Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Reconstrução 3D de Imagens Tomográficas de Raios-X de Arcos Coronais em Ambiente Paralelo

Lilian Nogueira de Faria

São Carlos - SP
2003

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

F224rt

Faria, Lilian Nogueira de.
Reconstrução 3D de imagens tomográficas de raios-x de arcos coronais em ambiente paralelo / Lilian Nogueira de Faria. -- São Carlos : UFSCar, 2005.
114 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2003.

1. Reconstrução de imagens tomográficas. 2. Processamento paralelo (computadores). 3. Teoria bayesiana da estimação. 4. Processamento de imagens. 5. Computação gráfica. I. Título.

CDD: 006.693 (20ª)

Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Reconstrução 3D de Imagens Tomográficas de Raios-X de Arcos Coronais em Ambiente Paralelo

Lilian Nogueira de Faria

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Membros da Banca:

_____

Prof. Dr. Nelson Delfino d'Ávila Mascarenhas
(Orientador – DC/UFSCar)

_____

Prof. Dr. Célio Estevan Moron
(Co-orientador – DC/UFSCar)

_____

Prof. Dr. Reinaldo Roberto Rosa
(INPE/SJC)

_____

Prof. Dr. Junior Barrera
(IME/USP)

São Carlos - SP
2003

Aos meus pais que me deram total apoio para
que eu pudesse vencer mais esta etapa.

Agradecimentos

Ao Prof. Dr. Nelson Delfino d'Ávila Mascarenhas, pela orientação e oportunidade de desenvolver este trabalho de pesquisa na área de Processamento de Imagens e Sinais, e pelas explicações sobre o método de Interpolação 3D de Imagens usando Teoria de Estimação.

Ao Prof. Dr. Célio Estevan Moron, pesquisador na área de Sistemas Distribuídos e Redes, pela minha iniciação na pesquisa científica e co-orientação no desenvolvimento deste trabalho, contribuindo significativamente para minha formação acadêmica, da Iniciação Científica ao Mestrado. Em especial, pela proposta do projeto de Mestrado em conjunto com o Prof. Nelson Mascarenhas, agregando ao meu aprendizado na área de Processamento de Imagens, o conhecimento e experiência adquiridos no desenvolvimento de programas paralelos em Sistemas de Tempo Real. Agradeço aos orientadores por disponibilizarem os recursos computacionais para o desenvolvimento do projeto e pelo suporte em todas as fases de desenvolvimento do Mestrado.

Ao Prof. Dr. Reinaldo Roberto Rosa, do Instituto Nacional de Pesquisas Espaciais (INPE/SJC), por compartilhar um pouco do seu conhecimento e entusiasmo pela área de Ciência Espacial, fornecendo informações sobre a dinâmica do arco coronal, sem as quais este trabalho não poderia ser realizado.

Ao Prof. Dr. André Luiz Battaiola, pela participação no Exame de Qualificação e, principalmente, pelo estímulo na disciplina de Computação Gráfica, onde pude aprender e praticar os conhecimentos específicos, incluindo o conceito de Visualização Volumétrica, muito útil neste trabalho de Mestrado.

Ao Prof. Dr. José Hiroki Saito, pela participação e sugestões no Exame de Qualificação.

Ao Prof. Dr. Junior Barrera, do Departamento de Ciência da Computação do Instituto de Matemática e Estatística (IME/USP), pela participação e sugestões na Defesa da Dissertação.

A todos os Professores do Departamento de Computação da UFSCar, pela qualidade do ensino oferecido na Graduação e na Pós-Graduação em Ciência da Computação.

À equipe de suporte da Eonic Solutions GmbH (Bélgica), pelo eficiente atendimento de suporte e, principalmente, ao Richard Middendorf, por me salvar num momento decisivo para o desenvolvimento do projeto de Mestrado, me proporcionando auxílio técnico para identificar e corrigir o problema na instalação do Virtuoso no sistema Atlas, me permitindo implementar com sucesso o programa paralelo, já quase no final do Mestrado.

Aos colegas de Pós-Graduação, administradores do Laboratório de Processamento de Imagens, que dedicaram parte de seu tempo para dar assistência técnica aos usuários.

À CAPES, pela bolsa de estudo concedida nestes dois anos de Mestrado, para que eu pudesse me dedicar em tempo integral às atividades da Pós-Graduação.

