..... 61
Figura 8.1: Tela de configuração dos parâmetros iniciais do programa em LabVIEW usado no BMUs..... 70
Figura 8.2: Display de visualização da imagem..... 71
Figura 8.3: Controles de contraste e ruído da imagem de BMUs..... 71
Figura 8.4: Tela de configuração de alguns parâmetros e controle do sistema de posicionamento..... 73
Figura 8.5: Janela de digitação da largura, normalizada, para a imagem 3D..... 74
Figura 8.6: Janela de digitação da altura, normalizada, para a imagem 3D..... 74
Figura 8.7: Janela de digitação da profundidade, normalizada, para a imagem 3D.... 74
Figura 8.8: Janela para a seleção do diretório..... 75
Figura 8.9: Janela de digitação do número da última imagem..... 76
Figura 8.10: Janela de digitação do nome do arquivo contendo as imagens 2D..... 76

LISTA DE SÍMBOLOS

c	velocidade do ultrassom
E	Matriz de Escalonamento
fa	frequência de amostragem
h	altura do triângulo isósceles formado pelos vértices O, P e Q, na Figura 3.12
M_F	Matriz Final
M_I	Matriz Intermediária
M_I’	segunda Matriz Intermediária
M_O	Matriz de dados amostrados
maxPontos	pontos máximo do quadro de imagem
minPontos	pontos mínimo do quadro de imagem
N_{RF}	número de pontos adquiridos após a digitalização do sinal de eco ao longo de cada feixe
Offset	quantidade de linhas radiais dentro de um intervalo
pinicio	ponto inicial
posConv	matriz de transformação
$r_{m,n}$	coordenada retangular covertida para o raio da coordenada polar
$\theta_{m,n}$	coordenada retangular covertida para o ângulo da coordenada polar
T	Matriz de Translação
vetorPorSetor	quantidade de linhas radiais por unidade de ângulo
X_n	coordenada de cada pixel de MI
X_{coluna}	número de colunas de MF

Y_m	coordenada de cada pixel de MI
Y_{linha}	número de linhas de MF
Δlinha	deslocamento vertical da translação
Δcoluna	deslocamento horizontal da translação
θ_{setor}	a região setorial varrida pelo feixe de ultrassom
θ_{offset}	ângulo correspondente ao posicionamento mais à esquerda do feixe de ultrassom para a varredura setorial

LISTA DE SIGLAS

BMU	Biomicroscopia Ultrassônica
BMUs	Biomicroscopia Ultrassônica Setorial
CC	Corrente Contínua
df	Distância Focal
DLL	*Dinamic Link-Library*
DOF	*Depth of Field*
DS	*Distribution Step*
FBM	*Function-Based Methods*
GPU	*Graphics Processing Unit*
HFS	*Hole-Filling Step*
JPEG	*Joint Photographic Experts Group*
PBM	*Pixel-Based Methods*
PCI	*Peripheral Component Interconnect*
PCIexpress	*Peripheral Component Interconnect express*
PNN	*Pixel Nearest Neighbor*
PVDF	Difluoreto de Polivinilideno
RAM	*Random Access Memory*
RF	Radio Frequência
RGB	(*Red*, *Green*, *Blue*)
RM	Ressonância Magnética

UBM	*Ultrasound Biomicroscopy*
US	Ultrassom
VBM	*Voxel-Based Methods*
VBMI	*Voxel-Based Methods with Interpolation*
VNN	*Voxel Nearest Neighbor*
1D	uma dimensão
2D	duas dimensões
3D	três dimensões

1 INTRODUÇÃO

Uma onda de som ou ultrassom (US) é um distúrbio mecânico de um meio que varia com o tempo e com a posição e que se propaga com uma velocidade que é uma característica do próprio meio. Uma onda de som com frequência acima de 20 kHz é chamada de ultrassom e sua velocidade, para tecidos biológicos moles, é de aproximadamente 1540 m/s (FISH, 1990).

O US é amplamente utilizado na medicina para auxiliar os médicos no diagnóstico de pacientes, através da geração de imagens em duas dimensões (2D) de diversos órgãos. Suas principais vantagens são o baixo custo, a alta eficácia e o escaneamento em tempo real (CHANG *et al.*, 2003; FRY *et al.*, 2003; HUANG e ZHENG, 2007). A geração de imagens de US em 2D baseia-se na utilização de um transdutor de US que emite uma onda de US em um tecido e recebe os sinais de eco provenientes das diversas regiões por onde a onda emitida se propaga. A magnitude e o tempo de chegada dos sinais de eco são utilizados para gerar imagens em 2D do tipo modo-B, em escala de cinza, de tecidos e órgãos (WEBB, 1988).

Apesar das imagens de US em 2D serem frequentemente utilizadas, elas possuem algumas limitações quando, por exemplo, é necessária a visualização e a análise da anatomia de um órgão em três dimensões (CHANG *et al.*, 2003). Tais limitações podem ser resolvidas ao se utilizar um sistema de US com geração de imagens em 3D, cujas vantagens incluem ver a imagem 3D, a possibilidade de rotacionás-la, visualizás-la e manipulá-la em diversos ângulos, tendo-se uma visão que é impossível com imagens em 2D (SOLBERG *et al.*, 2007). Com essas imagens 3D pode-se estimar, por exemplo o volume de determinados órgãos, e isto pode ser importante em um diagnóstico (COBBOLD, 2007).

A biomicroscopia ultrassônica (BMU) é uma técnica de geração de imagens de ultrassom em alta frequência (acima de 20 MHz). Com ela podem ser realizadas biópsias virtuais de tecidos saudáveis ou não, *in vivo*, ao se analisarem as imagens geradas com resolução próxima da obtida pela microscopia óptica panorâmica. A BMU é empregada na geração de imagens da parte frontal do olho, de tecido cutâneo e da parede arterial, entre outros (FOSTER *et al.*, 1999). Outra aplicação importante da BMU refere-se à obtenção de imagens de pequenos animais (rato ou camundongo), os

quais são preparados como modelos de diversas doenças e alterações anatômicas sobre as quais se realizam investigações através das imagens (CHIOU *et al.*, 2000, TURNBULL e FOSTER, 2002, JOLLY *et al.*, 2005, PEIXINHO *et al.*, 2011).

A realização da presente dissertação de mestrado tem uma aplicação direta com outro trabalho de pesquisa em andamento, cujos objetivos de longo prazo direcionam-se para testes de uma terapia, baseada em células tronco, para a regeneração de músculo esquelético lesionado do membro posterior de ratos. Um passo anterior à realização da proposta da terapia mencionada consiste no estabelecimento de um procedimento controlado e reprodutível de geração de lesão muscular para posterior tratamento terapêutico. Além de se estabelecer um procedimento confiável de geração de lesão muscular, é também necessário criar uma metodologia de sua avaliação. Neste sentido, o estudo proposto visa utilizar a obtenção de imagens de BMU em 3D que serão usadas na avaliação do protocolo de lesão muscular. Para isto, a metodologia a ser seguida baseia-se na obtenção de imagens de BMU bi-dimensionais do músculo afetado pela lesão para formar uma imagem em três dimensões (3D) que será usada na quantificação volumétrica da região lesionada. Já as imagens 2D permitem a determinação das características de arquitetura muscular (ângulo de penação, comprimento da fibra e espessura muscular) do músculo de ratos submetidos a um processo de lesão por incisão.

Esta dissertação refere-se à implementação de uma metodologia para a construção de imagens 3D de BMU a partir da múltipla aquisição de imagens 2D obtidas com a instrumentação de BMU de varredura setorial.

2 REVISÃO DE LITERATURA

2.1 GERAÇÃO DE IMAGENS EM 2D

A ideia de utilizar uma sequência de imagens em 2D para gerar uma imagem em 3D surgiu por volta do ano de 1956 (COBBOLD, 2007). O primeiro exemplo de construção de uma imagem de US em 3D ocorreu em 1961 (BAUM e GREENWOOD, 1961), para o diagnóstico em oftalmologia. Desde então, diversas tentativas de desenvolver sistemas que reproduzissem imagens em 3D de US foram realizadas.

Atualmente, existem dois tipos de sistemas de varredura do feixe de ultrassom para a geração de imagens em 3D: a varredura eletrônica e a mecânica.

No sistema de varredura eletrônica do feixe utiliza-se um transdutor de ultrassom *phased array* com os elementos piezoelétricos distribuídos numa matriz 2D (COBBOLD, 2007). Com isso, o feixe de ultrassom pode varrer todo um semi-espaço (ângulo sólido de 2π esferorradianos) à frente da superfície do transdutor. Isto permite a obtenção de sinais de eco oriundos de um volume.

Já o sistema de varredura mecânica pode ser motorizado ou do tipo *freehand*. No primeiro caso há um acionamento motorizado de deslocamento mecânico do transdutor de forma a permitir a obtenção de múltiplos planos de imagem. O transdutor pode ser transladado, rotacionado ou balançado, conforme visto na Figura 2.1 (COBBOLD, 2007; FENSTER *et al.*, 2000).

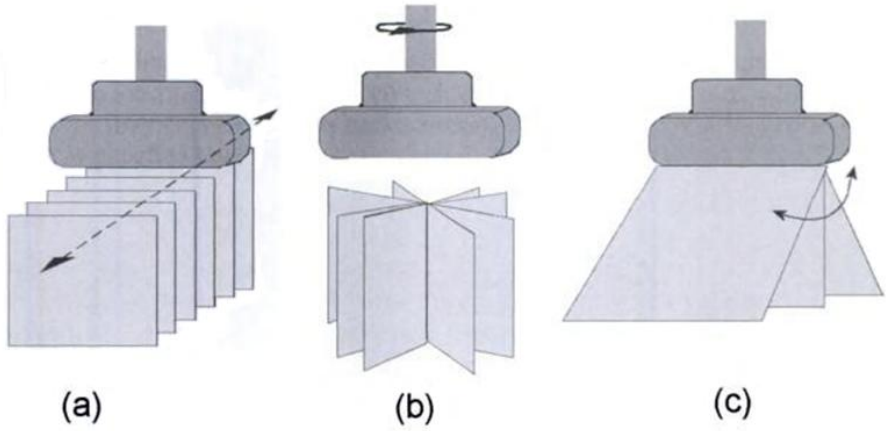

Figura 2.1: Exemplos de sistemas de varredura motorizado. (a) Varredura linear, (b) Movimento rotacional, (c) Movimento de angulação (*Rock*) (escaneamento em inclinação) (Com permissão para reprodução, COBBOLD, 2007)

No sistema do tipo *freehand*, o operador do sistema de US realiza a varredura do feixe manualmente e múltiplos planos de imagem são adquiridos conjuntamente. O equipamento de US com varredura do tipo *freehand* inclui um sistema de detecção da posição do transdutor referente a um sistema de coordenadas espaciais.

O sistema motorizado oferece, como vantagem, a possibilidade da aquisição de um determinado número de quadros de imagens 2D com espaçamentos entre si pré-determinados, o que resulta numa imagem em 3D mais uniforme. Por outro lado, o sistema de *freehand* permite a obtenção de imagens do tipo panorâmicas ou de regiões do corpo que exigem uma varredura do feixe ao longo de uma superfície de contorno (côncavo, convexo ou fechado), como a cocha ou o cólon.

Independentemente do sistema de varredura ser eletrônica ou mecânica, a geração de imagens em 3D contempla o uso de algoritmos computacionais para a conversão de varredura entre os sistemas de coordenadas adotados para a aquisição dos sinais de eco (dados de entrada) e para a exibição da imagem (imagem alvo), normalmente um sistema de coordenadas retangulares.

2.2 ALGORITMOS PARA RECONSTRUÇÃO VOLUMÉTRICA

Alguns algoritmos para reconstrução volumétrica baseiam-se em diferentes métodos comentados a seguir. Eles foram divididos em três grupos, de acordo com a

sua implementação: os métodos baseados em *voxel*, VBM - *Voxel-Based Methods*, os métodos baseados em *pixel*, PBM - *Pixel-Based Methods*, e os Métodos baseados em função, FBM - *Function-Based Methods,* (SOLBERG *et al.*, 2007). Onde *pixel* é o elemento de imagem (*picture cell*), o menor elemento num dispositivo de exibição como, por exemplo, um monitor, ao qual é possivel se atribuir uma cor e *voxel* é o elemento de volume (*volume cell*), o menor elemento numa grade retangular no espaço tridimensional. É análogo a um pixel, para ambiente tridimensional

2.2.1 VBM

Os métodos VBM percorrem cada *voxel* na grade do *voxel* alvo e reúnem informação das imagens em 2D de entrada para serem alocadas no *voxel*. Nos diferentes métodos de VBM, um ou vários *pixels* podem contribuir para o valor de cada *voxel* (SOLBERG *et al.*, 2007).

2.2.1.1 VNN

Neste algoritmo, para cada *voxel* é atribuído o valor de um *pixel*. Alguns métodos usam apenas um *pixel* para decidir o valor de um *voxel* e um método geralmente implementado dessa maneira é o do *voxel* do vizinho mais próximo (VNN, *Voxel Nearest Neighbor*). O VNN percorre cada *voxel* da imagem alvo e atribui o valor do *pixel* da imagem mais próximo. Segmentos de reta perpendiculares à direção dos feixes usados na geração da imagem 2D e que passa pelo centro do *voxel* são determinados e o valor do *pixel* mais próximo é usado para o *voxel.* Este processo pode ser visto na Figura 2.2, na qual os dados da imagem de entrada são ilustrados ao longo das linhas representativas das direções dos feixes de ultrassom durante a varredura dos mesmos para formar uma imagem 2D e os pontos nas linhas ilustram os centros dos *pixels*. A grade em 2D marca o centro dos *voxels* na grade de *voxels* 3D (SHEREBRIN *et al.*, 1996 apud SOLBERG et al., 2007).

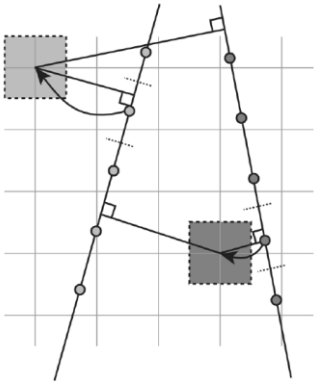

Figura 2.2: Algoritmo de Voxel do Vizinho mais Próximo: dados da imagem de entrada são ilustrados ao longo das linhas representativas das direções dos feixes de ultrassom durante a varredura dos mesmos para formar uma imagem 2D e os pontos nas linhas ilustram os centros dos *pixels*. A grade em 2D marca o centro dos *voxels* na grade de *voxels* 3D (Com permissão para reprodução, SOLBERG et al., 2007).

A seguir observa-se um algoritmo para uma implementação simples do VNN (MCCANN *et al.*, 1988, SHEREBRIN *et al.*, 1996 apud SOLBERG *et al.*, 2007)

- Para cada *voxel*
  - Encontre os *pixels* mais próximos:
    - Calcule as distâncias entre o centro do *voxel* e as linhas modo-A da imagem de entrada contendo os *pixels* mais próximos;
    - A normal com menor distância fornece a linha modo-A da imagem de entrada mais próxima;
    - O *pixel* da linha modo-A determinada e mais próximo da normal é o *pixel* mais próximo do *voxel*;
  - Insira o valor do *pixel* mais próximo no *voxel*.

A seguir observa-se um algoritmo para uma implementação rápida do VNN.Assume-se que na geração de imagem em ultrassonografia, uma varredura do feixe com movimento de translação (ou angulação) é realizada em apenas uma direção e não para trás e para frente.

- Para cada *voxel*

- Encontre *pixel* mais próximo:
  - Calcule as distâncias entre o centro do *voxel* e as linhas modo-A da imagem de entrada antes e depois do *voxel*;
  - A menor distância fornece a imagem mais próxima;
  - O *pixel* da linha de modo-A mais próximo da normal é o *pixel* mais próximo do *voxel*;
- Insira o valor o *pixel* mais próximo no *voxel*.

2.2.1.2 VBMI

O método baseado em *voxel* com interpolação (VBMI) usa uma interpolação entre diversos valores de *pixel* de entrada para decidir um valor de *voxel*. A seguir é apresentado um algoritmo para o método VBMI (TROBAUGH *et al.*, 1994, apud SOLBERG *et al.*, 2007)

- Para cada *voxel*:
  - Encontre duas linhas modo-A adjacentes ao centro do *voxel*;
  - Calcule a distância entre o centro do *voxel* e as duas linha modo-A;
  - Para cada linha modo-A, faça uma interpolação entre os quatro *pixels* mais próximos do ponto de contato da normal;
  - A seguir interpole os dois resultados resultante das duas interpolações anteriores.

2.2.2 PBM

Para o método PBM, o valor de cada *pixel* nas imagens de entrada é atribuído para um ou vários *voxels*. De uma maneira geral, o PBM pode consistir de dois passos: o passo de distribuição (DS, *Distribution Step*) e o passo de preenchimento de espaço

(HFS, *Hole-Filling Step*). No DS, os *pixels* de entrada são percorridos e o valor de cada *pixel* é atribuído a um ou vários *voxels*, podendo-se levar em conta um peso para a contribuição do *pixel*. No HFS, os *voxels* são atravessados e aqueles vazios são preenchidos. A maioria dos métodos HFS tem um limite de quão distante os *voxels* a serem preenchidos devem estar dos *voxels* com valores já atribuídos. Se os *pixels* das imagens de entrada forem muito distantes ou o limite de preenchimento de espaços na imagem alvo for muito pequeno, poderá haver *voxels* vazios no volume construído. Caso 2 *pixels* pertençam a um *voxel*, é calculada a média desses *pixels* para ser atribuída ao *voxel*.

No DS, o preenchimento do *voxel* utiliza o *pixel* do vizinho mais próximo (PNN, *Pixel Nearest Neighbor*). Nesse caso, cada valor de *pixel* é atribuído ao *voxel* mais próximo como mostrado na Figura 2.3, onde as imagens de entrada são ilustradas por linhas, e os pontos nas linhas ilustram o centro dos *pixels*. As grades em 2D marcam o centro dos *voxels*.

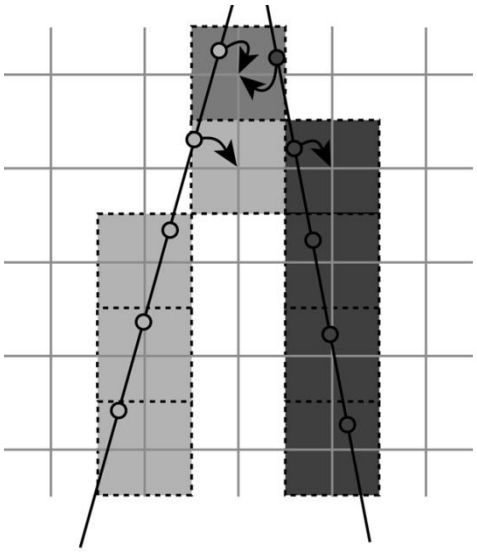

Figura 2.3: Método baseado em *pixel* com a excução do passo de distribuição, onde para cada *voxel* é atribuído o valor do *pixel* de interseção da linha com o *voxel*. As imagens de entrada são ilustradas por linhas, e os pontos nas linhas ilustram o centro dos *pixels* (Com permissão para reprodução, SOLBERG et al., 2007).

A seguir é apresentado um algoritmo para o método PNN com preenchimento de cada *voxel* (HOTTIER *et al.*, 1990, NELSON *et al.*, 1997, ROHLING *et al.*, 1999, apud SOLBERG *et al.*, 2007). Neste algoritmo ocorre o preenchimento por PNN com média.

- Para cada *pixel*:
  - Encontrar o *voxel* ao qual o *pixel* pertence
  - Se *voxel* não tiver valor
    - Valor de *Voxel* = Valor de *Pixel*
    - Contador de *Pixel* = 1
  - Se o voxel ao qual o *pixel* pertence já tiver um valor
    - Atribuir ao *voxel* a média de todos os *pixels* pertencentes:

$$Valor\ de\ Voxel = \frac{Valor\ de\ Voxel \times Contador\ de\ Pixel + Valor\ de\ Pixel}{Contador\ de\ Pixel + 1}$$

    - Contador de *Pixel* = Contador de *Pixel* + 1

Após a realização do DS podem existir, espaços vazios na matriz do *voxel*. No HFS, o volume preenchido na sequência do DS é percorrido e em cada *voxel* vazio tenta-se preenchê-lo com a informação dos *voxels* próximos já preenchidos. Este valor pode ser uma média ou mediana, ou uma interpolação entre dois, ou mais, *voxels* não zero mais próximos, como mostrado na Figura 2.4.

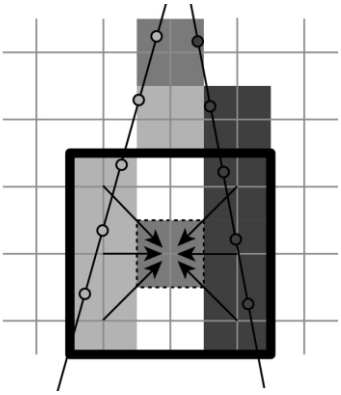

Figura 2.4: Método baseado em *pixel* com a execução do passo de preenchimento de espaço, onde a cada *voxel* vazio é atribuído um valor baseado no *voxel* já preenchido mais próximo. As imagens de entrada são ilustradas por linhas, e os pontos nas linhas ilustram o centro dos *pixels* (Permissão solicitada para reprodução, SOLBERG et al., 2007).

2.2.3 FBM

Os métodos FBMs utilizam uma função particular, como por exemplo um polinômio, e determinam os coeficientes do mesmo de forma que seus valores coincidam com os dos *pixels* da imagem de entrada, para uma dada direção definida por um dos eixos de sistema de coordenadas dos *voxels*. Posteriormente, a função é utilizada para gerar uma matriz com os valores atribuídos a cada *voxel*. Na Figura 2.5 onde as imagens de entrada são ilustradas por linhas, sobre as quais os pontos ilustram o centro do *pixel*, é ilustrado esse método. A grade em 2D marca o centro dos *voxels* na grade de *voxel* em 3D. O plano horizontal contendo as linhas verticais da grade de *voxels* contém, na forma de exemplo, uma determinada função particular obtida com base nos dados de entrada e os 'X's na curva mostram os valores de dados resultantes obtidos em intervalos regulares correspondendo à grade alvo, ao avaliar a função particular em posições separadas por intervalos regulares correspondendo à grade alvo.

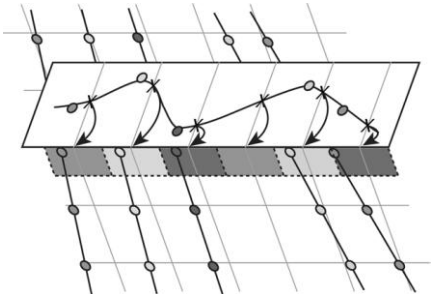

Figura 2.5: Método baseado em função que interpola um polinômio e esse polinômio passa pelos valores dos *pixels* de um determinado eixo e assim se consegue determinar o valor dos *voxels*, As imagens de entrada são ilustradas por linhas, e os pontos nas linhas ilustram o centro dos *pixels* (Permissão solicitada para reprodução, SOLBERG et al., 2007).