Aos meus pais, pelo apoio fundamental na conquista das melhores oportunidades de estudo para realizar minha formação profissional, pelas palavras certas nos momentos difíceis, e por estarem sempre presentes em minha vida.

Vencer esta etapa foi, sem dúvida alguma, um grande desafio... Sem a contribuição de todos, seria impossível realizar este trabalho.

Sumário

Apêndices

Lista de Figuras

RESUMO

Pesquisas espaciais estão sendo realizadas com o objetivo de prever as explosões solares, através de imagens capturadas por satélites equipados com telescópios de raios-X. As imagens solares de alta resolução possibilitam o estudo da dinâmica espaço-temporal de estruturas plasma-magnéticas em forma de arco, denominadas arcos coronais. Os arcos coronais podem fornecer informações importantes sobre a dinâmica das explosões solares, que podem causar sérias perturbações nos sistemas de comunicação terrestre. No entanto, estas informações estão sujeitas a erros devido à limitação espacial imposta pelas imagens bidimensionais. Motivado pela necessidade de extrair informações da estrutura tridimensional dos arcos coronais, um método para reconstrução 3D de arcos coronais a partir de duas imagens tomográficas de raios-X observadas pelo satélite japonês *Yohkoh* foi desenvolvido. Este método de reconstrução 3D baseado em metamorfose de imagens gera as imagens intermediárias de seções transversais do arco coronal usando um método de deformação de imagens controlado por uma curva Bezier e um método de *Interpolação 3D de Imagens usando a Teoria de Estimação Bayesiana*. Devido à necessidade de reconstrução 3D de arcos coronais em um tempo razoável para estudo da dinâmica espaço-temporal, um programa paralelo de reconstrução 3D foi implementado para executar em uma máquina paralela de alto desempenho.

ABSTRACT

Space researches have been carried with the objective to forecast solar explosions, through images obtained by satellites equipped with X-ray telescopes. The high-resolution solar images allow the study of the spatio-temporal dynamics of plasma-magnetic structures in arc shape, denominated coronal loops. The coronal loops can provide important information about the solar explosions dynamics that may cause serious perturbations in terrestrial communication systems. However, this information is subject to errors due to the spatial limitation imposed by bidimensional images. Motivated by the need to extract information of the tridimensional structure of the coronal loops, a method for 3D reconstruction of coronal loops from two X-ray tomographic images captured by the Japanese satellite Yohkoh was developed. This method of 3D reconstruction based on image morphing generates the intermediate images of the transversal sections of the loop using a image warping method controlled by a Bezier curve and an approach to *Image 3D Interpolation using Bayesian Estimation Theory*. Due to the need for the 3D reconstruction of the coronal loops in a reasonable time for study of the spatio-temporal dynamics, a parallel program was implemented to execute in a high performance parallel machine.

1

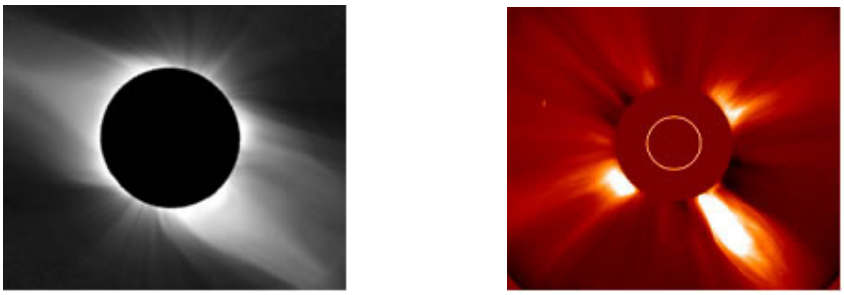
Introdução

Cientistas espaciais estão realizando pesquisas com o objetivo de detectar e prever as explosões solares, que podem causar sérias perturbações nos sistemas de comunicação terrestre, além de danificar satélites e sistemas de fornecimento de energia. As explosões ocorrem na coroa solar, devido à liberação repentina de grande quantidade de energia magnética, lançando partículas energéticas que podem afetar a magnetosfera terrestre.