A seguir é apresentado um algoritmo para o método FBM (ROHLING *et al.*, 1999, apud SOLBERG *et al.*, 2007):

- Divida a matriz *voxel* em pequenos segmentos retangulares (não necessariamente do mesmo tamanho, mas com número interno suficiente de pontos de dados),
- Defina uma janela em torno de cada segmento
  - Para cada janela
    - Aumente a janela em todas as direções separadamente, de modo que cada crescimento da direção mantenha um número de pontos de dados vizinhos
    - Se o segmento contém muitos pontos de dados
      - Divida-o em segmentos menores
    - Use todos os pontos dentro da janela para calcular a função de base radial para o segmento
    - Avalie a função em intervalos regulares para obter os dados do *voxel*.

2.3 GERAÇÃO DE IMAGENS EM 3D

A visualização de imagens 3D envolve um conjunto de técnicas que permite observar dados volumétricos sem a necessidade de uma representação intermediária. O volume visualizado é representado através de *voxels* que são projetados diretamente em *pixels* (2D) e armazenados como uma imagem. Para a realização deste processo, é feito um mapeamento dos valores dos *voxels* em valores de cor e opacidade, mapeamento este chamado de função de transferência (KINDLMANN e DURKIN, 1998). O papel da função de transferência é o de dar ênfase nas características dos dados ao mapear valores e medidas de outros dados para as propriedades ópticas. Projetar uma função de transferência é um procedimento difícil que requer uma compreensão significativa do conjunto de dados subjacente. Algumas informações são fornecidas pelo histograma de valores de dados, indicando as faixas de valores que devem ser enfatizadas. A interface

com o usuário é um componente importante do procedimento de projeto interativo. Tipicamente, a interface é constituída por um editor de curva 1D para especificar funções de transferência através de um conjunto de pontos de controle.

As principais técnicas de visualização direta de volumes são as de *ray casting*, *splatting*, *shear-warp* e as baseadas em textura, esta última utilizada nesta dissertação. Tais técnica e algoritmos estão descritos a seguir.

2.3.1 SPLATTING

O algoritmo *Splatting* (WESTOVER, 1990) mapeia cada *voxel* do volume no plano da imagem. Esse mapeamento é geralmente realizado a partir dos *voxels* mais próximos do observador para os mais distantes. Após o mapeamento no plano de imagem de cada *voxel*, através de um processo de acumulação, sua contribuição é adicionada à formação da imagem.

O primeiro passo do algoritmo é determinar a ordem em que o volume será percorrido. Este passo é essencial, pois a ordem correta de projeção permite acumular adequadamente as opacidades dos *voxels*. Para a correta projeção, é necessário escolher os dois eixos do volume mais paralelos ao plano da imagem. O plano formado por esses dois eixos será projetado no plano de projeção e terá suas contribuições acumuladas em um *buffer* denominado *sheet*. A projeção é realizada quadro a quadro, ou seja, todos os *voxels* de um determinado quadro são projetados no plano de projeção antes que o próximo quadro seja processado. O valor de densidade de cada *voxel* é classificado de acordo com as funções de transferência de cor e opacidade.

A reconstrução é o segundo passo, e é a parte mais importante do algoritmo, onde é calculada a contribuição de cada *voxel* no plano da imagem. Neste passo, o algoritmo reconstrói um sinal contínuo a partir de um conjunto discreto de dados para gerar a imagem, de modo a re-amostrar o sinal na resolução desejada. Para a realização desta etapa, um filtro de reconstrução (*kernel*) é utilizado para calcular a extensão da projeção do *voxel* sobre o plano da imagem. A projeção do *kernel* é chamada de *footprint* e, no caso de projeções ortográficas (ou paralelas), o *footprint* é o mesmo para todos os *voxels*. O *footprint* é uma tabela que determina como o *voxel* será “jogado” no

plano da imagem, como mostrado na Figura 2.6. A projeção é feita “jogando” os *voxels* do volume no plano de projeção de trás para frente. Isso significa que a contribuição do *voxel* é maior no centro de projeção sobre o plano da imagem (*pixel* central) e menor nos *pixels* mais afastados, como pode ser observado na Figura 2.6, onde o *pixel* central possui uma opacidade maior que os *pixels* mais afastados. Quando um determinado *pixel* do plano de projeção acumula opacidade próxima de 1, este *pixel* não precisa mais ser processado.

O passo seguinte é o processo chamado de visibilidade, que recebe a cor e a opacidade do *voxel*. Este processo também avalia os valores da cor e da opacidade para gerar a contribuição em todos os *pixels* que estão sob a extensão do *footprint*. Os valores avaliados são armazenados no *buffer* de acumulação, e utilizam o esquema de acumulação apropriado à ordem em que o volume é lido e levam em consideração a atenuação provocada pela aplicação do *footprint*.

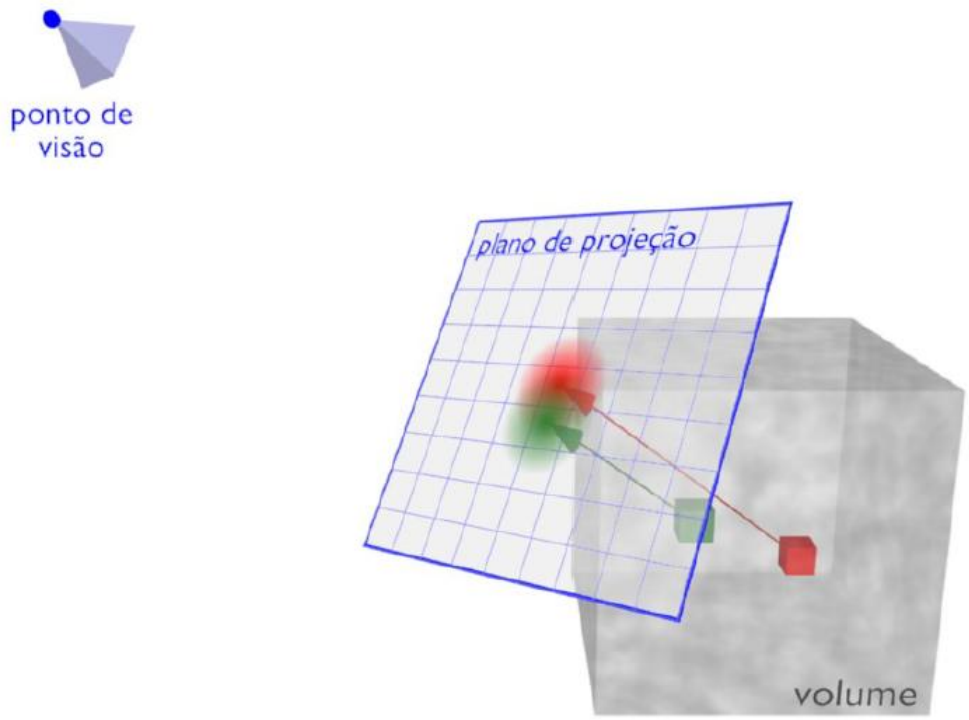

Figura 2.6: Técnica de splatting, onde a projeção é feita jogando os voxels do volume no plano de projeção de trás para frente (Com permissão para reprodução, HUFF, 2006).

2.3.2 SHEAR WARP

A ideia principal do algoritmo é a decomposição da transformação de projeção em transformação de cisalhamento (*shear*) e dobra (*warp*) (LACROUTE e LEVOY, 1994).

Conforme mostrado na Figura 2.7, o cisalhamento é usado para transformar cada fatia do volume de dados para um sistema de coordenadas intermediário. Assim, as fatias do volume se tornarão paralelas ao plano da imagem. Este processo facilita a projeção das fatias, pois os dados volumétricos são acessados na ordem de armazenamento, gerando uma imagem intermediária distorcida.

As dimensões do volume interferem na resolução da imagem intermediária. Essa interferência se deve ao fato da área total ocupada pela projeção das fatias cisalhadas estarem contida nesta imagem.

A imagem final é obtida através da aplicação da transformação de dobra (*warp*) na imagem intermediária distorcida, como mostrado na Figura 2.7. Essa transformação é realizada em 2D e restaura as reais dimensões da imagem que será visualizada pelo usuário.

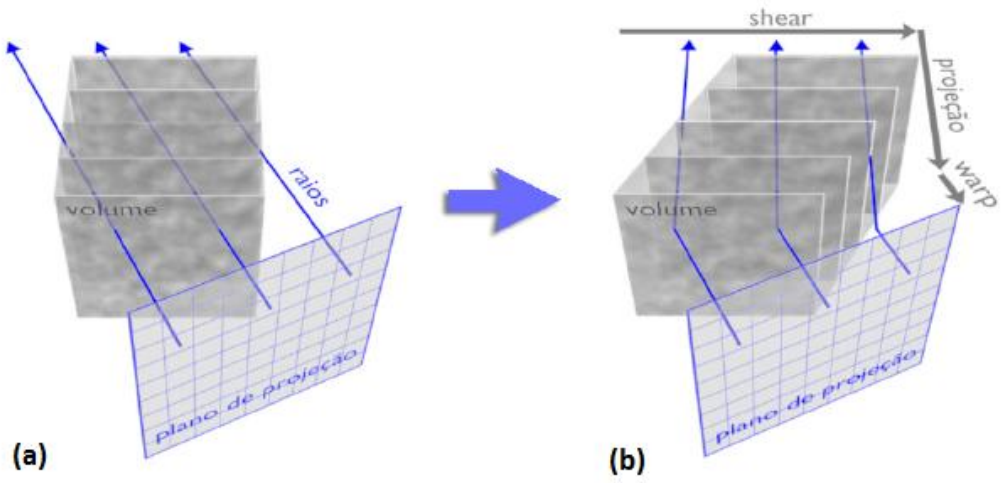

Figura 2.7: Algoritmo de *shear-warp* (a) Modo de visualização do plano de projeção sem o algoritmo de *shear-warp* (b) Algoritmo de *shar-warp* sendo aplicado ao volume (Com permissão para reprodução, HUFF, 2006).

2.3.3 RAY CASTING

O método *Ray Casting* (LEVOY, 1988) é um algoritmo de renderização volumétrica muito utilizado quando se precisa obter imagens de alta qualidade. O algoritmo lança um raio através de cada *pixel*, que atravessa o volume de dados, o que pode ser observado na Figura 2.8. A cor e a opacidade encontradas nas partes do volume interceptadas pelo raio são acumuladas para se determinar a cor final do *pixel*.

O primeiro passo do algoritmo é encontrar a primeira face interceptada pelo raio lançado através do *pixel*. Esta face é chamada de face externa e, a partir dela, são obtidos os valores iniciais de cor e opacidade. Depois de encontrar a primeira face interceptada pelo raio, as faces seguintes também interceptadas pelo raio, tem acumulados seus valores de cor e opacidade por meio de uma função de transferência até que o raio saia inteiramente do volume ou que o valor da opacidade acumulada atinja o valor máximo de 1. O resultado final da composição dos valores de cor e opacidade das faces por onde o raio passa é a cor do *pixel*.

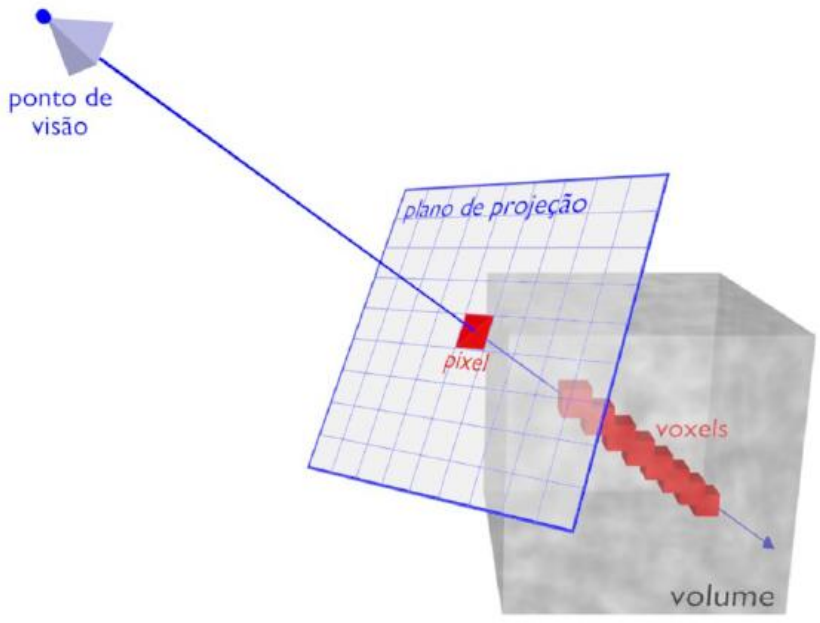

Figura 2.8: Técnica de *ray casting*, onde a projeção é feita disparando-se raios do ponto de visão através de cada *pixel* da imagem 2D (plano de projeção) e passando pelo volume (Com permissão para reprodução, HUFF, 2006).

2.3.4 TEXTURA 3D

A visualização direta por mapeamento de textura realiza a amostragem do volume com planos paralelos à imagem de projeção, e os “empilha” ao longo da direção de observação. A renderização desses planos é feita como polígonos recortados (fatias) nos limites da textura do volume. Os dados do volume são aplicados como texturas nessas fatias e as imagens resultantes são combinadas de trás para frente em direção à posição de observação como ilustrado na Figura 2.9. A combinação dos valores dos *pixels* de cada fatia no *frame buffer* é realizada durante a renderização para obter o efeito de transparência apropriado. Essa combinação dos *pixels* é feita utilizando uma função de transferência pré-determinada.

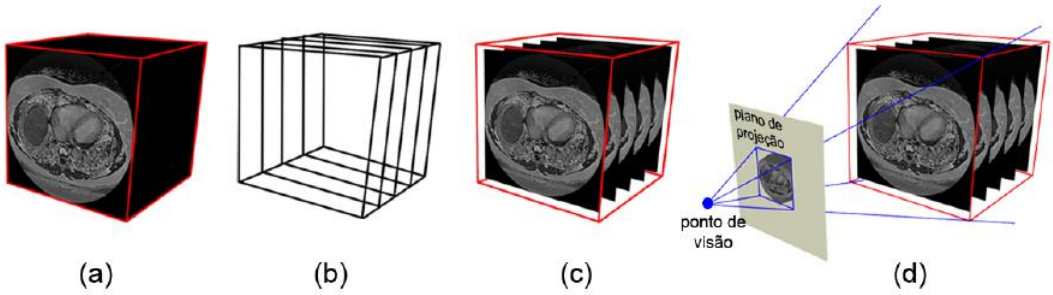

Figura 2.9: Visualização direta por mapeamento de texturas. (a) volume de dados (b) fatias, (c) mapeamento de fatias no espaço de textura, (d) renderização das fatias no *frame buffer* (Com permissão para reprodução, HUFF, 2006).

Considerando que a placa de vídeo do microcomputador suporte texturas 3D, é possível realizar a renderização das fatias paralelas à imagem de projeção com relação à direção de observação atual, ou seja, se a direção de observação em relação ao volume for alterada, as fatias devem ser recalculadas, como ilustrado na Figura 2.10 (REZK-SALAMA *et al*., 2000). Quanto maior o número de fatias, melhor a qualidade da imagem.

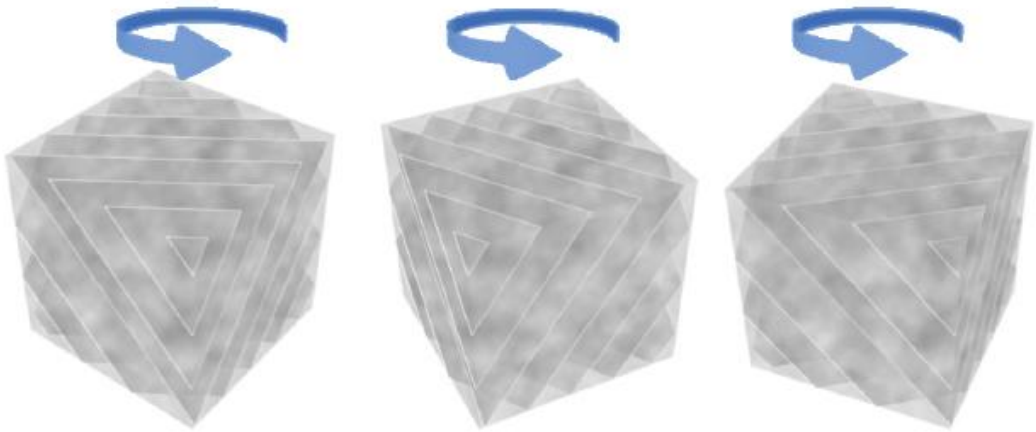

Figura 2.10: Fatias da visualização direta por mapeamento de textura perpendiculres à direção de observação. Cada vez que a direção de observação muda é feito novamente o cálculo da cor e opacidade do plano de projeção (Com permissão para reprodução, HUFF, 2006).

A implementação da visualização direta de volumes baseada no mapeamento de textura pode ser dividida em três estágios (IKITS *et al*., 2004): inicialização, atualização e desenho. No estágio de inicialização, o conjunto de dados volumétricos é processado e armazenado como uma textura 3D na GPU (*Graphics Processing Unit*). Durante esse processo existem duas opções a serem escolhidas: uma pré-cassificação determinada por uma função de transferência (KINDLMANN e DURKIN, 1998) ou o cálculo de valores escalares e gradiente para a iluminação e sombreamento do volume (WEISKOPF *et al.*, 2002). Já no estágio de atualização, o usuário interage com o sistema, modificando parâmetros de visualização, como alterando a posição de observação ou o plano de projeção, e se o *hardware* suportar texturas 3D, na abordagem com planos paralelos ao plano de projeção, esta modificação necessita que todas as fatias de amostragem sejam recalculadas. Para gerar a imagem final durante o estágio de desenho, a cor de cada fragmento das fatias de amostragem é calculada e combinada no *frame buffer*.

2.4 BIOMICROSCOPIA ULTRASSÔNICA

Com a BMU, podem ser obtidas informações de tecidos saudáveis ou doentes, in vivo, ao se analisarem as imagens geradas com resolução de uma microscopia panorâmica, realizando neste caso uma biópsia virtual. A biomicroscopia ultrassônica pode ser utilizada na geração de imagens para a oftalmologia, de cartilagem e da parede

dos vasos sanguíneos, entre outros (FOSTER et al., 1999). Para a obtenção de uma resolução muito melhor (em torno de 50 μm) daquela obtida com a ultrassonografia convencional, que é 10 vezes menor. A BMU utiliza frequencias mais elevadas e com isso perde poder de penetração pela elevada atenuação da onda ao se propagar pelos tecidos biológicos. Tipicamente, a profundidade de penetração para a BMU está na escala de milímetros ao passo de que para a ultrassonografia convencional situa-se na escala de centímetros.

3 MATERIAIS E MÉTODOS

3.1 BMUs

Os componentes essenciais de um sistema de BMUs são semelhantes aos do sistema convencional para geração de imagens em modo B, exceto por operar em frequências elevadas (aproximadamente uma ordem de magnitude acima).

O sistema de BMUs, cujo diagrama de blocos é mostrado na Figura 3.1, consiste de um gerador de pulsos monociclo, com 400 volts de amplitude pico a pico e frequência central ajustada entre 25 e 50 MHz (AVB2-TA-C, Avtech Eletrosystem, Ottawa, Canadá), usado para excitar o transdutor. O pulso de US gerado é transmitido pelo meio e os ecos retroespalhados são detectados pelo mesmo transdutor. O sinal de rádio frequência (RF) na saída do transdutor é utilizado para a construção das imagens 2D.

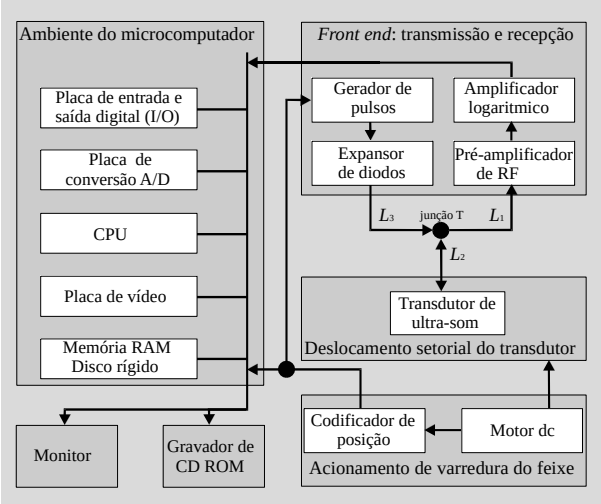

Figura 3.1: Diagrama de blocos do sistema de BMUs.

O transdutor utilizado, com focalização esférica, opera em 40 MHz e contém uma fina camada piezoelétrica de difluoreto de polivinilideno (PVDF) com metalização a ouro (Capistrano Labs, Inc.; San Clemente, EUA). Tem ainda como características: distância focal de 12 mm, profundidade de campo de 1,7 mm e $f_{número}$ de 2,4.

Basicamente, a obtenção de imagens de ultrassom começa com o transdutor de ultrassom movimentando-se em forma pendular no plano de imagem (1,5 por 10 mm). Para isto, o transdutor foi montado em um sistema eletromecânico de varredura setorial constituído por um motor CC com eixo acoplado a um redutor de 14:1 (A-MAX26+GP26B; Maxon, Suíça) e um codificador óptico (BHK 06.24K1024-I6-5; Baumer, Suíça), que permite a obtenção de imagem em tempo real numa taxa de 2 quadros/segundo.

O codificador óptico gera 1024 pulsos de sincronismo por volta e outro pulso, denominado indexador, por cada volta. O sistema de BMUs somente adquire os sinais de eco durante meio ciclo da oscilação do pêndulo de varredura do feixe. Desta forma, o transdutor é excitado 512 vezes em cada oscilação e para cada excitação são capturados os sinais de eco gerados ao longo do feixe, ou de cada linha de varredura. O sistema de BMUs varre um setor de 14,6°, com as linhas de varredura espaçadas de 0,029°, sendo geradas 512 linhas de modo-A por cada quadro de imagem. Os sinais de pulso emitidos pelo codificador são usados como sinal de sincronismo, sendo então responsáveis pelo disparo do gerador de pulsos monociclo. Uma vez excitado, o transdutor emite um pulso de US no meio e o sinal retroespalhado é coletado no mesmo transdutor, cujo sinal de saída passa por um amplificador de RF (AU-1114, Miteq, Hauppauge, Canadá), por um filtro passa-banda (25 a 70 MHz) constituído de um filtro passa alta com frequência de corte em 25 MHz (modelo BHP-25, Mini circuits, Brooklyn, NY, EUA) em série com um filtro passa-baixa com frequência de corte em 70 MHz (modelo BLP-70; Mini circuits, Brooklyn, NY, EUA). A saída do filtro é conectada à entrada de uma placa de conversão analógico para digital, A/D, (modelo NI PCI-5114; National Instruments, Austin TX, EUA). Os sinais digitalizados pela placa são transferidos para o microcomputador que realiza a detecção da envoltória dos sinais de eco, a compressão logarítmica da envoltória e a conversão de varredura para a geração das imagens de BMUs. A placa A/D, com dois canais e 8 MB de memória RAM por canal, funciona com frequência de amostragem de 250 MHz. Esta placa é instalada no barramento *Peripheral Component Interconnect* (PCI) do microcomputador.

A conexão entre o transdutor, o pulsador e o amplificador de RF usa circuitos de chaveamento a diodo para acoplar o transdutor ao pulsador, durante sua excitação, e para acoplá-lo ao amplificador de RF durante a recepção do sinal retroespalhado.

Adicionalmente, o microcomputador é também responsável por controlar um sistema de posicionamento, estando conectado a este via interface RS-232. Toda a programação computacional do sistema de BMUs realiza-se em linguagem LabVIEW (versão 7.1; National Instruments, Austin, TX, EUA).

3.2 SISTEMA DE POSICIONAMENTO

A geração de imagem 3D baseia-se na aquisição de múltiplas imagens 2D de BMUs correspondentes a planos de imagem paralelos e equidistantes. Neste caso, um sistema de posicionamento é usado para realizar o deslocamento relativo entre o meio, do qual é gerada a imagem de BMUs, e o sistema de varredura do feixe de ultrassom. Nesta dissertação foi usado o sistema de posicionamento modelo ESP300 (Newport; Irvine, EUA) composto por um equipamento controlador de movimento mostrado na Figura 3.2 que funciona como um *driver* responsável pelo acionamento e controle dos motores usados nos estágios lineares (MFA-CC, Newport, Irvine, EUA) configurados em um plano horizontal XY. Cada estágio linear (Figura 3.3) possui curso total de 2,54 cm, passo com precisão de 0,0175 µm e uma velocidade máxima de 2,5 mm/s. O meio, cuja imagem de BMUs 3D se pretende obter, é colocado sobre uma plataforma fixada sobre os dois estágios. Durante a aquisição de imagens de BMUs 2D, a plataforma é deslocada, de forma controlada, numa direção perpendicular aos planos de imagens 2D.

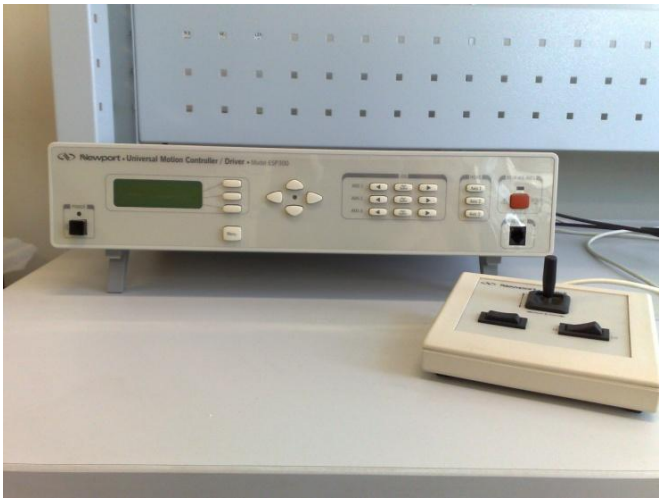

Figura 3.2: Equipamento controlador de movimento, ESP300, e *joystick*

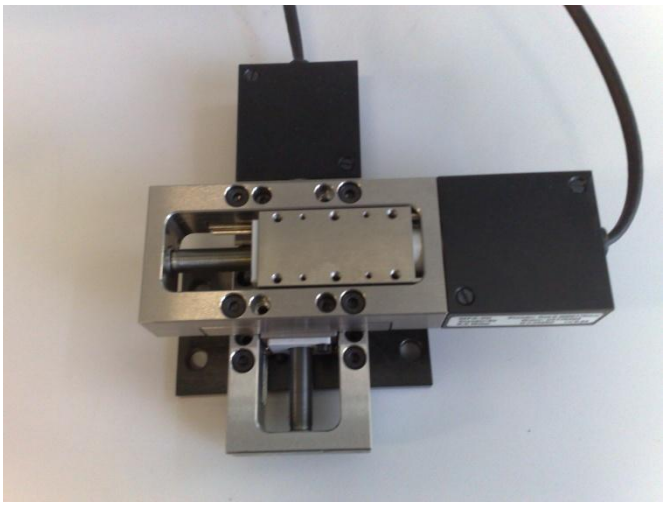

Figura 3.3: Sistema de posicionamento em vista superior, formado por dois estágios lineares dispostos em 90°

3.3 AQUISIÇÃO DE IMAGENS EM 2D

Para a aquisição das imagens em 2D, os motores usados nos estágios lineares configurados em um plano horizontal XY necessitam de um controle segundo o qual o estágio linear no eixo X (paralelo aos planos de imagem) fica parado e o do eixo Y (perpendicular aos planos de imagem) se move a passos controlados. Após cada passo no eixo Y, o sistema de posicionamento se mantém parado por 3 segundos (este tempo pode ser estipulado pelo usuário) para adquirir e salvar a imagem 2D do plano correspondente. O operador define o tamanho do passo e o curso total, e o sistema determina o número de passos correspondentes e a velocidade máxima possível, de acordo com as especificações do fabricante, para a atuação do estágio linear.O método de reconstrução volumétrica utilizado foi o VNN.

Após a aquisição dos sinais correspondentes a todas as linhas modo-A que compõem um quadro de imagem, ocorre a conversão de varredura, realizada por interpolação bilinear, a qual mapeia os dados que foram obtidos utilizando coordenadas polares em um sistema de coordenadas retangulares. A etapa computacional da conversão de varredura é executada em linguagem C (AT&T Bell *Laboratories*) através de uma DLL (*Dinamic Link-Library*) carregada no programa computacional do BMUs desenvolvido em LabVIEW. A Figura 3.4 apresenta um diagrama de blocos

correspondente a sequência de etapas relacionadas com a aquisição dos múltiplos quadros de imagem 2D, onde o N inicial é o número de passos a serem dados pelo estágio linear de eixo Y, ou seja, o número de imagens em 2D a serem adquiridas.

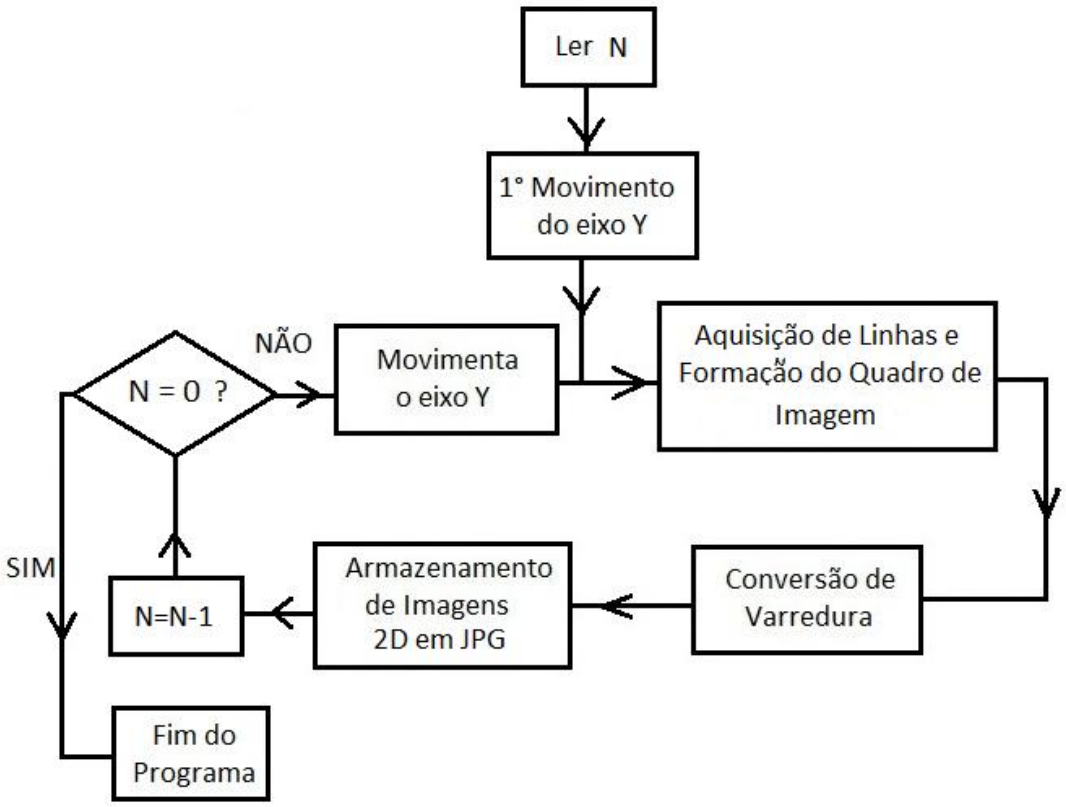

Figura 3.4: Diagrama de blocos contendo a sequência de operações para a obtenção do conjunto das imagens 2D usadas para a geração da imagem 3D, onde N é o número de imagens a serem adquiridas.

3.4 GEOMETRIA DA VARREDURA SETORIAL DO FEIXE DE ULTRASSOM

O transdutor utilizado na instrumentação BMUs realiza um movimento pendular, como observado na Figura 3.5. O ponto "O" é o centro de um eixo em torno do qual o transdutor oscila de forma pendular com ângulo total θ_{setor}. A região definida pela profundidade de campo (*DOF*, do inglês *depth of field*) do feixe emitido pelo transdutor utiliza os sinais de eco adquiridos e usados na construção de imagem do BMUs. O quadro de imagem é construído a partir dos sinais de eco oriundos da DOF e que correspondem a 512 direções do feixe emitido pelo transdutor. As linhas de

varredura do feixe são compostas pelas direções do feixe e o número de pontos amostrados dos sinais de eco para cada direção do feixe é indicado por N_{RF}.

A Figura 3.5 também ilustra algumas definições relacionadas com a geometria da varredura do feixe que serão usadas como referência para os cálculos a serem demonstrados ao longo da dissertação. Exemplo delas são a DOF e a distância focal, df, que compreende a distância da face do transdutor ao seu ponto focal. Tais parâmetros são características exclusivas do transdutor de ultrassom, enquanto outros parâmetros tais como o Raio, ângulo (θsetor) e Largura são inerentes à geometria de varredura do feixe, sendo definidos pelo operador, como uma variável no programa de geração de imagens 2D, de acordo com a aplicação de interesse. A DOF está centralizada no ponto focal, se estendendo de DOF/2 para ambos os lados de df.

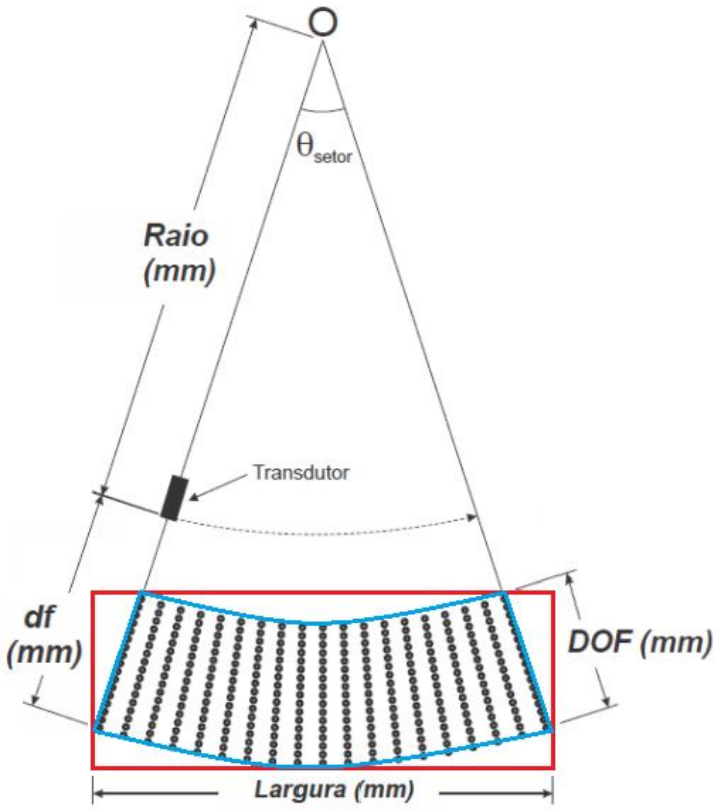

Figura 3.5: Ilustração da disposição dos dados amostrados dos sinais de eco que compõem a imagem de biomicroscopia ultrassônica setorial, levando-se em conta a geometria da região submetida à varredura do feixe de ultrassom. O retângulo em vermelho corresponde ao quadro de imagem que será visualizado no montor. A região setorial varrida pelo feixe é limitada por θ_{setor} e alguns parâmetros utilizados na formação da imagem de biomicroscopia ultrassônica setorial, como a profundidade de campo e a distância focal.

Com a aquisição dos sinais de eco usados na formação da imagem, é gerada uma matriz de dados de dimensão 512 (número de feixes por quadro de imagem) por N_{RF} (número de pontos adquiridos em cada feixe).

O valor de N_{RF} é dado por:

$$N_{RF} = \frac{2\,fa}{c} DOF\,, \tag{1}$$

onde *fa* a frequência de amostragem (250 MHz) e *c* a velocidade de propagação da onda ultrassônica no meio.

A placa de aquisição inicia a digitalização do sinal de eco correspondente a uma determinada direção do feixe assim que recebe o sinal de *trigger*, que é o pulso de sincronismo do codificador óptico, correspondente ao instante de excitação do transdutor. Apenas os dados digitalizados correspondentes à *DOF* são armazenados e usados para a geração da imagem. O valor de N_{RF} é contado a partir do ponto *pInicio* calculado por:

$$pInicio = \frac{2\,fa}{c}\left(df - \frac{DOF}{2}\right) \tag{2}$$

Os dados amostrados formam uma matriz, M_O (512xN_{RF}), ilustrada em (3). Cada linha dessa matriz contém os dados de uma determinada linha de varredura do feixe compreendidos entre os instantes *pInicio* e *pInicio* + N_{RF}.

$$Mo = \begin{bmatrix} M_{0_{0,0}} & \cdots & M_{0_{0,NRF}} \\ \vdots & \ddots & \vdots \\ M_{0_{511,0}} & \cdots & M_{0_{511,NRF}} \end{bmatrix} \tag{3}$$

3.5 CONVERSÃO DE VARREDURA

A conversão de varredura mapeia os dados que foram obtidos utilizando coordenadas polares em outros relacionados a um sistema de coordenadas retangulares, possibilitando a exibição dos mesmos no monitor de um computador.

3.5.1 FORMAÇÃO DA IMAGEM

3.5.1.1 Matriz Final

A conversão de varredura é realizada de trás pra frente, sendo iniciado pela matriz que formará a imagem, denominada a Matriz Final (M_F), como observado na Figura 3.6.

A M_F é construída como visto na parte inferior da Figura 3.7, onde cada coluna contém as coordenadas de um ponto (*pixel*) da imagem e nas linhas tem-se respectivamente a coordenada da linha (Y), da coluna (X) e 1. A enumeração dos índices relacionados com linhas e colunas inicia-se com zero.

A variável *Largura* (Figura 3.8) é responsável pelo cálculo do número de colunas X_{coluna} e a variável *Altura* (Figura 3.8) é responsável pelo cálculo do número de linhas Y_{linha}, como explicitado em (4).

$$X_{coluna} = \frac{2\,fa}{c} Largura \qquad Y_{linha} = \frac{2\,fa}{c} Altura \tag{4}$$

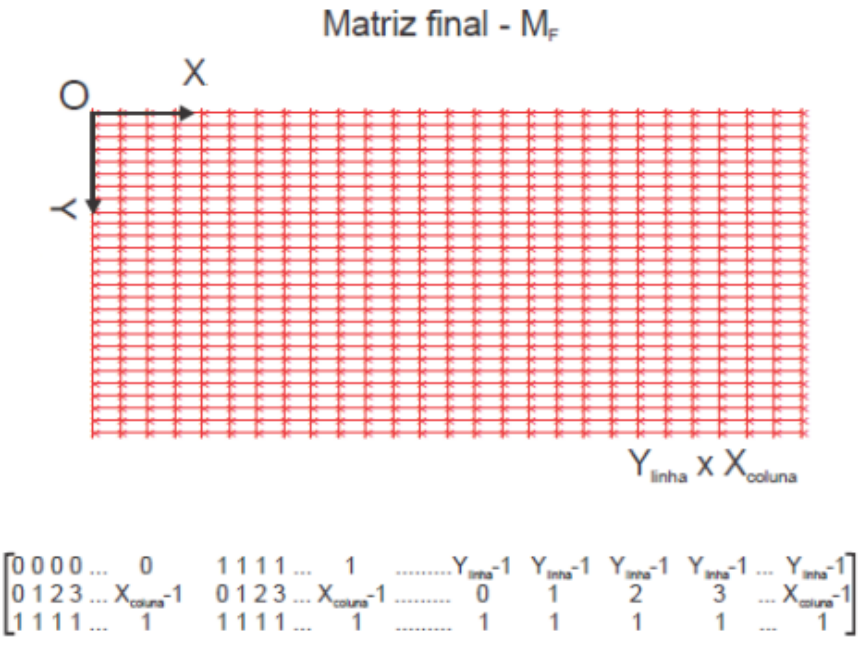

Figura 3.6: Representação da Matriz Final de dados para exibir a imagem e sua forma matemática utilizada pelo programa computacional. A enumeração dos índices relacionados com linhas e colunas inicia-se com zero.

Com base na Figura 3.7, chega-se ao cálculo da *Altura* de acordo com:

$$Altura = (raio + df - \frac{DOF}{2})[1 - \cos(\theta_{setor}/2)] + DOF \quad (5)$$

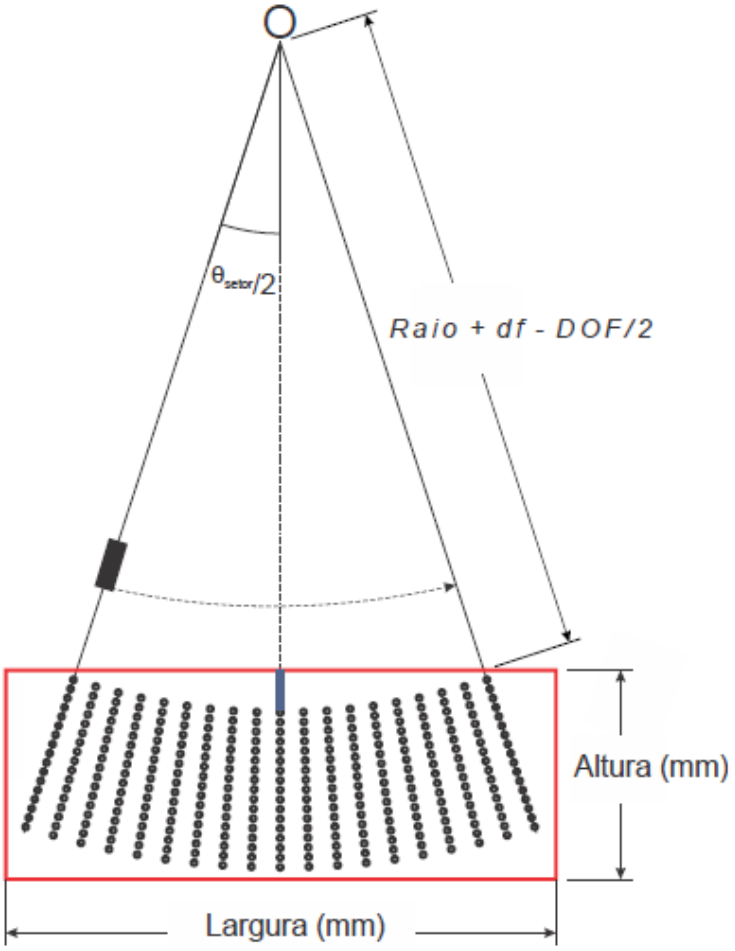

Figura 3.7: Representação da geometria da varredura, usada para o cálculo da Altura. A Altura é a profundidade de campo mais o traço verde dentro do quadro vermelho.

3.5.1.2 Translação

O eixo do sistema, indicado pelo ponto 'O', é ponto de origem do sistema de coordenadas polares e do sistema de coordenada cartesiana (0,0) da Matriz Final.

Para a geração da imagem é necessária a localização espacial dos dados de interesse, que estão geometricamente bem abaixo das coordenadas de cada *pixel* da Matriz Final. Assim, é associada à localização dos pontos amostrados situados na *DOF*

uma Matriz Intermediária, M_I, que após uma translação tem seus *pixels* coincidentes com os da M_F. A disposição espacial de M_I e M_F é representada na Figura 3.8.

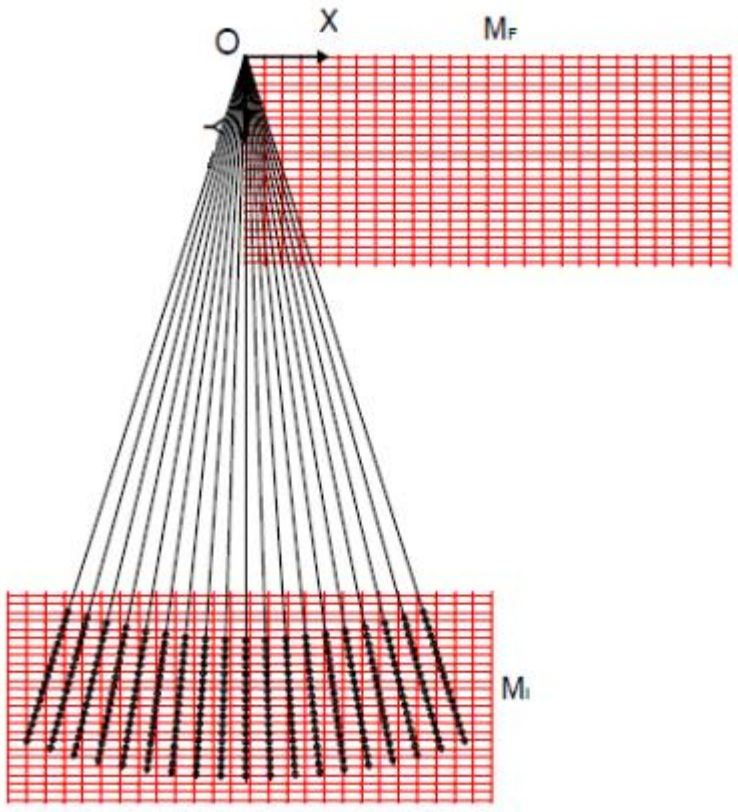

Figura 3.8: Diagrama que apresenta a disposição espacial entre as matrizes M_I e M_F.

A translação de M_I para M_F é realizada através de uma operação entre M_I e M_F por meio da Matriz de Translação, T, definida em (7). Os termos de T são esquematizados na Figura 3.9, a qual contém os deslocamentos verticais e horizontais, da translação, definidos por Δ_{linhas} e $\Delta_{colunas}$, respectivamente.