A coroa solar é a camada mais externa da atmosfera solar, extremamente rarefeita e constituída de plasma (gás ionizado) a altas temperaturas. As emissões de alta energia da coroa podem ser detectadas em diferentes faixas do espectro eletromagnético, desde ondas de rádio, até raios-X e raios gama. As emissões de raios-X, medidos em unidades de milhares de elétron-volts (keV), variam na faixa de 1 keV a 100 keV. Os raios gama possuem a mais alta energia entre todas as formas de radiação eletromagnética, apresentando energia acima de 100 keV.

As emissões de ondas de rádio das explosões solares podem ser observadas com radiotelescópios na Terra. A radiação óptica da coroa é extremamente fraca e, por isso, só pode ser observada da Terra durante um eclipse solar total ou utilizando telescópios especiais, chamados coronógrafos, que simulam

um eclipse bloqueando a luz proveniente do disco solar. A Figura 1.1(a) mostra a coroa solar, no eclipse total de 26 de fevereiro de 1998, observada com a câmara de luz branca do observatório *High Altitude Observatory* (HAO) do *National Center for Atmospheric Research* (NCAR). A Figura 1.1(b) mostra a coroa solar observada pelo coronógrafo do satélite SOHO (*Solar and Heliospheric Observatory*), em 18 de março de 2003. As radiações de raios-X e raios gama não penetram na atmosfera terrestre e, portanto, só podem ser detectadas por satélites lançados no espaço.



(a) Coroa solar no eclipse total. (b) Imagem de um coronógrafo.

FIGURA 1.1 – Imagens da coroa solar.

O avanço tecnológico na área espacial possibilitou a aquisição de imagens solares através de satélites equipados com sistemas de imageamento. Vários satélites já foram lançados com o objetivo de estudar as emissões de alta energia das explosões solares. O satélite japonês *Yohkoh* esteve em operação durante dez anos, desde que foi lançado em 31 de agosto de 1991, no Japão.[†] Outros satélites, como o RHESSI (*Ramaty High Energy Solar Spectroscopic Imager*), TRACE (*Transition Region and Coronal Explorer*) e SOHO, permitem estudar as radiações de raios ultravioleta, raios-X e raios gama das explosões solares.

[†] Em dezembro de 2001, durante um eclipse anular, o satélite *Yohkoh* perdeu contato com o Sol e começou a girar fora de controle. Enquanto isso, as baterias se descarregaram. Este evento ocorreu durante um período raro da órbita do satélite (conhecida como órbita invisível), quando o satélite estava fora de comunicação com os controladores na Terra, que não puderam detectar e corrigir o posicionamento do satélite. Os controladores restabeleceram o contato com o satélite e desligaram todos os instrumentos para economizar energia. Como o satélite continua girando e recebe periodicamente a iluminação do Sol, os controladores esperam que as baterias sejam recarregadas, para que possam readquirir o controle do satélite e o contato com o Sol.

O satélite japonês *Yohkoh*[*] é equipado com dois telescópios: *Hard X-Ray Telescope* e *Soft X-Ray Telescope*, que detectam raios-X duros (HXR) e raios-X moles (SXR), respectivamente. Os raios-X duros são os raios-X de mais alta energia do espectro eletromagnético, enquanto os raios-X de mais baixa energia são denominados raios-X moles. A distinção entre raios-X duros e moles não é bem definida, mas, tipicamente, raios-X duros são aqueles com energia maior que 10 keV. A Figura 1.2 mostra uma das últimas imagens da coroa solar observada pelo telescópio de raios-X moles do satélite *Yohkoh* em 14 de dezembro de 2001.[†]

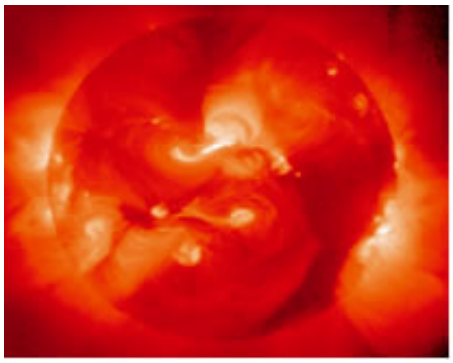


FIGURA 1.2 – Imagem de raios-X moles da coroa solar observada pelo satélite Yohkoh.