As relações entre M_I e M_F são dadas pelas seguintes equações:

$$M_F = T \cdot M_I \tag{6}$$

$$T = \begin{bmatrix} 1 & 0 & -\Delta_{linhas} \\ 0 & 1 & \Delta_{colunas} \\ 0 & 0 & 1 \end{bmatrix} \tag{7}$$

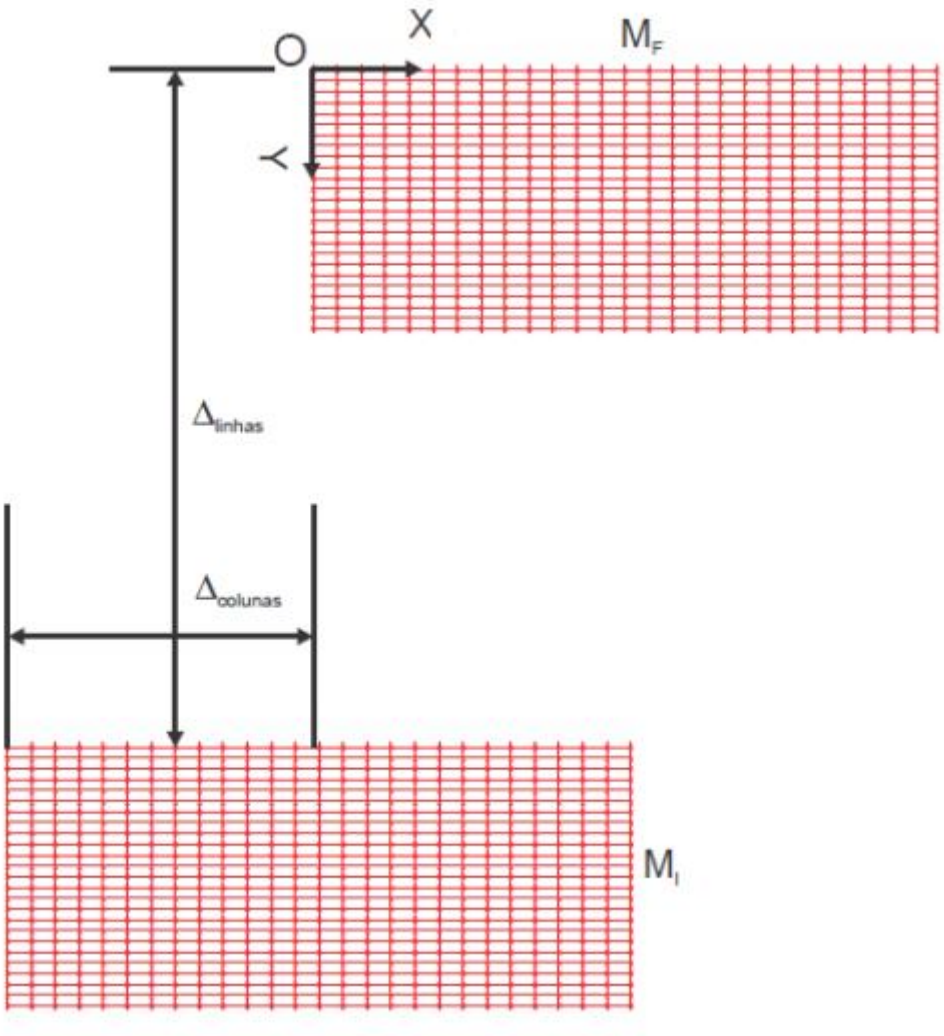

Figura 3.9: Esquematização para o cálculo do deslocamento para a translação da Matriz Final para a Matriz Intermediária.

3.5.1.3 Cálculo do Deslocamento

A Matriz Final deve ter deslocamento na horizontal igual à metade da sua largura, ou seja, $0{,}5X_{coluna}$. Para o cálculo do deslocamento de Δ_{linha} é necessário determinar a altura, *h*, do triângulo isósceles formado pelos vértices O, P e Q ilustrado na Figura 3.10. Utiliza-se a relação trigonométrica abaixo para se determinar *h*:

$$h = (Raio + df - \frac{DOF}{c})\cos(\theta_{setor}/2) \tag{8}$$

A altura *h* é calculada em milimetros e para ser convertida em número de *pixels* , fornecendo Δ_{linha}, usa-se a mesma relação utilizada no cálculo de X_{coluna}:

$$\Delta_{linha} = \frac{2\,fa}{c} h \tag{9}$$

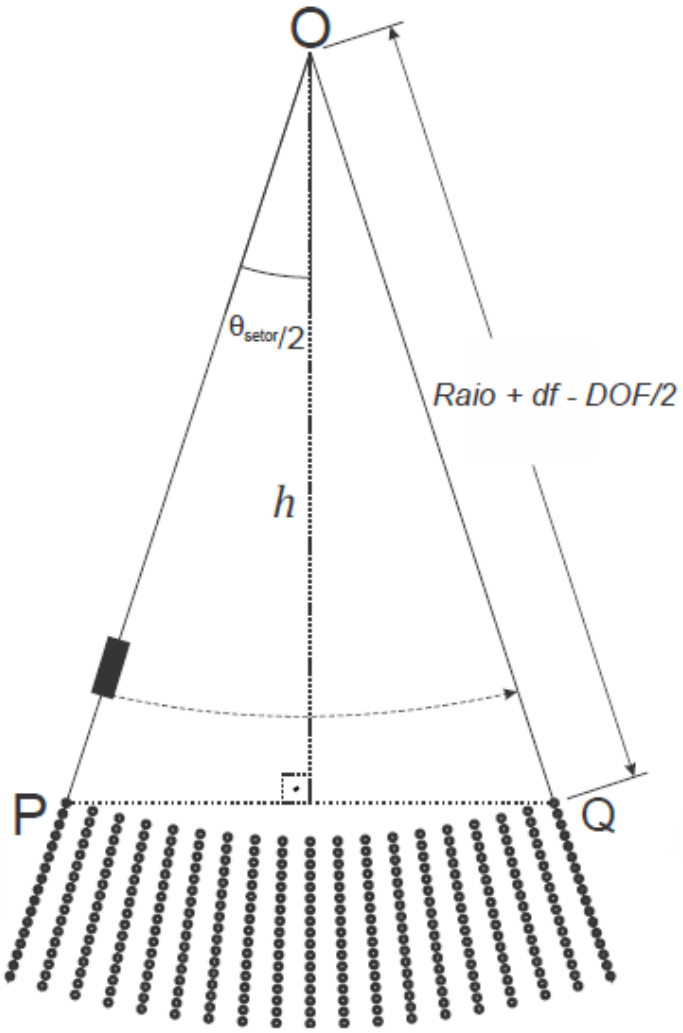

Figura 3.10: Representação do triângulo isósceles OPQ utilizado para cálculo da sua altura *h*.

Assim, a partir das matrizes T e M_F pode-se chegar a matriz M_I. A matriz M_F está em coordenadas retangulares conforme mostrado na Figura 3.10, logo, M_I também estará. Por disso, M_I é convertida para coordenadas polares ($r_{m,n}$,$\theta_{m,n}$) seguindo as fórmulas:

$$r_{m,n} = \sqrt{X_n^2 + Y_m^2} \tag{10}$$

$$\theta_{m,n} = \tan^{-1}\left(\frac{X_n}{Y_m}\right), \tag{11}$$

onde X_n e Y_m são as coordenadas de cada *pixel* de M_I.

A partir da equação das coordenadas polares, a M_I pode ser equacionada como mostrado abaixo:

$$M_I = \begin{bmatrix} r_{0,0} & r_{0,1} & r_{0,2} & & r_{(Y_{linha}-1, X_{coluna}-1)} \\ \theta_{0,0} & \theta_{0,1} & \theta_{0,2} & \cdots & \theta_{(Y_{linha}-1, X_{coluna}-1)} \\ 1 & 1 & 1 & & 1 \end{bmatrix} \tag{12}$$

A matriz de dados amostrados (M_O) tem suas linhas referenciadas ao índice da linha de varredura do feixe, ou seja, de 0 a 511. Esse índice está diretamente relacionado com o ângulo da coordenada polar correspondente à mesma direção do feixe. Portanto, deve-se trabalhar com os índices referentes às coordenadas de linhas da matriz M_I, e não com o ângulo da coordenada polar, de forma a representá-la com a mesma notação da matriz de dados amostrados. Para se chegar a esse índice, que funciona como um contador das linhas, algumas operações são necessárias, as quais incluem o cálculo de um *offset* de ângulo e relação entre quantidade de linhas de varredura por unidade de ângulo. Estas operações são definidas a seguir.

3.5.1.4 *Offset*

Essa transformação em coordenadas polares citada anteriormente e calculada pelo LabVIEW utiliza a mesma referência para o ângulo que o círculo trigonométrico. Entretanto, o ângulo utilizado nesta transformação é o ângulo ascendente no sentido horário (Figura 3.11).

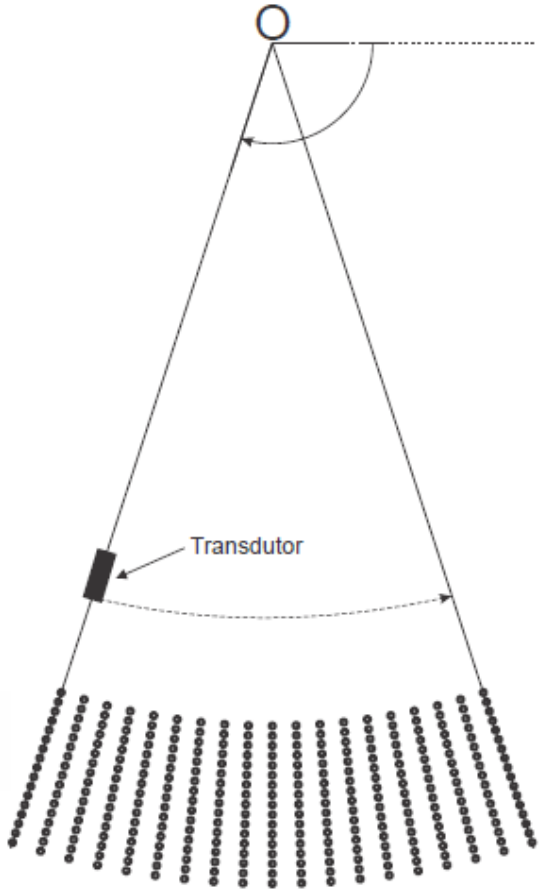

Figura 3.11: Referência angular utilizada na conversão das coordenadas retangulares para polares.

A função do *offset* é a de mudar a referência para que a posição do transdutor no início de cada ciclo da varredura do feixe (extremidade esquerda na Figura 3.13) apresente o valor do ângulo nulo. Portanto, cada ângulo inicia com valor zero e é incrementado a cada amostra até chegar em 511, totalizando 512 feixes por ciclo. Desta maneira, os ângulos são representados em contagem de linhas.

3.5.1.5 Cálculo do *Offset*

Como o *offset* é definindo como origem da contagem de ângulos, então de acordo com a Figura 3.13, o ângulo correspondente ao posicionamento mais à esquerda do feixe de ultrassom é dado por:

$$\theta_{offset} = \frac{\pi}{2} + \frac{\theta_{setor}}{2} \tag{13}$$

Uma variável que relaciona a quantidade de linhas radiais por unidade de ângulo, definida por *vetorPorSetor*, será utilizada para o cálculo do *offset* e seu cálculo é realizado por:

$$vetorPorSetor = \frac{512}{\theta_{setor}} \tag{14}$$

Para se obter a quantidade de linhas radiais dentro desse intervalo de *offset* (θ_{offset}) é necessário multiplicar o ângulo do *offset* pela quantidade de linhas por unidade de ângulo. A partir desta multiplicação tem-se:

$$offset = \theta_{offset} vetorPorSetor \tag{15}$$

A partir do valor de *offset* que foi calculado anteriormente, pode-se inserí-lo na matriz M_I. Para tal cálculo, é necessário gerar mais uma matriz de transformação, *posConv*, definida da maneira:

$$posConv = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -vetorPorSetor & offset \\ 0 & 0 & 1 \end{bmatrix} \tag{16}$$

Essa matriz é então multiplicada pela M_I já em coordenadas polares. O resultado desta multiplicação é uma matriz onde a primeira linha contém a coordenada radial de cada *pixel*, a segunda linha um índice para contagem das linhas de varredura e a última linha o valor unitário. Esta matriz, $M_I^{'}$ é calculada por:

$$M_I^{'} = posConv \cdot M_I \tag{17}$$

3.5.2 INTERPOLAÇÃO

A última etapa para a conversão de varredura é realizada com o cálculo do valor, em níveis de cinza, atribuído a cada *pixel* da matriz $M_I^{'}$. Este processo é realizado através de uma interpolação, com a qual é feito o mapeamento dos valores da matriz dos dados amostrados, M_O, em M_F, a partir de $M_I^{'}$.

3.5.2.1 Interpolação Bilinear

A interpolação bilinear realiza a média ponderada dos valores correspondentes aos quatro *pixels* vizinhos mais próximos do *pixel* de $M_I^{'}$, para o qual se deseja obter seu valor. Essa média ponderada leva em consideração a distância de cada *pixel* vizinho para o *pixel* de $M_I^{'}$, como mostrado na Figura 3.12.

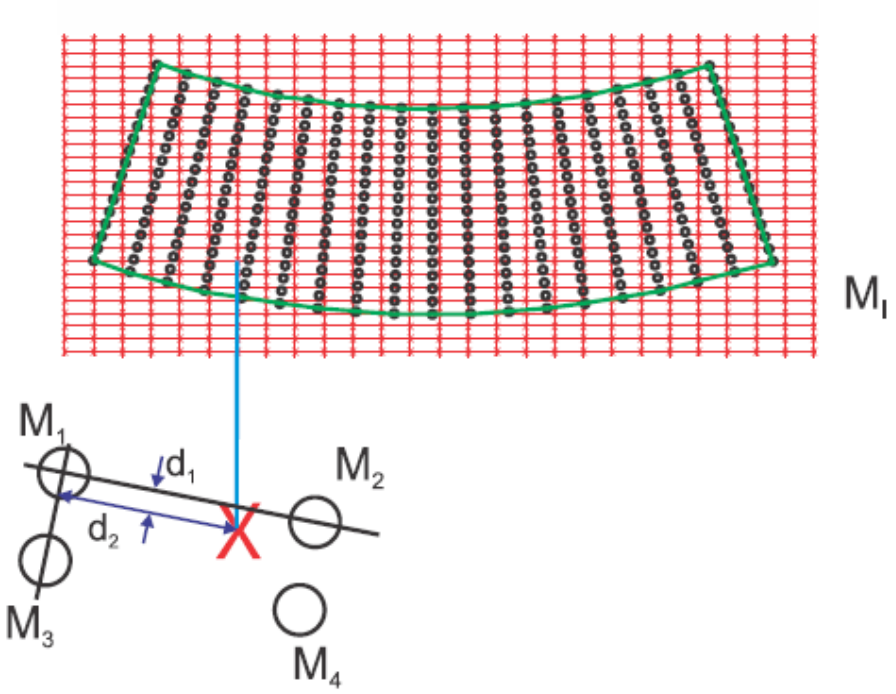

Figura 3.12: Mapeamento dos valores dos *pixels* de M_O, representada pelos pontos pretos, em $M_I^{'}$, representada pela grade retangular. A figura salienta um pixel de $M_I^{'}$ rodeado de 4 outros *pixels* de M_O para os quais correspondem os valores M_1, M_2, M_3 e M_4. As distâncias d_1 e d_2 são usadas na interpolação bilinear.

O cálculo do valor atribuído ao pixel *X*, representado por *f*, e determinado por interpolação bilinear realiza-se como segue:

$$f(X) = M_1 \cdot (1-d_1)(1-d_2) + M_2 \cdot (1-d_1) \cdot d_2 + M_3 \cdot d_1 \cdot (1-d_2) + M_4 \cdot d_1 \cdot d_2 \qquad (18)$$

3.5.2.2 DLL

O cálculo da interpolação bilinear é implementado por uma DLL (*Dynamic Link Library*) desenvolvida especificamente para esse projeto. Essa DLL é responsável pela

exibição em tempo real da imagem, representando a etapa de maior processamento da máquina. Sem a utilização dessa DLL, o programa computacional do BMUs para gerar as imagens e baseado em LabVIEW não foi capaz de realizar todo o processamento da imagem em tempo real.

Primeiramente, a DLL utiliza os vetores com as partes inteiras da magnitude e do ângulo da coordenada de cada *pixel*, anteriormente gerados a partir da $M_I^{'}$, para determinar se cada *pixel* está dentro da região que será preenchida com os dados da aquisição. Os pontos externos a essa área não participam do processo de interpolação e recebem o valor determinado para cor de fundo, no caso o preto.

Duas limitações são utilizadas para determinar se um ponto está dentro ou fora da região que receberá a imagem. Uma delas é em relação ao ângulo, contado em termos de índice, que deve estar compreendido entre o intervalo de 0 a 511, limitação essa imposta conforme explicado anteriormente. A segunda limitação é em relação à magnitude (módulo) da coordenada polar da posição do *pixel* deve estar entre o $raio + df - DOF/2$ e $raio + df + DOF/2$.

As coordenadas radiais de cada *pixel* da matriz $M_I^{'}$ são convertidas para números de pontos equivalentes, tendo-se por base a velocidade de propagação da onda e a frequência de amostragem dos sinais de eco.

Portanto, os limites do intervalo $raio + df - DOF/2$ e $raio + df + DOF/2$ são expressos por:

$$min\,Pontos = \frac{2\,fa}{c}\left(raio + df - \frac{DOF}{2}\right) \tag{19}$$

e

$$max\,Pontos = \frac{2\,fa}{c}\left(raio + df + \frac{DOF}{2}\right) \tag{20}$$

A região da imagem é delimitada a partir dessas condições preliminares. Os dados que são preenchidos pela interpolação estão representados, como exemplo, pela

região branca na Figura 3.13. No eixo Y observa-se o número de linhas e no eixo X o número de colunas da Matriz Final.

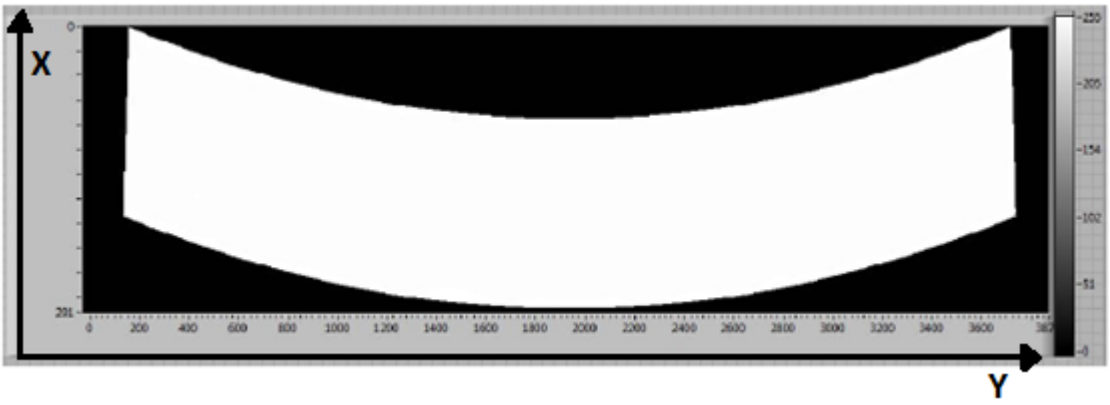

Figura 3.13: Exemplo das regiões da imagem preenchidas durante a execução da DLL. A região em branco contém os *pixels* cujos valores são obtidos pela interpolação bilinear.

Após o fim da interpolação a imagem está pronta para ser exibida no painel frontal do programa em LabVIEW executado no âmbito do BMUs.

3.5.2.3 Quadros de Imagem

O programa gera os arquivos em formato JPEG no mesmo instante que eles são exibidos na tela do monitor.Os dados oriundos da interpolação são convertidos em uma forma compatível com o formato dos arquivos JPEG, e são armazenados em disco rígido com o auxílio de blocos de funções preexistentes no LabVIEW para tal finalidade.

Essa conversão dos dados consiste da passagem da matriz que está em duas dimensões para um vetor. Além disso, é necessária a utilização de uma tabela de cores para armazenar a imagem em uma escala de cinza.

O diretório e o nome do arquivo utilizados para armazenar as imagens são definidos pelo usuário antes da inicialização do BMUs. Os nomes dos arquivos são compostos pela união do seu nome com uma numeração gerada de forma automática e crescente, que tem como objetivo ordená-los e armazená-los cronologicamente.

3.6 CONSTRUÇÃO DAS IMAGENS EM 3D

O programa computacional que gera a imagem de BMU 3D é escrito em linguagem Python (versão 2.7.1, python.org) e executado através da placa de vídeo (NVIDIA GeForce GTS 450) instalada no barramento *Peripheral Component Interconnect express* (PCIexpress) do microcomputador. A execução deste programa parte da leitura dos arquivos contendo as imagens 2D (formato JPEG).

O algoritmo utilizado nesta dissertação foi o de textura 3D. Como comentado anteriormente, o primeiro passo realizado para gerar imagens 3D consiste em ler os dados de cada imagem 2D do conjunto usado para a construção da imagem 3D. As imagens 2D são lidas de trás pra frente como mostra a Figura 3.14a, na qual os quadros ilustram o conjunto das imagens 2D. Após a leitura das imagens 2D, as mesmas são unidas para gerar um volume 3D. Embora o conjunto de dados seja interpretado como uma função contínua no espaço, para praticidade, ele é representado por uma matriz 3D de amostras. Na Figura 3.14b, o ponto $S_{i,j,k}$ representa o endereço de um *voxel* na imagem 3D, situado no *pixel* (i,j) da fatia k. Para gerar uma representação visual do dado volumétrico, os valores associados a cada *voxel* são mapeados em uma cor representada por três canais RGB (*red*, *green* e *blue*) e um valor de opacidade através de uma função de transferência.

A função de transferência utilizada no presente trabalho é unidimensional, e mapeia um valor do dado volumétrico em uma cor (RGB) e um valor de opacidade de forma independente. Normalmente, estas funções de transferência são implementadas como uma "*lookup table*" em textura 1D, onde dado um valor escalar é retornado uma cor e um valor de opacidade. Quando a *lookup table* é construída, a cor e a opacidade são geralmente atribuídas separadamente pela função de transferência. Para uma renderização correta, os componentes da cor tem que ser multiplicados pela opacidade, porque a cor aproxima a emissão e a absorção dentro de um segmento de raios (WITTENBRINK *et al.*, 1998).

Para gerar a visão volumétrica, realiza-se uma amostragens em planos de corte paralelos ao raio de visão (que define a direção de visão). Na memória gráfica, essa textura 3D é criada como uma pilha de fatias de texturas 2D. Como mostrado na Figura 3.14, estes planos geram cortes arbitrários no volume, que podem ser compostos

de trás para frente gerando os *pixels* da imagem final. A vantagem deste método é que as funções para amostrar a textura 3D já são implementadas diretamente no hardware, tornando o processo eficiente e permitindo uma visualização em tempo real.

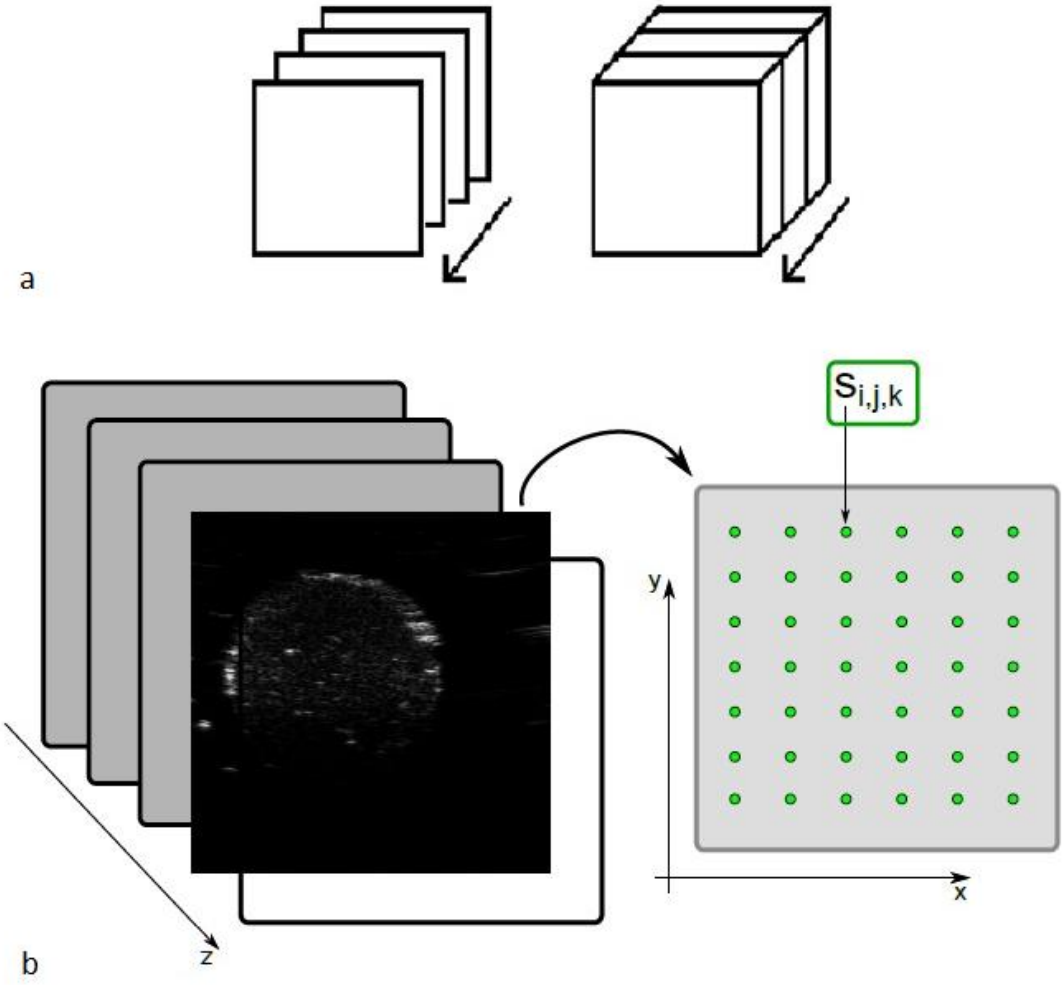

Figura 3.14: a) Imagens 2D organizadas de trás para frente e unidas para formar o cubo. Cortes da imagem para formar o cubo. b) Imagens organizadas de trás para frente e matriz indicando cada *voxel* (pontos verdes).

Para gerar os *pixels* de cada plano de corte, valores em localizações arbitrárias do volume são obtidos por interpolação de dados em elementos de volumes vizinhos (*voxels* vizinhos). No início da renderização, os volumes de dados são carregados na memória da placa gráfica e a função de transferência é tipicamente criada na fase de inicialização da renderização.

Para compor os planos de cortes amostrados a partir do volume 3D é necessário utilizar as equações 21 e 22 para atribuir um valor de cor e opacidade para cada *pixel* do quadro 2D final. Este é um processo de amostragem e desempenha um papel importante na renderização volumétrica, pois permite visualizar o dado a partir de qualquer ponto de vista. Quanto mais planos de cortes são utilizados, maior será a qualidade da imagem final, porém, mais custosa computacionalmente será a visualização. A Figura 3.15 ilustra os *pixels* dos planos de corte (como os cubos azuis) gerados a partir de um ponto de vista, onde as cores e opacidades foram acumuladas a partir das equações 21 e 22 para se chegar ao *pixel* da imagem final (quadrado azul).

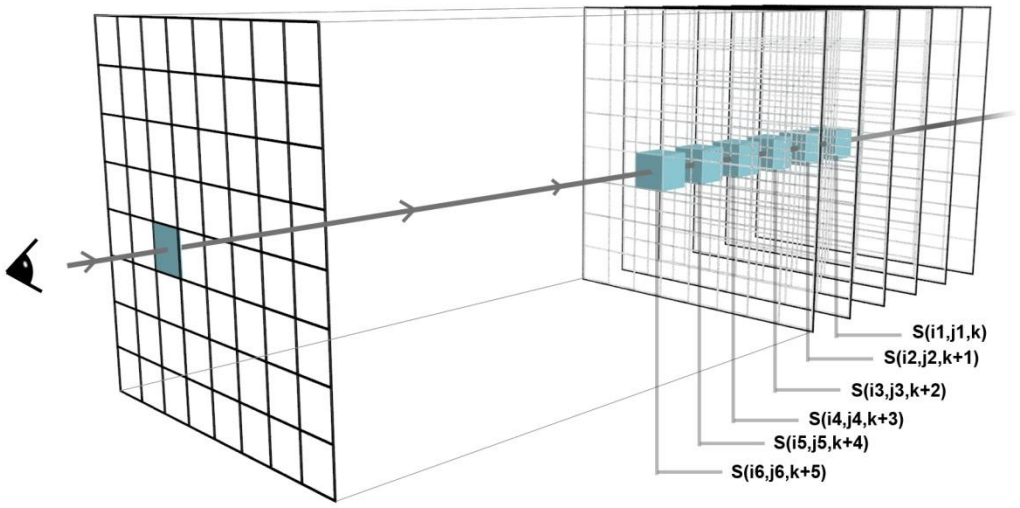

Figura 3.15: *Voxels* dos planos de corte representados pelos cubos azuis para serem acumulados para a formação do corte da imagem 2D. Onde S(i,j,k) representa o valor do *voxel*.

Esses cálculos são realizados para todos os *pixels* da imagem. Para avaliar eficientemente a renderização do volume, as amostras dos planos de corte são ordenadas de trás para frente, e a cor e a opacidade acumuladas são calculadas iterativamente conforme as equações 21 e 22:

$$C_{acum} = C_{acum} + (1 - A_i)C_i \tag{21}$$

$$A_{acum} = A_{acum} + (1 - A_i)A_i, \tag{22}$$

onde: C_{acum} e A_{acum} são a cor e opacidade, respectivamente, acumuladas até o momento, C_i e A_i são a cor e opacidade da fatia sendo processada e a C_{acum} e A_{acum} iniciais são consideradas iguais a zero. Para cada nova fatia a cor e a opacidade de cada *pixel* da imagem é atualizada até que todas as fatias sejam processadas.

Como comentado anteriormente, os valores acumulados de cor e opacidade são calculados toda vez que for feita uma mudança de vista do volume 3D, e uma nova imagem é gerada.

Para a imagem 3D possuir uma escala igual à das imagens 2D, foi realizado um cálculo com a altura, largura e profundidade originais, sendo a profundidade, a distância entre os quadros. O cálculo destes parâmetros é realizado pelo programa de aquisição de imagens 2D.

O diagrama de blocos (Figura 3.16) abaixo ilustra os passos a serem tomados pelo usuário para a geração da imagem 3D.

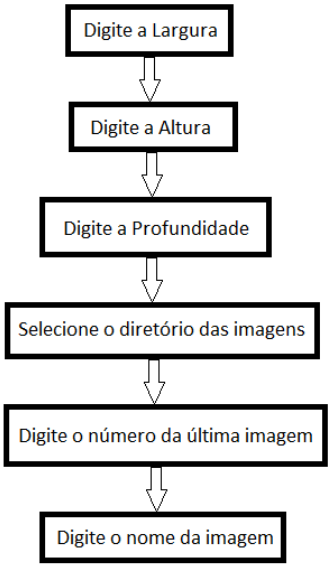

Figura 3.16: Diagrama de blocos dos passos a serem tomados pelo usuário do programa.

3.7 CONSTRUÇÃO DE *PHANTOMS* PARA BMU

Um *phantom* para BMU consite de um meio que mimetiza as propriedades acústicas (velocidade de propagação e coeficiente de atenuação da onda) do tecido

biológico para frequência usada na BMU, sendo composto, por exemplo, de uma mistura de gelatina com partículas de sílica. A construção dos *phantoms* seguiu a metodologia descrita por Ryan e Foster (1997): pó de gelatina (Tipo A: de pele de porco, G-2500, Sigma Chemical Co., EUA) dissolvido em água destilada, numa temperatura controlada de 50ºC, na concentração de 15% em peso até não haver grumos. Como espalhador de ultrassom foi utilizado pó de sílica (S-5631, Sigma Chemical Co., EUA), numa concentração de 2% em peso.

Os *phantoms* foram construídos no formato cilíndrico e constituídos de duas partes; a parte interna moldada em um formato predominantemente cônico e a parte externa, recobrindo a parte interna, que conferiu aos *phantoms* a forma externa cilíndrica. A moldagem da parte externa baseou-se em uma seringa de 3 ml (Becton Dickinson, NJ, EUA) que teve sua extremidade do lado da agulha seccionada transversalmente ao eixo de simetria da seringa. Já a moldagem da parte interna baseou-se na utilização de ponteiras, de ponta fina, de micropipetas (Plast Labor, RJ, Brasil) em polipropileno transparente. Foram usadas dois tipos de ponteiras, uma para micropipetas de até 200 µL, ilustrada na Figura 3.17, e outra para micropipetas de até 1000 µL e ilustrada na Figura 3.18.

Figura 3.17: Ponteira de micropipeta com volume variável entre 1 e 200 µL da marca Plast Labor.

Figura 3.18: Ponteira de micropipeta com volume variável entre 100 e 1000 µL da marca Plast Labor.

Uma seringa com uma de suas extremidades fechada pelo êmbolo foi preenchida com gelatina dissolvida em água quente, misturada ou não com espalhadores, e mantida na posição vertical. A seguir, uma ponteira de micropipeta foi inserida através da extremidade aberta da seringa e alinhada com seu eixo de simetria. Após a inserção da ponteira de micropipeta, a seringa contendo a gelatina foi colocada na geladeira, por cerca de 12 horas, para aguardar o endurecimento da gelatina. Depois de retirada da

geladeira, foi removida a ponteira de micropipeta, ficando assim moldada a parte interna e cônica do *phantom*, que foi posteriormente preenchida com gelatina misturada com espalhadores ou com apenas água. Previamente ao preenchimento da seringa, foi borrifado em sua parede interna óleo de silicone líquido para facilitar a retirada do *phantom* de dentro da seringa.

Basicamente, dois tipos de *phantoms* foram construídos. Para o *phantom* do tipo1, a parte externa foi formada de gelatina pura e a parte interna da mistura de gelatina e pó de sílica. Já o do tipo 2 foi construído com a parte externa contendo a mistura de gelatina com pó de sílica e a parte interna contendo apenas água. A parte formada pela mistura de gelatina com pó de sílica é hiperecóica, ao passo que a parte formada por gelatina pura ou preenchida com água é anecoica.

Após a retirada dos *phantoms* da seringa, os mesmos foram imersos, por 5 minutos, em uma solução de formol a 5% (B.Herzog, Brasil) a fim de lhes dar mais consistência e prolongar a vida útil dos mesmos, evitando a formação de fungos (RYAN e FOSTER, 1997).

Foram construídos 3 *phantoms* diferentes, sendo dois do tipo 1. Na Figura 3.19 está ilustrado o *phantom* 1, do tipo 1, cuja parte interna foi moldada com a ponteira de micropipeta menor. Já a Figura 3.20 ilustra o *phantom* 2, também do tipo 1, cuja parte interna foi moldada com a ponteira de micropipeta maior.

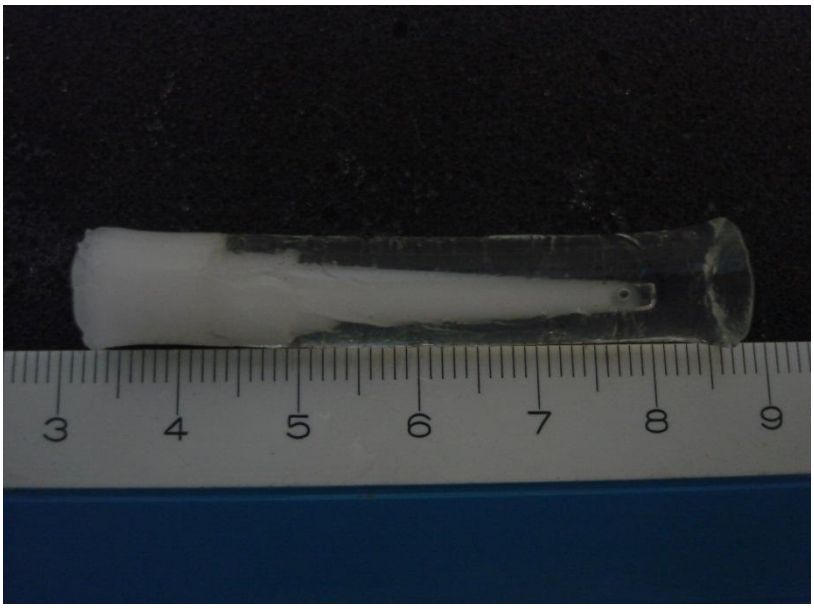

Figura 3.19: Foto do *phantom* 1. Sua parte interna foi moldada com a ponteira de uma micropipeta de 3 segmentos consistindo, da direita para a esquerda, dos formatos cônico, cilíndrico (diâmetro menor) e cilíndrico (diâmetro maior).

Figura 3.20: Foto do *phantom* 2. Sua parte interna foi moldada com a ponteira de uma micropipeta de formato cônico e seu formato é cônico

A Figura 3.21 ilustra o *phantom* 3, o qual é do tipo 2. Sua parte interna foi moldada com a mesma ponteira de micropipeta utilizada no *phantom* 1.

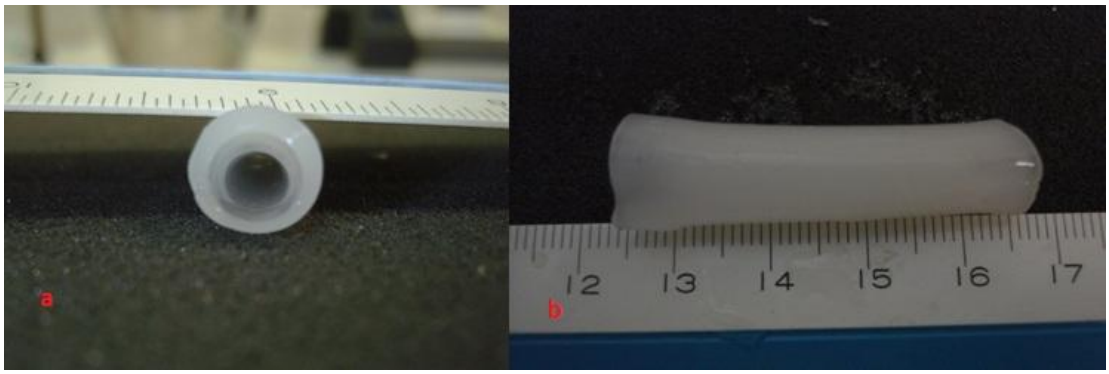

Figura 3.21: (a) Foto com vista frontal do *phantom* 3, do tipo 2, onde se observa o orifício moldado pela ponteira de micropipeta e que constituiu a parte interna do *phantom*. (b) Vista lateral do *phantom* 3.

Os *phantoms* foram dispostos, um de cada vez, em uma cuba com água em cima de um suporte. A cuba foi apoiada na plataforma citada anteriormente. O transdutor foi inserido na cuba com água para realizar a aquisição das imagens 2D.

4 RESULTADOS

4.1 PHANTOM 1

A Figura 4.1 ilustra uma das imagens do *phantom* 1, em 2D e com plano de imagem transversal ao eixo de simetria do *phantom*, obtidas por meio do BMUs. A imagem mostra claramente, em tons de cinza, o círculo referente à uma seção da parte interna do *phantom*. Algumas regiões hiperecóicas na imagem referem-se a bolhas de ar incrustradas na parte externa do *phantom*. Os sinais de eco presentes na imagem de BMUs oriundos destas bolhas estão indicados pelas setas vermelhas com a letra "A". A escala na Figura 4.1 está em centímetro.

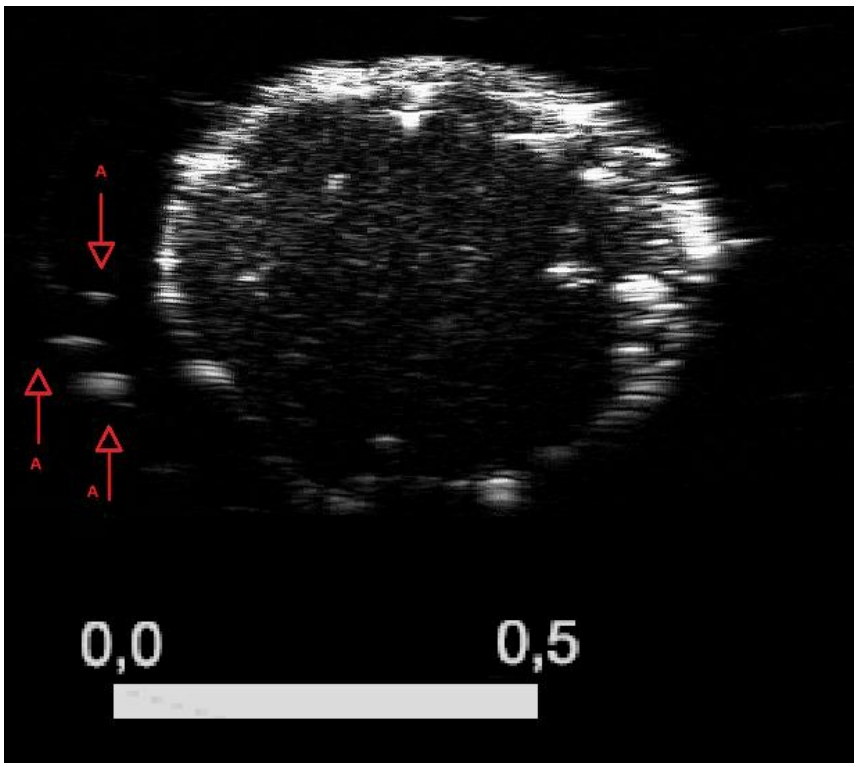

Figura 4.1: Imagem de biomicroscopia ultrassônica setorial da seção transversal do *phantom* 1. As regiões hiperecoicas, ao redor da superfície da parte interna do *phantom*, referem-se à presença de bolhas de ar incrustradas na região externa do *phantom*, indicadas pelas setas com a letra "A". A escala está em centímetros.

A Figura 4.2 ilustra a imagem, em 3D, do *phantom* 1 gerada a partir de um conjunto de imagens 2D de BMUs referentes a planos paralelos e consecutivos

(espaçados de 0,1 mm) usando-se o programa computacional que gera as imagens em 3D. Na imagem pode-se ver toda a parte interna do *phantom* em formato de um cone. Foi realizado imagens 2D de apenas uma parte do *phantom* devido à limitação de 2,54 cm do sistema de posicionamento. A faixa de cores no final da imagem é uma opção para o usuário de escolha da cor que deseja visualizar o volume. No caso das imagens desta dissertação foi utilizada a cor branca. O retângulo cinza acima da faixa de cores é a região utilizada nesta imagem do histograma. O histograma é a faixa acima do retângulo cinza. O histograma representa o número de vezes que cada valor associado aos *voxels* está presente no conjunto de dados. A escala das figuras estão em centímetro.

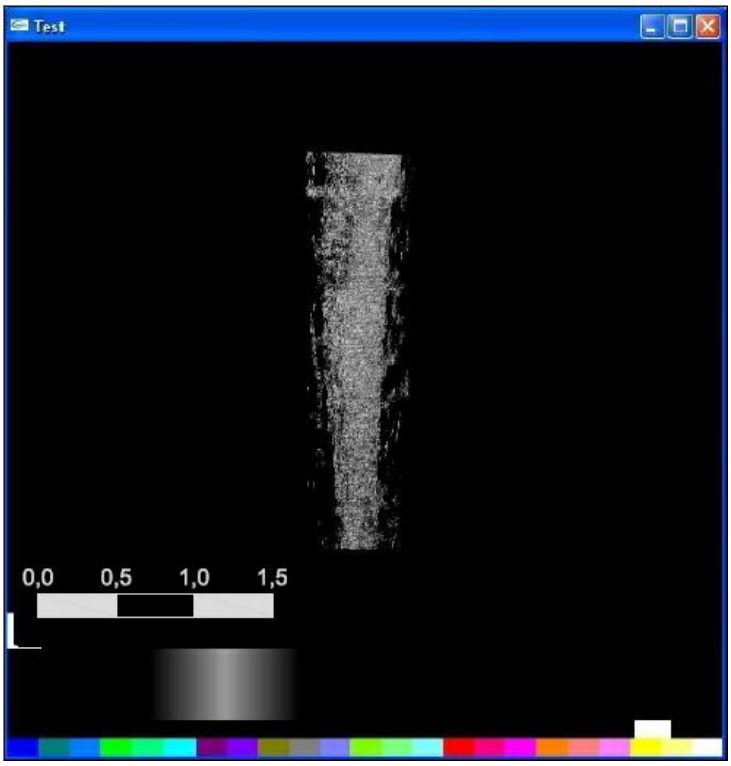

Figura 4.2: Imagem de ultrassom do *phantom* 1 em 3D gerada a partir de um conjunto de imagens 2D processadas com o programa em Python. Essa imagem mostra a parte interna, moldada de forma cônica, do *phantom*. A escala está em centímetros.

A Figura 4.3 mostra outra imagem 3D de US do mesmo *phantom* exibido na Figura 4.2, porém de outro ângulo e contendo uma vista frontal do *phantom*.

Figura 4.3: Imagem de ultrassom do *phantom* 1 em 3D contendo uma vista frontal do mesmo. A escala está em centímetros.

A Figura 4.4 também mostra outra imagem de US 3D do *phantom* 1, mas de outro ângulo. Nessa imagem pode-se observar também a forma cônica no *phantom* 1.

Figura 4.4: Imagem de ultrassom do *phantom* 1 em 3D, com vista em diagonal. Pode-se observar a forma cônica do *phantom*. A escala está em centímetros.

4.2 PHANTOM 2

A Figura 4.5 ilustra uma das imagens 2D no plano transversal obtida por meio do BMUs do *phantom* 2. Essa imagem, assim como a da Figura 4.1, mostra claramente em tons de cinza o círculo referente à uma seção do cone feito à base de uma mistura de gelatina e pó de sílica. Nessa imagem do *phantom* observam-se poucas bolhas de ar incrustradas na parte externa do *phantom*, feita apenas de gelatina, correspondentes a regiões hiperecoicas (seta vermelha).