A superfície solar, denominada *fotosfera*, apresenta uma temperatura muito baixa, em torno de 6000 °C, quando comparada com a temperatura do núcleo solar, de aproximadamente 15 milhões de graus Celsius, e da coroa solar, que pode atingir dois milhões de graus Celsius. Devido à baixa temperatura, a fotosfera não emite quantidade suficiente de raios-X e, por isso, aparece escura nas imagens, em contraste com a coroa solar.

As imagens solares de alta resolução espacial e temporal mostram que a coroa solar é altamente heterogênea e dinâmica [29, 30]. As estruturas plasma-magnéticas observadas na coroa solar emitem diferentes quantidades de energia, sendo que estruturas próximas das regiões ativas são fontes de emissões mais

[*] Termo japonês que significa "*Raios de Sol*".

[†] Outras informações, imagens e filmes solares podem ser encontrados nos sites dos satélites Yohkoh, RHESSI, TRACE e SOHO.

potentes de raios-X. Dentre as estruturas observadas nas regiões ativas, os arcos coronais ou *loops* surgem a partir de linhas de campo magnético entre regiões positivas e negativas da superfície solar, gerando um fluxo de plasma magnético em forma de arco. A Figura 1.3 mostra um arco coronal observado pelo TRACE, em março de 1999.

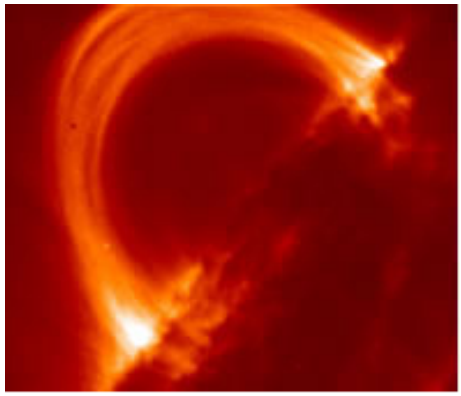


FIGURA 1.3 – Arco coronal observado no limbo solar pelo satélite TRACE.

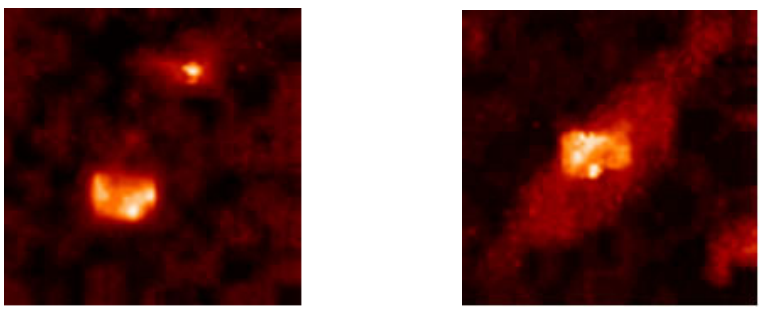
As imagens de raios-X dos arcos coronais mostram freqüentemente estruturas torcidas. Um movimento contínuo de rotação fluida em ambas as extremidades do arco gera ondas de torção que se propagam ao longo do arco. Às vezes, os arcos magnéticos distorcem após uma mudança topológica das linhas do campo magnético. Caso contrário, após diversas rotações, o arco magnético torna-se instável e ocorre uma liberação explosiva de energia com ejeção de massa na atmosfera solar.

Sendo assim, entender a estrutura tridimensional e a dinâmica dos arcos coronais é fundamental para a previsão de explosões solares. Esta é uma área de pesquisa em desenvolvimento que vem estudando a atividade solar, para desvendar a dinâmica das explosões solares.

1.1 Motivação e Objetivos

Os cientistas da Divisão de Astrofísica do Instituto Nacional de Pesquisas Espaciais (DAS/INPE) vêm realizando um estudo da dinâmica espaço-temporal das estruturas solares das regiões ativas, através de observações de satélites equipados com telescópios de raios-X [21, 24]. Entretanto, estes estudos são feitos com imagens bidimensionais sem qualquer informação sobre a estrutura tridimensional da coroa solar. Desta forma, parâmetros físicos das estruturas magnéticas (temperatura, densidade e campo magnético) e geométricos (altura, rotação e posição) estão sujeitos a erros devido à limitação espacial imposta pelas imagens bidimensionais. Motivados por esta necessidade, esforços têm sido feitos para permitir a reconstrução tridimensional de estruturas da coroa solar a partir de imagens 2D obtidas por satélite.