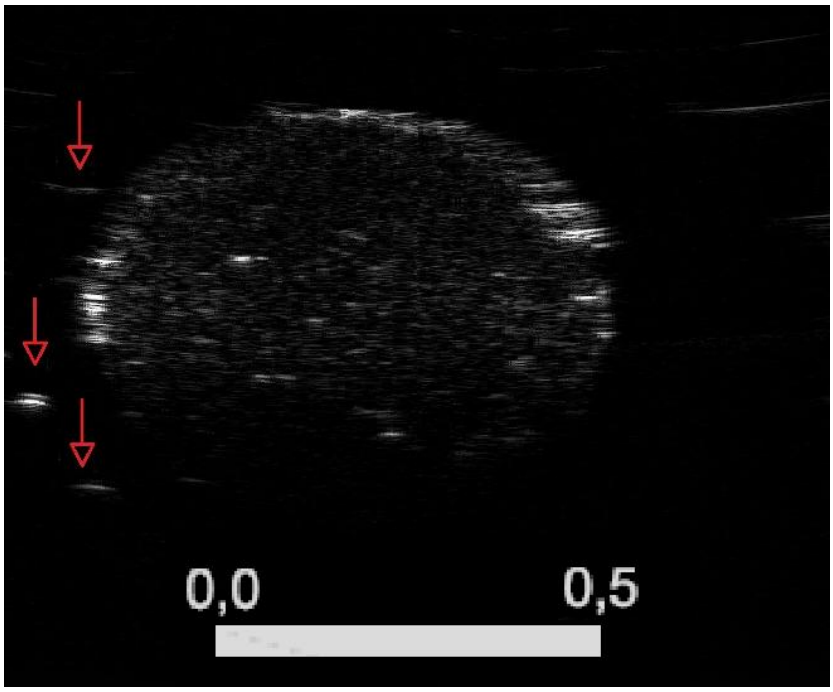

Figura 4.5: Imagem 2D do *phantom* 2 obtida por meio do biomicroscópio ultrassônico setorial. A imagem circular corresponde a um corte do *phantom* 2 transversal ao seu eixo de simetria, e as regiões hiperecoicas referem-se à bolhas de ar. Essas bolhas são indicadas na figura pelas setas vermelhas. A escala está em centímetros.

A Figura 4.6 ilustra a imagem em 3D gerada pelo conjunto de imagens em 2D coletadas em planos paralelos e separados de 0,1 mm do *phantom* 2. Nessa imagem pode-se observar toda a parte interna do *phantom* constituída de uma mistura de gelatina com pó de sílica. Esta imagem também tem o formato de um cone correspondente ao formato da ponteira de pipeta usada para moldá-lo.

Figura 4.6: Imagem 3D de ultrassom do *phantom* 2, gerada a partir de um conjunto de imagens 2D processadas com o programa em Python. Essa imagem mostra a forma cônica da parte interna do *phantom*. A escala está em centímetros.

A Figura 4.7 ilustra uma vista frontal do *phantom* 2. Observa-se o círculo da parte do *phantom* feita com gelatina e pó de sílica.

Figura 4.7: Imagem de ultrassom com vista frontal do *phantom* 2. A escala está em centímetros.

A Figura 4.8 mostra outro ângulo da imagem em 3D de US do *phantom* 2. Nesta imagem pode-se observar a forma cônica da parte interna do *phantom*. Algumas imagens estão deslocadas em relação ao eixo do *phantom* devido a um problema de sincronismo no BMUs e isto aparece como uma distorção na imagem 3D. Esta distorção aparece somente em uma vista porque ocorreu a falta de sincronismo na aquisição das imagens 2D.

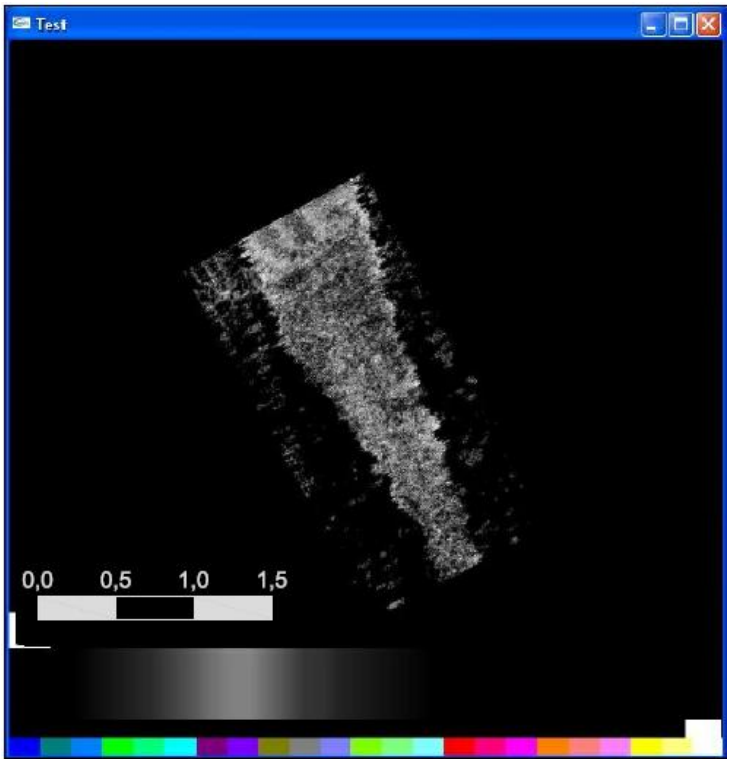

Figura 4.8: Imagem 3D de ultrassom do *phantom* 2. Nesta vista ainda pode-se observar a forma cônica da parte interna do *phantom*. A escala está em centímetros.

A Figura 4.9 ilustra uma vista lateral da imagem em 3D de *phantom* 2. Essa imagem não ficou de acordo com o esperado devido ao problema citado anteriormente com relação ao sincronismo no sistema de BMUs. Com isso, alguns planos de imagem 2D são deslocados em relação ao eixo de simetria do cone e tornam a imagem de US diferente da forma real do *phantom*. Observa-se também que a imagem como um todo está distorcida, a base do cone está deslocada para a esquerda em relação ao ponto superior do cone. Isso se deve também ao problema de sincronismo no sistema de BMUs.

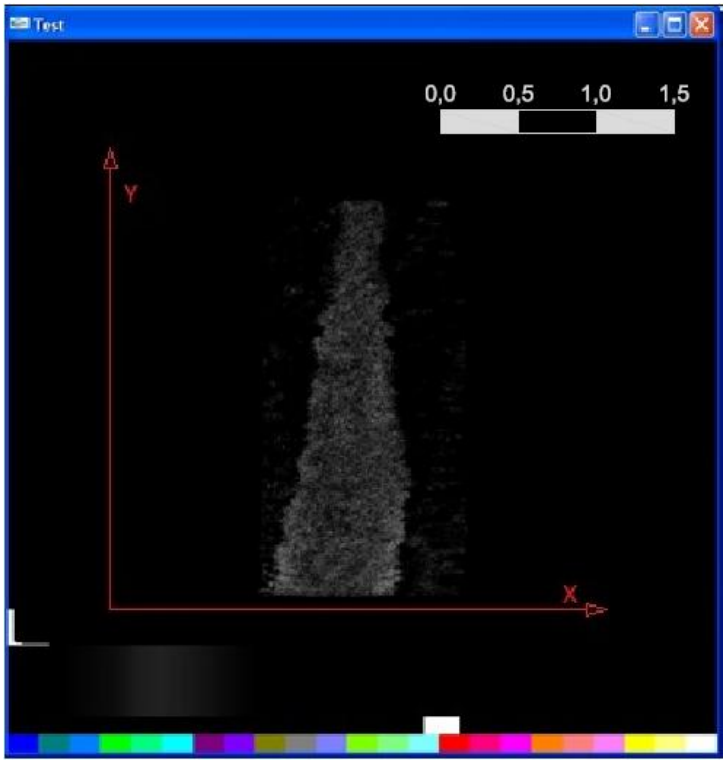

Figura 4.9: Imagem 3D de ultrassom do *phantom* 2, em vista lateral. A imagem possui uma distorção de forma devido a um problema de sincronismo do sistema de biomicroscopia ultrassônica setorial que ocasiona deslocamento dos planos 2D de imagem em relação ao eixo de simetria do cone. A escala está em centímetros.

4.3 PHANTOM 3

A Figura 4.10 ilustra uma das imagens em US 2D do *phantom* 3. Observa-se claramente a parte externa do *phantom*, formada pela mistura de gelatina com pó de sílica, que se apresenta hiperecoica e em tons de cinza. Já a parte interna apresenta-se com a forma de um círculo hipoecoico; no presente *phantom* constituído por água inserida no espaço moldado pela ponteira de uma micropipeta. A seta em vermelho com a letra “A” aponta a base de sustenção em que o *phantom* foi apoiado. Já a seta com a letra “B” aponta mais um problema gerado pela falta de sincronismo do sistema BMUs.

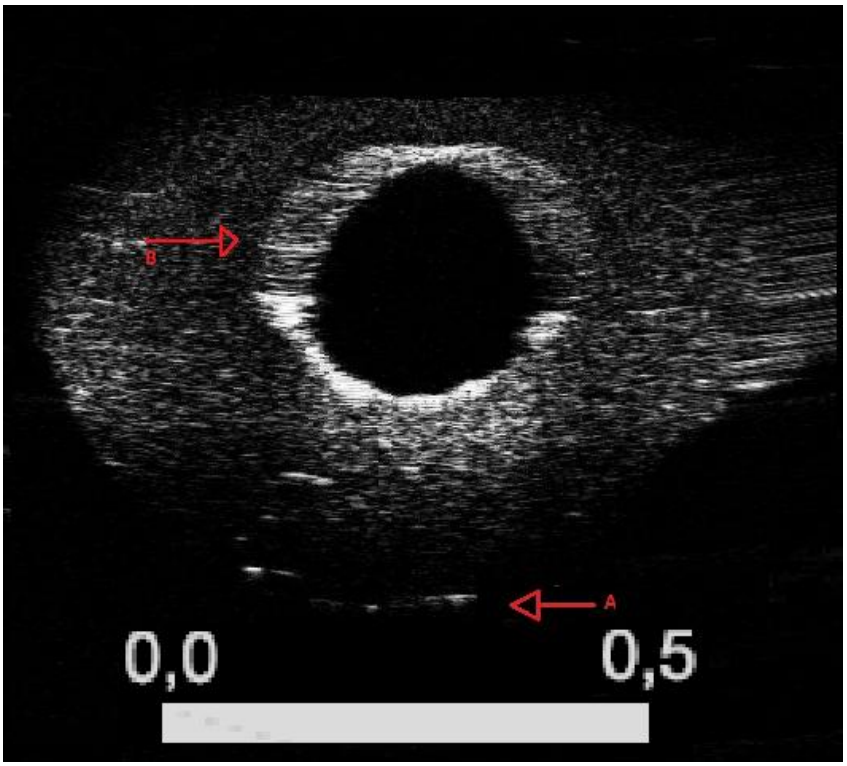

Figura 4.10: Imagem de biomicroscopia ultrassônica setorial da seção transversal do *phantom* 3. A parte externa do *phantom*, formada pela mistura de gelatina com pó de sílica, que se apresenta hiperecoica e em tons de cinza. Já a parte interna apresenta-se com a forma de um círculo hipoecoico. A seta em vermelho com a letra "A" na imagem aponta a base em que o phantom foi colocado. Já a seta com a letra "B" aponta mais um problema gerado pela falta de sincronismo do sistema biomicroscopia ultrassônica setorial. A escala está em centímetros.

A Figura 4.11 ilustra a imagem em 3D do *phantom* 3. Na imagem pode-se observar a forma cilíndrica do *phantom* formado pela parte externa. Observa-se também que em relação ao eixo Y, o *phantom* tem uma curvatura em ambos os lados, de um lado côncavo e do outro convexo, e isso se deve ao fato do *phantom* estar apoiado em uma base pelas suas extremidades e por causa de seu peso observa-se a curva.

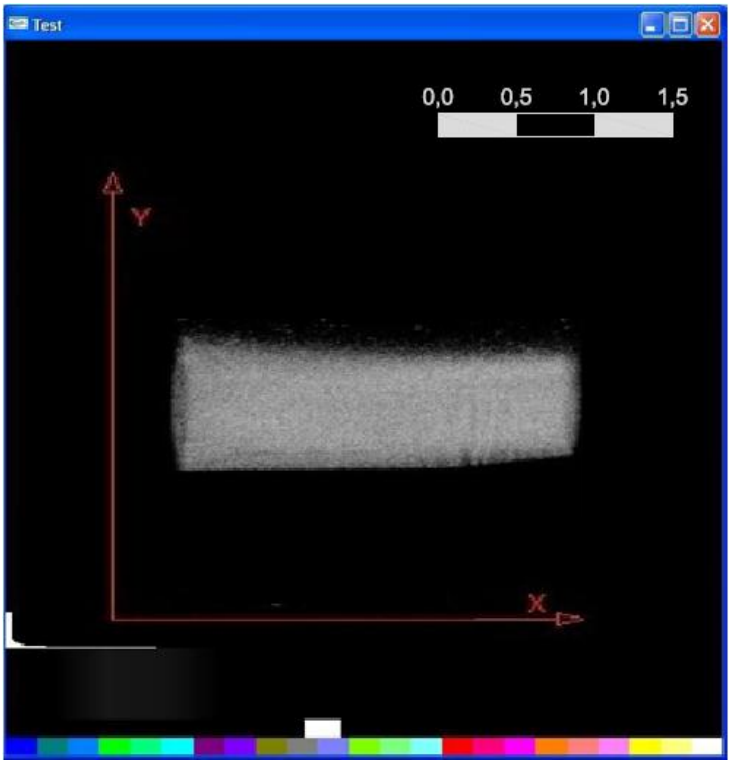

Figura 4.11: Imagem 3D de ultrassom do *phantom* 3. A imagem mostra apenas a parte exterior do *phantom*, que é hiperecoica por ser formada pela mistura de gelatina com pó de sílica. Curva em relação ao eixo X se deve ao peso do próprio *phantom* apoiado em uma base. A escala está em centímetros.

A Figura 4.12 ilustra a vista frontal da imagem em 3D do *phantom* 3. Pode-se observar o orifício, anecoico, no *phantom* que tem um formato cônico.

Figura 4.12: Imagem 3D de ultrassom do *phantom* 3 com vista frontal. Pode-se observar o orifício, anecoico, correspondente à parte interna do *phantom* que foi moldada pela ponteira (cônica) de uma micropipeta. A escala está em centímetros.

A Figura 4.13 ilustra a mesma imagem em 3D do *phantom* 3, mas vista de outro ângulo. Nesta imagem pode-se observar a região externa, hiperecoica em tons de cinza, do *phantom* e a parte interna de forma cônica e anecoica. Nesta imagem, pode-se observar o interior do phantom porque o operador do programa de geração de imagens 3D escolheu uma região diferente do histograma, utilizando *voxels* com menor opacidade.

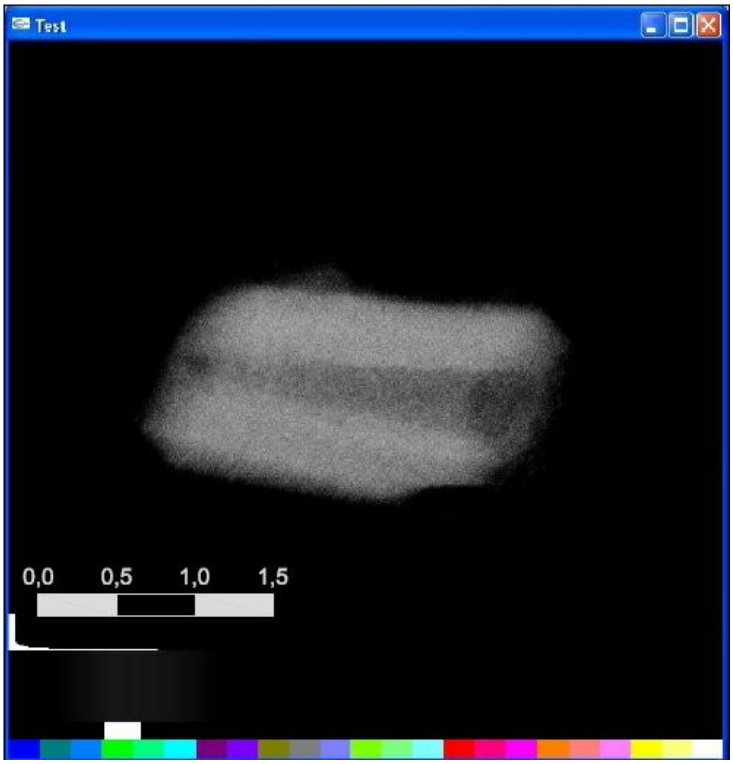

Figura 4.13: Imagem 3D de ultrassom do *phantom* 3. Nesta vista pode-se observar a forma cônica da parte interna do *phantom* que é anecoica. A escala está em centímetros.

5 DISCUSSÃO

Com esses resultados pode-se futuramente medir as dimensões de um músculo lesionado da pata de um rato antes e depois de alguma intervenção terapêutica, para conferir se houve alguma regeneração muscular e portanto alguma eficácia da terapia. Além disso, pode-se também realizar imagens em 3D de melanomas e medir suas principais dimensões para com isto definir, de forma não invasiva, o seu grau de malignidade.

Na presente dissertação foi desenvolvido um sistema de imagens 3D de BMU, baseando-se no aprimoramento de um sistema de BMUs já implementado. Basicamente, foi incorporado ao BMUs um sistema de posicionamento, em dois eixos ortogonais, de precisão para a obtenção de planos de imagens 2D de BMUs paralelos e de igual espaçamento entre si. Adicionalmente, foi incorporado ao programa computacional, em LabView, do BMUs uma parte destinada ao controle de movimento do sistema de posicionamento e outra ao armazenamento de todos os quadros de imagens 2D de BMUs correspondentes aos planos de imagens 2D de BMUs paralelos e igualmente espaçados entre si. Para a obtenção das imagens 3D, é executado um outro programa computacional que realiza a renderização volumétrica baseada em textura.

Esse sistema de BMUs-3D destina-se, no momento, à obtenção de imagens de gastrocnêmio de ratos, cujas dimensões típicas são da ordem de 20 mm de comprimento, 10 mm de largura máxima e 3,0 mm de espessura máxima. Para isso, esse sistema foi concebido para gerar imagens com comprimento e largura de 25,4 e 15 mm, respectivamente, dimensões essas compatíveis com as respectivas dimensões do músculo do animal. Já com relação à altura das imagens, a mesma depende da profundidade de campo do transdutor de ultrassom empregado. No presente caso, foi utilizado um transdutor com profundidade de campo de 1,7 mm. Com esse valor, menor do que a espessura máxima típica do músculo (aproximadamente 3,0 mm), há uma tendencia de se gerar uma imagem mais focalizada na profundidade do músculo onde está o foco do feixe de ultrassom. No entanto, esse efeito fica minimizado pela compressão logarítmica dos sinais de eco antes da construção da imagem.

Com o intuito de testar a capacidade do sistema de BMUs-3D para gerar imagens de estruturas com dimensões compatíveis com aquelas do músculo

gastrocnêmio, foram realizados testes experimentais para a obtenção de imagens de *phantoms* com dimensões próximas daquelas do músculo. Nesse caso, para os *phantoms* 1 e 2, do tipo 1, a parte cônica foi moldada com ponteira de micropipeta menor cujo diâmetro externo varia de 0,85 a 6,9 mm entre ambas as extremidades. Já o *phantom* 1, do tipo 2, foi moldado com ponteira de micropipeta maior cujo diâmetro externo varia de 1,6 a 9,6 mm. Com o "empilhamento" de cada conjunto de imagens (254) 2D tomadas em planos paralelos, espaçados de 0,1 mm, e transversais ao eixo longitudinal de cada tipo de *phantom*, foram geradas as imagens em 3D ilustradas nas Figuras 4.2, 4.6 e 4.13, nas quais pode-se observar claramente o formato cônico das partes internas dos *phantoms*, moldadas por ponteiras de micropipetas.

Nas Figuras 5.1 e 5.2 foram superpostos os traçados, em vermelho e na mesma escala de dimensão da imagem do *phantom*, ilustrando o formato das ponteiras de micropipeta usadas para moldar a parte interna dos *phantoms* 1 e 2, respectivamente. Os traçados das ponteiras de micropipeta sobrepostos nas imagens em 3D dos *phantoms* demonstram uma concordância entre as dimensões da imagem e da ponteira real. Observa-se, ainda, que o formato da parte interna, cônica, dos *phantoms* 1 e 2 segue razoavelmente o da ponteira usada para a moldagem. Além disso, observa-se que a imagem do *phantom* da Figura 4.2 tem dimensões variando entre 2,5 e 7,45 mm, o que confirma a capacidade do sistema de BMUs-3D em termos de resolução espacial adequada para gerar imagens de estruturas com dimensões de até 10 mm, compatível com as dimensões do músculo.

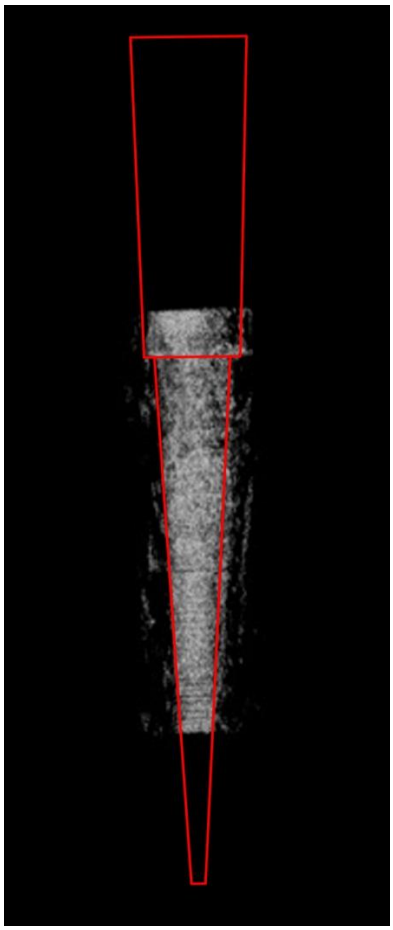

Figura 5.1: Sobreposição do contorno (em vermelho) da ponteira de micropipeta usada para moldar o *phantom* 1 com sua imagem 3D de ultrassom. Observa-se um casamento bem próximo entre o formato da parte interna do *phantom* 1 e o da ponteira usada para moldá-la.

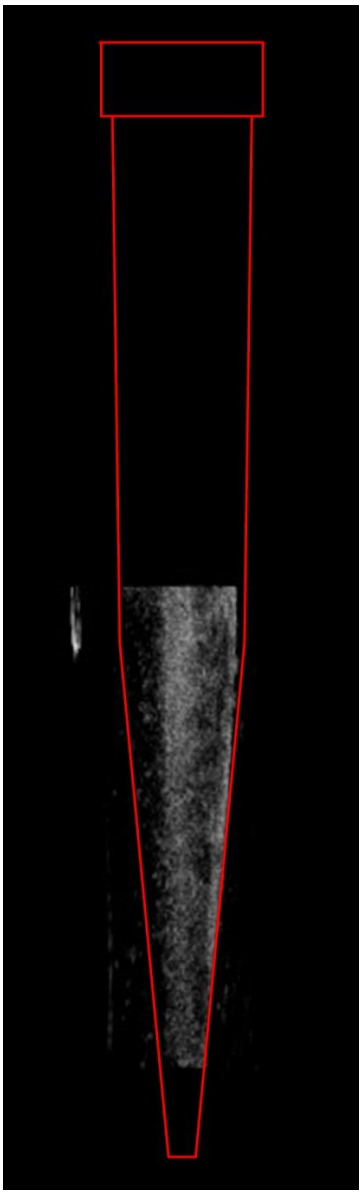

Figura 5.2: Sobreposição do contorno (em vermelho) da micropipeta usada para moldar o *phantom* 2 com sua imagem 3D de ultrassom. Observa-se um casamento bem próximo entre o formato da parte interna do *phantom* 2 e o da ponteira usada para moldá-la.

A geração de imagens 3D de US envolve custo computacional e arquitetura de máquina superiores ao que se consegue com a geração de imagens 2D. Isto gera uma argumentação em desfavor ao emprego clínico do US-3D levando-se em conta a relação custo-benefício para a geração de imagens 3D *versus* 2D. Uma revisão das potencialidades do uso clínico de imagens 3D-US foi apresentada por Nelson e Pretorious (1998) onde vários dos aspectos discutidos por esses autores são atuais em termos de justificativa de emprego clínico e do que se faz para o aprimoramento da técnica de geração de imagens 3D-US.

Uma das grandes vantagens na geração das imagens 3D-US está na possibilidade de determinar o volume de determinados órgãos ou lesões. Isso tem implicações, por exemplo, no planejamento de tratamento como também na monitoração da resposta de tumores à terapia. Com relação ao músculo, a ressonância magnética (RM) é considerada o padrão ouro para a medição direta de volume e comprimento muscular, in vivo. No entanto, além do seu alto custo esta modalidade de imagem necessita um tempo elevado para realizar a aquisição das imagens e em alguns casos é necessária a sedação do paciente. Assim, a determinação direta de imagem 3D-US torna-se mais favorável em relação ao emprego de RM.

Um dado importante a ser considerado na obtenção de imagens de US-3D é o tempo necessário para a geração das mesmas. No passado, esse tempo chegou a situar-se entre 30 e 40 minutos para a geração de imagens de US-3D do sistema pelvicalicial *in vitro* (GHANI *et al*., 2008). Nesse caso a geração das imagens 3D baseava-se em duas etapas distintas: a aquisição dos quadros de imagem 2D, a reconstrução da imagem 3D e sua posterior visualizção. Hoje, é possível realizar a reconstrução e a visualização de volume simultaneamente com a aquisição de dados. Para isso, Dai et al. (2010) utilizaram um sistema de microcomputador pessoal com processador Intel Core2 1.86 GHz e uma GPU NVIDIA GeForce 8800 GT encarregada do processamento dos sinais para as etapas de reconstrução e de renderização volumétrica, amabas por incremento de volume e a renderização. Esses autores conseguiram imagens de US-3D com dimensões de 256×256×256 *voxels*, a partir de imagens 2D de 480×413 *pixels*, numa taxa de 10,5 imagens por segundo.

O tempo de aquisição/armazenamento das 254 imagens 2D foi de 8 minutos e o tempo para reconstrução e visualização de volume foi de 1 minuto. Uma vez gerada a imagem 3D, a rotação da mesma para visualização em diferentes ângulos é praticamente instantânea. Nota-se portanto que para esse sistema de BMUs-3D o ponto crítico no tempo total para a geração de imagens 3D é a aquisição/armazenamento dos quadros de imagens 2D. Mesmo assim, esse intervalo de tempo da ordem de 8 minutos não compromete a utilização prevista para o sistema de BMUs-3D, uma vez que o animal estará anestesiado por um período que pode chegar até 40 minutos.

Já com relação a outros trabalhos relacionados com aplicações de BMU-3D, a literatura relata o seu uso, *in vivo*, para a caracterizção do corpo ciliar e a iris posterior,

de forma a quantificar alterações de configuração do corpo ciliar durante sua acomodação (STACHS *et al.*, 2002). Nesse trabalho, os autores usaram um equipamento comercial de BMU (Modelo 840; Humphrey Instruments, Carl Zeiss Group, Alemanha) com um sistema de posicionamento controlado por computador, para realizar a obtenção de imagens 2D em planos paralelos compreendidos numa varredura com 2,5 mm de comprimento. Cada quadro de imagem de BMU foi gerado com 256x1024 *pixels*, que depois foram convertidos para 440x440 *pixels* após um sistema de captura de imagem. Foi utilizado um software comercial para a renderização de volume e os autores não especificaram o tempo de processamento para a geração das imagens 3D. mesmo assim, pode-se obervar que eles utilzaram uma quantidade de dados muito menor daquela usada para a geração das imagens de BMUs-3D do presente trabalho.

Outra utilização de imagens de BMU-3D foi relatada por Shemesh *et al*. (2007) para a descrição da morfologia 3D de biofilmes durante o crescimento, *in vitro*, de *Streptococcus mutans*. Para o referido trabalho os autores usaram um equipamento comercial de BMU (Vevo 770; Visualsonics, Toronto, Canadá) operando em 55 MHz. A geração de imagens 3D foi baseada na captura de imagens 2D, usando um sistema motorizado para deslocar o transdutor numa diração perpendicular ao plano de imagem, e os dados para uma imagem de 8x8x10mm foram coletados em 10 segundos de varredura. Os autores não informaram o tempo de processamento necessário para a renderização da imagem volumétrica. Esse trabalho, assim como o anterior, tratou de investigar uma região de interesse com volume menor do que se pretende analisar com o sistema de BMUs-3D. Com isso, nesse dois últimos trabalhos citados a geração de imagens 3D envolveu uma quantidade muito menor de dados, quando comparado com as imagens geradas pelo BMUs-3D, para a geração das imagens de BMU-3D.

Não foram encontrados, na literatura, trabalhos relacionados com a obtenção de imagens de BMU 3D para pequenos animais e suas estruturas musculares. Foi apenas encontrado o relato da determinação do volume de músculos de cães, *in vivo*, usando US-3D com a técnica de varredura a mão livre para a obtenção de múltiplas imagens 2D (WELLER *et al*., 2007). Esses autores compararam seus resultados para o volume muscular obtidos por US-3D com os resultados baseados em tomografia computadorizada e concluíram que o US-3D gera resultados precisos e acurados, sendo

portanto uma técnica inovativa que permite a determinação objetiva de volume muscular *in vivo*.

Sendo assim, não foi possível comparar as características do atual sistema com outras de trabalhos distintos e envolvendo a obtenção de imagens volumétricas de órgãos de pequenos animais. Mesmo assim, de acordo com as dimensões dos *phantoms* usados no presente trabalho, o sistema de BMUs-3D está rpeparado para a geração de imagens de BMU-3D compatíveis com dimensões do músculo gastrocnêmico de ratos.

As imagens obtidas com o sistema de BMUs-3D não passaram por nenhum processamento. Conforme Forsberg *et al*. (2010) é possível melhorar a qualidade das imagens através de técnicas de processamento baseadas em filtragem adaptativa não-estacionária para a remoção de speckle e ruído, enquanto mantendo as estruturas das imagens.

Além da possibilidade de melhorar a qualidade das imagens através técncicas de processamento, trabalhos futuros com o sistema de BMUs-3D deverão incluir a segmentação e detecção de bordas, para com isso chegar-se à determinação do volume de uma determinada região de interesse, como é o caso da medição volumétrica do músculo do gastrocnêmio de ratos submetidos em um processo de regeneração tecidual após lesão do músculo.

Comparando-se a forma de como os resultados obtidos no presente trabalho em relação ao que foi relatado na literatura, observa-se que o sistema de BMUs-3D está levando um tempo muito maior para gerar as imagens de BMU-3D. Para esse tempo diminuir deverá haver uma modificação substancial do sistema de varredura mecânica do transdutor de ultrassom para a obtenção dos múltiplos planos de imagens 2D paralelos.

6 CONCLUSÃO

Diante dos resultados obtidos nesta dissertação nota se que o sistema de BMUs em conjunto com os programas de aquisição de imagens em 2D e o de geração de imagens em 3D é capaz de gerar imagens em US em 3D de *phantoms* para biomicroscopia ultrassônica, com formas e dimensões relativas calibradas.

7 REFERÊNCIAS BIBLIOGRÁFICAS

BAUM, G., GREENWOOD, I., 1961, “Orbital lesion localization by three dimensional ultrasonography”, New York State Journal of Medicine, 61, 4149-4157.

CHANG, R. F., WU, W. J., CHEN, D. R., CHEN, W. M., SHU, W. LEE, J. H., JENG, L. B., 2003, “3-D US Frame Positioning using Speckle Decorrelation and Image Registration”, Ultrasound in Medicine and Biology, v. 29, n. 6, pp. 801-812.

CHIOU, A., CHIU, B., OPPAT W., MATSUMURA, J. S., CHISHOLM, R. L., PEARCE, W. H., 2000, “Transrectal ultrasound assessment of murine aorta and iliac arteries.” Journal of Surgical Research; 88:193-199.

COBBOLD, R. S. C., 2007, Foundations of Biomedical Ultrasound, 1 ed., New York, New York, Oxford University Press.

DAI, Y., TIAN, J., DONG, D., YAN, G., ZHENG, H., 2010, "Real-Time Visualized Freehand 3D Ultrasound Reconstruction Based on GPU", IEEE Transactions On Information Technology In Biomedicine, v. 14, n. 6.

FENSTER, A., DOWNEY, D.B., CARDINAL, H. N., 2000, “Three-dimensional ultrasound imaging” Physics in Medicine and Biology 2001; 46: R-67 – R-99

FISH, P. 1990, “Physics and Instrumentation of Diagnostic Medical Ultrasound”, 1 ed., New York, New York, John Wiley & Sons.

FORSBERG, F., BERGHELLA, V., MERTON, DA., RYCHLAK, K., MEIERS, J., GOLDBERG, BB., 2010, "Comparing Image Processing Techniques for Improved 3-Dimensional Ultrasound Imaging", Journal of Ultrasound in Medicine; v.29, pp.615–619.

FOSTER, F. S., PAVLIN, C. J., HARASIEWICZ, K. A., CHRISTOPHER, D. A., TURNBULL, D. H., 1999, “Advances in Ultrasound Biomicroscopy”, Ultrasound in Medicine and Biology, v. 26, n. 1, pp. 1-27.

FRY, N. R., GOUGH, M., SHORTLAND, A. P., 2003, “Three-dimensional realization of muscle morphology and architecture using ultrasound”, Gait and Posture, 20, pp. 177-182.

GHANI, KR., PILCHER, J., PATEL, U., ROWLAND, D., NASSIRI, D., ANSON, K., 2008, “Three-Dimensional Ultrasound Reconstruction of the Pelvicaliceal System: An In-Vitro Study”, World Journal of Urology, v.26, pp.493–498.

HUANG, Q. H., ZHENG, Y. P., 2007, “Volume reconstruction of freehand three-

dimensional ultrasound using median filters", Ultrasonics, 48, pp. 182-192.

HUFF, R., 2006, "*Recorte Volumétrico Usando Técnicas de Interação 2D e* 3D" Dissertação de Msc, Instituto de Informática, Programa de Pós-Graduação em Computação, UFRGS, Porto Alegre, RS, Brasil.

IKITS, M., KNISS, J. M., LEFOHN, A., HANSEN, C. D., 2004, "Volume Rendering Techniques", Addison Wesley, p 667-6692

JOLLY, C., JEANNY, J., BEHAR-COHEN, F., LAUGIER, P., SAIED, A., 2005, "High-resolution ultrasonography of subretinal structure and assessment of retina degeneration in rat." Experimental Eye Research; 81:592-601.

KINDLMANN, G. L., DURKIN, J. W., 1998, "Semi-Automatic Generation of Transfer Functions for Direct Volume Rendering", Symposium on Volume Visualization, VVS, p79-86.

LACROUTE, P., LEVOY, M., 1994, "Fast Volume Rendering Using A Shear-Warp Factorization Of The Viewing Transformation", Computer Graphics, v. 28, n. 4, pp.451-458.

LEVOY, M., 1988, "Display of Surfaces from Volume Data", IEEE Computer Graphics and Applications, v. 5, n. 3, pp. 29-37.

MCCANN, HA., SHARP, JC., KINTER, TM., MCEWAN, CN., BARILLOT, C., GREENLEAF, JF., 1988, "Multidimensional Ultrasonic Imaging for Cardiology", Proceedings of IEEE, v.76, pp.1063-1073. In: SOLBERG, O. V., LINDSETH, F., TORP, H., BLAKE, R.E., HERNES, T. A. N., 2007, "Freehand 3D Ultrasound Reconstruction Algorithms – A Review", Ultrasound in Medicine and Biology, v. 33, n. 7, pp. 991-1009.

NELSON, TR., PRETORIUS, DH., 1998, "Three-dimensional ultrasound imaging", Ultrasound in Medicine & Biology, v. 24, n. 9, pp. 1243–1270.

PEIXINHO, C. C., RIBEIRO, M. B., RESENDE, C. M. C., WERNECK-DE-CASTRO, J. P. S., DE OLIVEIRA, L. F., MACHADO, J. C., 2011, "Ultrasound biomicroscopy for biomechanical characterization of healthy and injured triceps surae of rats", The Journal of Experimental Biology, vol. 214, pp. 3880-3886.

REZK-SALAMA, C., ENGEL, K., BAUER, M., GREINER, G., ERTL, T., 2000, "Interactive Volume Rendering on Standard PC Graphics Hardware Using Multi-Textures and Multi-Stage Rasterization", ACM Siggraph/Eurographics Workshop

on Graphics Hardware, p 109-118.

ROHLING, R., GEE, A., BERMAN, L., A, 1999, “Comparison of Freehand Three-Dimensional Ultrasound Reconstruction Techniques”, Medical Image Anal, v.3, pp.339-359. In: SOLBERG, O. V., LINDSETH, F., TORP, H., BLAKE, R.E., HERNES, T. A. N., 2007, “Freehand 3D Ultrasound Reconstruction Algorithms – A Review”, Ultrasound in Medicine and Biology, v. 33, n. 7, pp. 991-1009.

RYAN, L. K. e FOSTER, F. S., 1997, "Tissue Equivalent Vessel Phantoms for Intravascular Ultrasound", Ultrasound in Medicine and Biology, v. 23, n. 2, pp. 261-273.

SCHEIPERS, U., KOPTENKO, S., REMLINGER, T., FALCO, T., LACHAINE, M., 2010, “3-D Ultrasound Volume Reconstruction Using the Direct Frame Interpolation Method”, IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control, v. 57, n. 11 (Nov), pp. 2460-2470.

SHEMESH, H., GOERTZ, DE., VAN DER SLUIS, LWM., DE JONG, N., WU, MK,. WESSELINK, PR, 2007, "High Frequency Ultrasound Imaging of a Single-Species Biofilm", Journal of Dentistry v.35, pp. 673 – 678.

SHEREBRIN, S., FENSTER, A, RANKIN, R. K., SPENCE, D., “Freehand three-dimensional ultrasound: Implementation and applications”, Physics of Medical Imaging, SPIE, 1996; v. 2708: 296 – 303.

SOLBERG, O. V., LINDSETH, F., TORP, H., BLAKE, R.E., HERNES, T. A. N., 2007, “Freehand 3D Ultrasound Reconstruction Algorithms – A Review”, Ultrasound in Medicine and Biology, v. 33, n. 7, pp. 991-1009.

STACHS, O., MARTIN, H., KIRCHHOFF, A., STAVE, J., TERWEE, T., GUTHOFF, R., 2002, "Monitoring accommodative ciliary muscle function using three-dimensional ultrasound", Graefe's Archive for Clinical and Experimental Ophthalmology v. 240, pp.906–912.

TROBAUGH, JW., TROBAUGH, DJ., RICHARD, WD., 1994, “Three-dimensional Imaging with Stereotactic Ultrasonography”, Computerized Medical Imaging and Graphics, v.18, pp.315-323. In: SOLBERG, O. V., LINDSETH, F., TORP, H., BLAKE, R.E., HERNES, T. A. N., 2007, “Freehand 3D Ultrasound Reconstruction Algorithms – A Review”, Ultrasound in Medicine and Biology, v. 33, n. 7, pp. 991-1009.

TURNBULL, D. H., FOSTER, F. S., 2002, “In vivo ultrasound biomicroscopy in developmental biology.” Trends in Biotechnology; 20:S29-S33.

WEBB, S. 1988, “The Physics of Medical Imaging” Bristol: Institute of Physics Publishing.

WEISKOPF, D., ENGEL, K., ERTL, T., 2002, “Volume Clipping via Per Fragment Operations in Texture-Based Volume Visualization”, IEEE Conference on Visualization, pp. 93-100.

WELLER, R., PFAU, T., FERRARI, M., GRIFFITH, R., BRADFORD, T., WILSON, A., 2007, “The Determination of Muscle Volume with a Freehand 3D”, Ultrasound in Medicine & Biology, v. 33, n. 3, pp. 402–407.

WESTOVER, L., 1990, “Footprint Evaluation for Volume Rendering”, Computer Graphics (SIGGRAPH ’90 Proceedings), v. 24, n. 4, pp. 367-376.

WITTENBRINK, C., MALZBENDER, T., GOSS, M., 1998, "Opacity-Weighted Color Interpolation for Volume Sampling", *IEEE Symposium on Volume Visualization*, pp. 135–142.

8 ANEXOS

8.1 ANEXO I

Este anexo é basicamente um manual de usuário para o programa de geração de imagens 2D escrito em LabVIEW.

Ao clicar no ícone do arquivo do programa de geração de imagens 2D chamado "Geração de Imagens 2D" abre-se a tela de comandos do BMUs e visualização das imagens. Após o usuário clicar na seta abaixo do menu 'Edit' para executar o programa, a tela (Figura 8.1) para a configuração de parâmetros como 'DOF' e 'df' aparece. Ao terminar a configuração o usuário deve clicar em 'CONFIRMA'.

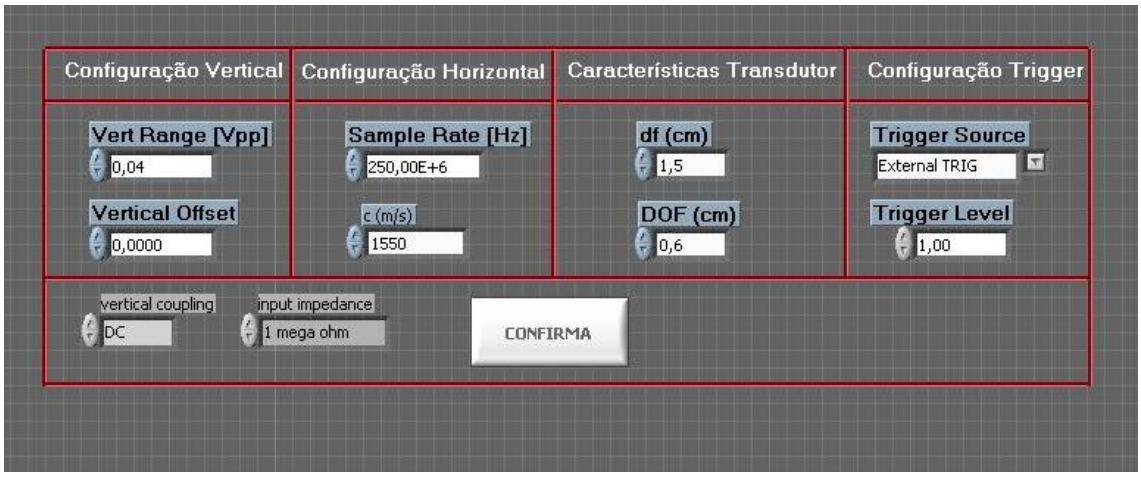

Figura 8.1: Tela de configuração dos parâmetros iniciais do programa em LabVIEW usado no biomicroscópio ultrassônico setorial

Assim que o usuário clicar em 'CONFIRMA' uma segunda tela irá aparecer, como na Figura 8.2 onde pode-se observar em tempo real, a imagem em 2D de BMU do meio em análise. Para controlar o contraste e o nível de ruído existem 2 barras de controle mostradas na Figura 8.3.

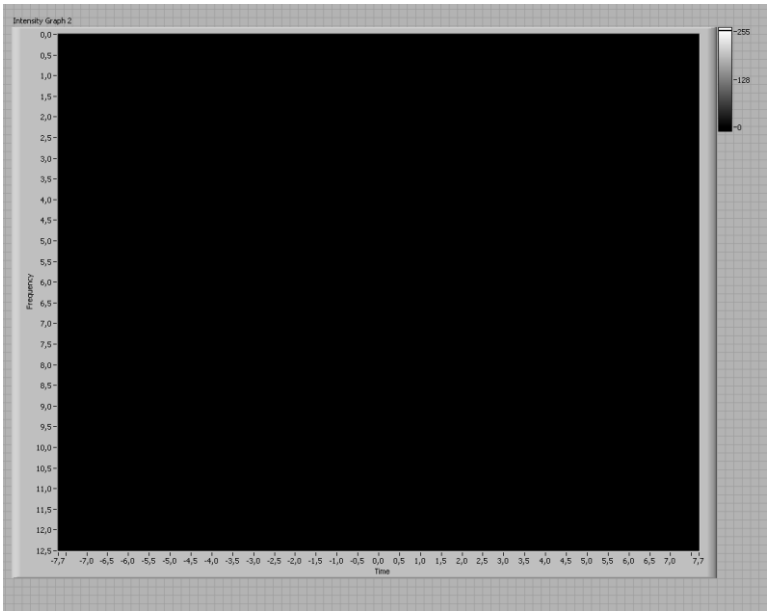

Figura 8.2: Display de visualização da imagem

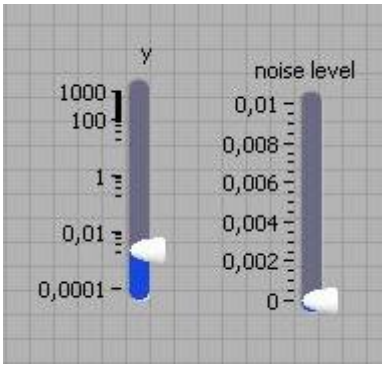

Figura 8.3: Controles de contraste e ruído da imagem de BMUs.

Já a tela da Figura 8.4 mostra os controles principais para a aquisição das imagens 2D de BMUs. O primeiro passo é clicar em ‘MOTOR ON’ para habilitar o motor 1 (eixo X) do sistema de posicionamento. No parâmetro 'Eixo' passar para ‘2’ e clicar novamente em ‘MOTOR ON’ para habilitar o motor 2 (eixo Y). A seguir deve-se clicar em ‘JOYSTICK ON’ para habilitar o Joystick (o LED em verde deverá acender para indicar que o Joystick está habilitado). Com o Joystick é possível realizar uma varredura da região desejada para a aquisição das imagens 2D de BMU. Ao ser escolhida o primeiro ponto ao longo do eixo Y com o Joystick, deve-se clicar em ‘DEFINIR POSIÇÃO INICIAL’. Após essa definição, deve-se chegar na posição que se deseja realizar o último ponto ao longo do eixo Y com o Joystick e clicar em ‘DEFINIR POSIÇÃO FINAL’. Depois de realizar estas definições, pode-se escolher a distância

entre cada plano de imagem que será adquirida em ‘Passo’ e o tempo de espera que será seguido entre cada aquisição de imagem (este tempo é necessário apenas para proteger o sistema de posicionamento e para adquirir uma imagem com o sistema de posicionamento parado). Com a seleção do tamanho do passo, o número de passos será calculado pelo programa. O usuário também deve definir o diretório em que as imagens 2D de BMU serão armazenadas em ‘Caminho para as imagens’ e também o nome que será dado aos arquivos das imagens em ‘nome da imagem’. O ‘fator’ se refere à qualidade da imagem, o valor 3 pré determinado mostra uma imagem com menos *pixels* que a original, mas sem perda de qualidade. O parâmetro ‘ângulo’ deve ser determinado também pelo usuário, e indica o ângulo de abertura do setor varrido pelo feixe do transdutor durante a varredura. O valor máximo do ângulo é de 14,6°.

Ao se definirem tais parâmetros o usuário deve clicar em ‘VOLTAR PARA POSIÇÃO INICIAL’ e em seguida em ‘ADQUIRIR’ para começar a aquisição das imagens 2D. O tempo de processamento do programa varia de acordo com o número de imagens a serem adquiridas e o tempo de espera de aquisição entre cada imagem.

Durante a execução do programa “Geração de Imagens em 2D”, são geradas as escalas para a imagem 3D. No caso atual, largura (eixo X), profundidade (eixo Y) e altura (eixo Z). A largura é determinada em função do ângulo, da distância focal e da DOF. A profundidade é determinada em função do número de planos de imagens 2D e do passo entre eles. Por último, a altura é determinada em função da DOF. Os valores para essas escalas, normalizados pela profundidade são apresentados na tela da Figura 8.4.

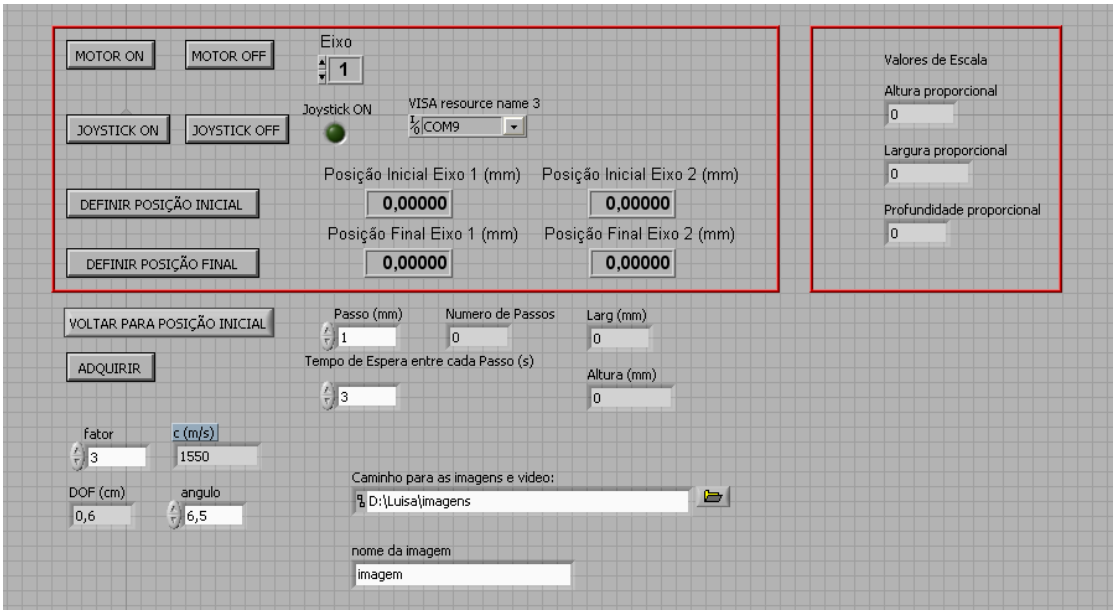

Figura 8.4: Tela de configuração de alguns parâmetros e controle do sistema de posicionamento, do ângulo de varredura do feixe de ultrassom e de armazenagem das imagens 2D de biomicroscopia ultrassônica.

8.2 ANEXO II

Este anexo é basicamente um manual de usuário para executar o programa de geração de imagens 3D escrito em Python.

Ao clicar no ícone do arquivo do programa “Geração de Imagens 3D” abre-se uma janela, mostrada na Figura 8.5. O usuário deve digitar a dimensão normalizada para a largura da imagem 3D, apresentada na tela da Figura 8.4 ao se executar o programa “Geração de Imagens em 2D”. De forma análoga, o usuário deve também digitar as escalas normalizadas para a altura e depois para a profundidade. As Figuras 8.6 e 8.7 mostram, respectivamente, as janelas para a entrada dos valores normalizados da altura e da profundidade e que devem ser digitados pelo usuário.

Figura 8.5: Janela de digitação da largura, normalizada, para a imagem 3D.

Figura 8.6: Janela de digitação da altura, normalizada, para a imagem 3D.

Figura 8.7: Janela de digitação da profundidade, normalizada, para a imagem 3D.

Após o usuário digitar os três parâmetros ilustrados na Figura 8.5 a janela (Figura 8.8) com o diretório contendo os arquivos do conjunto das imagens em 2D geradas pelo LabVIEW deve ser selecionado e clicar em ‘OK’.

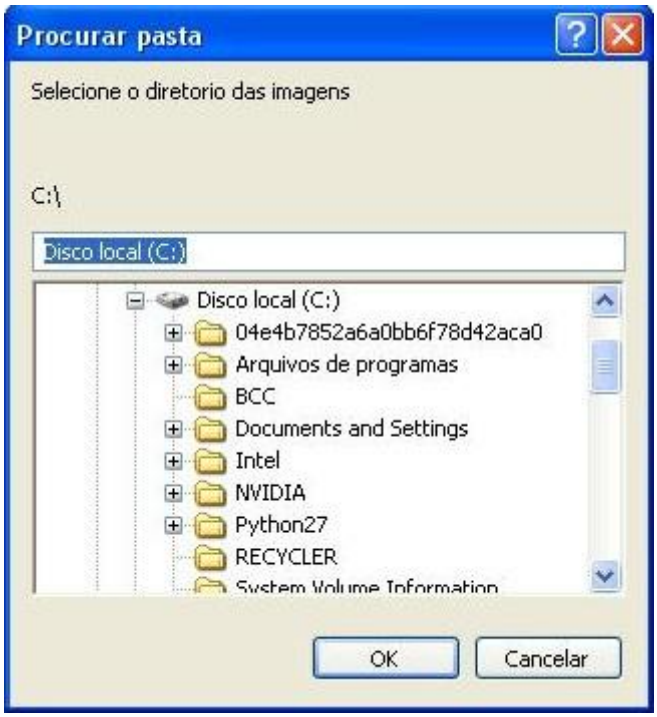

Figura 8.8: Janela para a seleção do diretório contendo os arquivos das imagens 2D usadas para compor uma imagem 3D.

Após a seleção do diretório das imagens 2D o usuário deve digitar o número da última imagem 2D que ele gostaria de incluir na geração da imagem 3D. Como exemplo tem-se o nome da última imagem 2D gerada pelo programa LabVIEW é “imagem245”, o número 245 deve ser digitado e logo após deve-se clicar em’OK’. A seguir aparee uma janela (Figura 8.9) onde deve-se digitar o nome dado pelo usuário aos arquivos das imagens 2D geradas pelo programa LabVIEW. No caso do exemplo “imagem245”, o usuário deve digitar “imagem” na janela da Figura 8.10. Assim, que o usuário clicar em ‘OK’ e esperar alguns segundos a imagem 3D será gerada.

Figura 8.9: Janela de digitação do número da última imagem.

Figura 8.10: Janela de digitação do nome do arquivo contendo as imagens 2D de biomicroscopia ultrassônica.

Abaixo está a programação em Python. O primeiro programa, chamado pelo programa principal, realiza a renderização do volume. Já o segundo programa é o programa principal.

PROGRAMA DE RENDERIZAÇÃO DO VOLUME

```
import Image

def readVolumeRaw(nx,ny,nz,filename):
    """Reads volume data from a binary 'raw' file, i.e., with no header.
    nx,ny and nz contain the dimensions of the voxel grid and
    filename the name of the file. Returns a 3D array of bytes.
    """

    data = fromfile(filename, dtype='u1')
    assert(data.shape == (nx*ny*nz,))
    data.shape = (nz,ny,nz)
    return data


def readVolumeFromImages(firstslice, lastslice, filenamepattern):
    """Reads volume data from a series of image files with names
    obtained by using filenamepattern as a format
    for the slice number. Thus, if filenamepattern is "file%d.jpg",
    and firstslice and lastslice are 1 and 10, respectively, then
    files "file1.jpg" thru "file10.jpg" are generated and read from.
```

```
        All images should have the same size.

        The contents of all images is stored into a 3D array of bytes
        which is returned.
        """

        # Open the first image to get the width and height
        fname = filenamepattern%firstslice
        im = Image.open(fname)
        nx, ny = im.size[0],im.size[1]

        # Allocate array for the texture
        data = zeros((lastslice-firstslice+1,ny,nx), dtype='u1')

        # Read the slices from several files
        for i in range(firstslice,lastslice+1):
            filename = filenamepattern%i
            im = Image.open(filename)
            assert (nx==im.size[0] and ny==im.size[1])
            imgdata = array(tuple(im.getdata()), dtype='u1')
            print filename
            imgdata.shape = (nx, ny)
            data[i-firstslice]=imgdata

        # Return the read volume
        return data

    def readVolumeFromColorImages(firstslice, lastslice, filenamepattern,
resize=False):
        """Reads volume data from a series of image files with names
        obtained by using filenamepattern as a format
        for the slice number. Thus, if filenamepattern is "file%d.jpg",
        and firstslice and lastslice are 1 and 10, respectively, then
        files "file1.jpg" thru "file10.jpg" are generated and read from.

        All images should have the same size.

        The contents of all images is stored into a 3D array of bytes
        which is returned.

        The resize flag, if true, will reduce the images so that no
        dimension will be greater than 512.
        """

        # Open the first image to get the width and height
        fname = filenamepattern%firstslice
        im = Image.open(fname)
        nx, ny = im.size[0],im.size[1]
```

```
    # Obtain new dimensions if resizing
    if resize:
        if nx>512: nx = 512
        if ny>512: ny = 512
        if (nx,ny) == (im.size[0], im.size[1]): resize = False

    # Allocate array for the texture
    data = zeros((lastslice-firstslice+1,nx,ny), dtype='u1')

    # Read the slices from several files
    for i in range(firstslice,lastslice+1):
        filename = filenamepattern%i
        print filename
        im = Image.open(filename)
        if resize: im = im.resize((nx,ny), Image.BILINEAR)
        assert (nx==im.size[0] and ny==im.size[1])
        #lista = [(r,) for r,g,b in tuple(im.getdata())]
        #imgdata = array(lista, dtype='u1')
            imgdata = array (tuple(im.convert("L").getdata()), dtype='u1')
        imgdata.shape = (nx, ny)
        data[i-firstslice]=imgdata

    # Return the read volume
    return data

PROGRAMA PRINCIPAL
from numpy import *
from OpenGL.GL import *
from OpenGL.GLUT import *
import Image
from transferfunction import TransferFunction
from arcball import ArcBall
from volumereader import *
import Tkinter, tkFileDialog, tkMessageBox, tkSimpleDialog

# Number of slices to use for composition
nslices = 300

# Alpha increment from one slice to the other
basealpha = 1

# screen size
width = height = 600

# Volume aspect ratio
root = Tkinter.Tk()
root.withdraw()
```

```
largura = tkSimpleDialog.askfloat('Test', 'Digite a largura: ')
altura = tkSimpleDialog.askfloat('Test', 'Digite a altura: ')
profundidade = tkSimpleDialog.askfloat('Test', 'Digite a profundidade: ')
aspectRatio = largura, altura, profundidade


# current mouse position
mousex, mousey = 0,0

# current rotation matrix
rotmatrix = [1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1]

# Default color ramps for creating the transfer function
ramps = [(60,20,1,1,1,1.0)]

# Some colors for use in the transfer function editting interface
colors = []
bands = 0, 0.5, 1.0
for r in bands:
  for g in bands:
    for b in bands:
      if r+g+b>=1: colors += [(r,g,b)]

# The currently selected color which will be used for adding another ramp
# to the transfer function
curcolor = (1,1,1) #white

# Whether to display the Transfer Function editing interface
showTransferFunction = True

# Whether to display the Histogram
showHistogram = True

def display():
   """Display Callback function"""
   glClear (GL_COLOR_BUFFER_BIT)

   # Draw the volume
   glEnable (GL_TEXTURE_3D)
   glEnable (GL_BLEND)
   glBindTexture(GL_TEXTURE_3D, tex3D)

   glColor4f (1,1,1,basealpha)

   glBegin (GL_QUADS)
   for ir in range (nslices):
      for x,y,s,t in ((-1,-1, -0.4, -0.4),
```

```
                (1, -1, 1.4, -0.4),
                (1,  1, 1.4, 1.4),
                (-1, 1, -0.4, 1.4)):
        glTexCoord3f(s,t,-0.4+ir*1.8/nslices)
        glVertex3f(x,y,0)
glEnd()
glDisable (GL_TEXTURE_3D)

if showHistogram:
    #
    # Draw a frequency histogram
    #
    glDisable(GL_BLEND)
    global counts, maxcount
    dx = 2.0/len(counts)
    glColor4f(1,1,1,1)
    for i,count in enumerate(counts):
        x = -1 + i*dx
        h = min(1,count*1.0/maxcount)
        glRectf (x,-0.7, x+dx, -0.7+0.1*h)

if showTransferFunction:
    #
    # Draw the Transfer Function editing interface
    #

    # display a bar with the transfer function
    glEnable (GL_TEXTURE_1D)
    glBindTexture (GL_TEXTURE_1D, tex1D)
    glEnable(GL_BLEND)
    glMatrixMode (GL_TEXTURE)
    glPushMatrix()
    glLoadIdentity()
    glColor3f(1,1,1)
    glBegin (GL_QUADS)
    for x,y,s in ((-1,-0.9, 0),
                  (1, -0.9, 1),
                  (1, -0.7, 1),
                  (-1,-0.7, 0)):
        glTexCoord2f(s,0)
        glVertex2f(x,y)
    glEnd()
    glPopMatrix()

    # Display a small rectangle with the current color
    # underneath the bar position which will be editted
    # if the mouse button is pressed
    glDisable (GL_TEXTURE_1D)
```

```
        glDisable (GL_BLEND)
        glColor3f(*curcolor)
        x = mousex * 2.0 / width - 1
        y = -0.9
        glRectf (x-0.05,y-0.05, x+0.05, y)

        # Display a small colored bar at the bottom of the
        # screen so that a color can be chosen by clicking
        # on it
        ncolors = len(colors)
        dx = 2.0/ncolors
        for i,color in enumerate(colors):
            x = -1 + i*dx
            glColor3f(*color)
            glRectf (x,-0.95, x+dx, -1.0)

    glutSwapBuffers()

def reshape(w,h):
    """Callback called for window resize operations."""
    global width, height
    width, height = w,h
    glViewport (0,0,w,h)


def mouse(button,state,x,y):
    """Mouse Callback for button press/button release events."""
    global mousex, mousey, ramps, curcolor, arcball
    if state == GLUT_DOWN:
        if button == GLUT_LEFT_BUTTON:
            # Rotate scene
            arcball = ArcBall ((width/2,height/2,0), width/2)
            mousex, mousey = x, y
            glutMotionFunc (rotateVolume)
        elif button == GLUT_RIGHT_BUTTON:
            mousex, mousey = x, y
            if y>height*0.975:
                # clicked on the color bar: select a new color
                icolor = max(0, min (len(colors)-1, x*len(colors)/width))
                curcolor = colors [icolor]
                glutMotionFunc (None)
            else:
                # clicked elsewhere on the screen: add a new
                # ramp to the transfer function and mouse dragging will
                # adjust its alpha value and width
                ramps += [(mousex*256/width,0)+curcolor+(0,)]
                glutMotionFunc(adjustramp)
            glutPostRedisplay()
```

```
        else:
            # Do nothing on other button click
            glutMotionFunc (None)
    elif button == GLUT_RIGHT_BUTTON:
        # 3D texture is reloaded only after releasing the right mouse button
        global data
        load3DTexture(data)
        glutPostRedisplay()

def adjustramp (x,y):
    """Callback for mouse dragging while editing the
    transfer function.
    """
    global ramps, data
    center = mousex*256/width
    rampwidth = abs (x*256/width - center)
    alpha = max (0, min (1.0, (mousey-y)*2.0/height))
    ramps [-1] = (center,rampwidth)+curcolor+(alpha,)
    modifyTransferFunction()
    load1DTexture()
    glutPostRedisplay()

def mousehover (x,y):
    """Callback for mouse motion."""
    global mousex, mousey
    mousex, mousey = x, y
    glutPostRedisplay()

def setTransformation():
    """Sets up the proper texture transformation Matrix"""
    glMatrixMode(GL_TEXTURE)
    glLoadIdentity();
    glTranslate(0.5,0.5,0.5)
    glScalef (1.0/aspectRatio[0], 1.0/aspectRatio[1], 1.0/aspectRatio[2]);
    glTranslate(-0.5,-0.5,-0.5)
    glMultMatrixf(rotmatrix);

def rotateVolume(x,y):
    """Mouse callback for mouse drag with left button events. Rotates
    the volume using an Arcball interaction abstraction."""
    global mousex,mousey, arcball, rotmatrix
    angle, axis = arcball.rot (mousex, height-mousey, x, height-y)
    mousex,mousey = x,y
    glMatrixMode(GL_TEXTURE)
    glLoadIdentity()
    glMultMatrixf (rotmatrix)
    glTranslatef(0.5,0.5,0.5)
    glRotatef(-degrees(angle),*axis)
```

```
            glTranslatef(-0.5,-0.5,-0.5)
            rotmatrix = glGetDoublev (GL_TEXTURE_MATRIX)
            setTransformation()
            glutPostRedisplay()

        def keyboard(ch,x,y):
            """Callback for keyboard strokes."""
            key = ord(ch)
            if key==27: # Escape
                exit(0)
            elif key==32: # Space -> Toggles the display of transfer function
                global showTransferFunction
                showTransferFunction = not showTransferFunction
                glutPostRedisplay()
            elif ch in "hH": # h -> Toggles the display of the histogram
                global showHistogram
                showHistogram = not showHistogram
                glutPostRedisplay()
            elif key in (8,127): # Backspace, Del
                global ramps, data
                if len(ramps)>0:
                    ramps.pop()
                    modifyTransferFunction()
                    load1DTexture()
                    load3DTexture(data)
                    glutPostRedisplay()

        def initTextureEnvironment():
            """Initializes several opengl variables related to textures."""
            glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
            glPixelStorei(GL_PACK_ALIGNMENT, 1);

            glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR)
            glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR)
            glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_S,
GL_CLAMP_TO_BORDER)
            glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_T,
GL_CLAMP_TO_BORDER)
            glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_R,
GL_CLAMP_TO_BORDER)

            glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR)
            glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR)
```

```
            glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_WRAP_S,
GL_CLAMP_TO_BORDER)

            glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
            glEnable(GL_BLEND)
            glTexEnvf( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
GL_MODULATE )

        def create3DTexture ():
            """Creates a new 3D texture object, binds it and returns its name."""
            tex = glGenTextures(1)
            glBindTexture(GL_TEXTURE_3D, tex)
            return tex

        def load3DTexture (data):
            """ Creates a 3D texture from an input 'data' array, which
            should be a 3D array of bytes.
            """

            # Obtain geometry from the data array shape
            nz,ny,nx = data.shape

            # Create texture
            glTexImage3D (GL_TEXTURE_3D,        #
                    0,                  # level
                    GL_RGBA,            # internal format
                    nx,                 # width
                    ny,                 # height
                    nz,                 # depth
                    0,                  # border
                    GL_COLOR_INDEX,     # format
                    GL_UNSIGNED_BYTE,   # type
                    data)               # data
            setTransformation()

        def create1DTexture ():
            """Creates a one-dimensional texture object and returns it."""
            tex = glGenTextures(1)
            glBindTexture(GL_TEXTURE_1D, tex)
            return tex

        def load1DTexture():
            """Loads a one-dimensional texture for colormap indices
            with a ramp from 0 to 255.
            """
            # Array of bytes with values from 0 to 255
            ramp = array(range(256),dtype='u1');
            # Create texture
```

```
        glTexImage1D (GL_TEXTURE_1D,        #
                0,                  # level
                GL_RGBA,            # internal format
                256,                # width
                0,                  # border
                GL_COLOR_INDEX,     # format
                GL_UNSIGNED_BYTE,   # type
                ramp)               # data

    def modifyTransferFunction():
        """Regenerates the transfer function from the values in ramps."""
        global tf, ramps
        tf = TransferFunction()
        for ramp in ramps:
            tf.addRamp(*ramp)
        tf.load()

    def genHistogram(data):
        """Generates an histogram of the data array.
        Stores in global variable counts the frequency of each
        scalar value (from 0 to 255) and in variable maxcount
        the maximum frequency for all values other than 0."""
        bins = range(256)
        global counts,maxcount
        counts,bins = histogram(data,bins)
        maxcount = max(counts[1:254])

    #
    # Main Program
    #

    # OpenGL initialization
    glutInit(sys.argv)
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE)
    glutInitWindowSize(width, height)
    glutCreateWindow("Test")
    glutDisplayFunc(display)
    glutReshapeFunc(reshape)
    glutMouseFunc (mouse)
    glutPassiveMotionFunc(mousehover)
    glutKeyboardFunc(keyboard)

    # Read data and set texture
    dire = tkFileDialog.askdirectory(parent=root,initialdir="/",title='Selecione o
diretorio das imagens')
    numero = tkSimpleDialog.askinteger('Test', 'Digite o numero de imagens: ')
    nome = tkSimpleDialog.askstring('Test', 'Digite o nome da imagem ')
    diretorio = dire + '/' + nome + '%03d.jpg'
```

```
data = readVolumeFromColorImages (0,numero,diretorio,True)
genHistogram(data)
# Set Transfer function
modifyTransferFunction()

# Set textures
tex3D = create3DTexture ()
load3DTexture(data)
tex1D = create1DTexture ()
load1DTexture()
initTextureEnvironment()
print "Set textures"
glutMainLoop()
```