Uma técnica para reconstrução de estruturas da coroa solar é a tomografia espectral que possibilita a observação simultânea, do mesmo evento, em diferentes freqüências do espectro eletromagnético. Imagens tomográficas de raios-X obtidas pelos telescópios do satélite *Yohkoh* mostram a emissão de energia de um *loop* magnético em duas profundidades da atmosfera solar, sendo que a base do *loop* emite raios-X duros, enquanto o topo emite raios-X moles. A Figura 1.4 mostra as imagens tomográficas da base e topo de um *loop*, obtidas pelos telescópios de raios-X duros e raios-X moles do satélite *Yohkoh*, respectivamente.



(a) Imagem da base do *loop* (HXR). (b) Imagem do topo do *loop* (SXR).

FIGURA 1.4 – Imagens tomográficas de raios-X obtidas pelos telescópios do satélite Yohkoh.

Mais do que observar imagens 2D, os cientistas têm um grande interesse em visualizar a estrutura volumétrica dos *loops*. No entanto, devido a limitações físicas dos telescópios de raios-X do satélite *Yohkoh*, a quantidade de imagens tomográficas disponíveis não é suficiente para visualizar a estrutura tridimensional. Diante desta limitação, surge a necessidade de técnicas de reconstrução tomográfica 3D para obter imagens intermediárias entre as imagens disponíveis [22, 24].

Neste documento, será apresentado um método de *Reconstrução 3D de Imagens Tomográficas de Raios-X de Arcos Coronais em Ambiente Paralelo*. O objetivo deste trabalho é reconstruir a estrutura magnética tridimensional de um *loop*, a partir das imagens tomográficas de raios-X do satélite *Yohkoh*. Como as seções transversais do *loop* apresentam uma significativa mudança de forma e, principalmente, devido à inexistência de outras informações sobre a estrutura 3D do *loop*, a não ser as imagens da base e do topo, os métodos existentes de reconstrução 3D não são capazes de gerar uma reconstrução realística do arco. Para solucionar este problema, o método apresentado neste documento gera imagens intermediárias com uma mudança de forma entre as imagens da base e topo. Esta deformação é controlada por uma curva Bezier em forma de arco, que define a estrutura do campo magnético do *loop*.

Os arcos coronais não são estáticos e, por isso, há necessidade de reconstruções tridimensionais de uma seqüência de imagens tomográficas evoluindo no tempo, para realizar um estudo da dinâmica destes arcos. Desta forma, devido ao grande volume de dados a serem processados, a reconstrução tomográfica e a visualização dos dados precisam estar integradas em um ambiente paralelo de alto desempenho. Com isso, será possível, no futuro, um estudo em três dimensões da dinâmica espaço-temporal dos arcos coronais.

1.2 Visão Geral do Projeto

O projeto de mestrado *Reconstrução 3D de Imagens Tomográficas de Raios-X de Arcos Coronais em Ambiente Paralelo* tem como objetivo reconstruir um arco coronal da atmosfera solar, a partir de duas imagens tomográficas de raios-X do satélite *Yohkoh*. Neste projeto, foi desenvolvido um método de reconstrução 3D baseado em metamorfose de imagem, controlado por uma curva Bezier, para reconstruir a arcada magnética do *loop*. No processo de metamorfose, um método de *Interpolação 3D de Imagens usando a Teoria de Estimação Bayesiana* é usado para estimar a intensidade dos *pixels* de uma imagem intermediária entre um par de imagens.

Devido à necessidade de obter a reconstrução 3D de arcos coronais em um tempo razoável para estudo da dinâmica espaço-temporal, um programa paralelo de reconstrução foi implementado com o auxílio de um *Ambiente Visual de Desenvolvimento de Programas Paralelos de Tempo Real* [19], para ser executado em uma máquina paralela usando DSPs [1]. O ambiente visual oferece suporte ao desenvolvimento de aplicações utilizando o *kernel* de tempo real Virtuoso [32], que define apenas os objetos e serviços de *microkernel* necessários para o desenvolvimento de aplicações paralelas. Cada objeto de *microkernel* – tarefa, semáforo, recurso, fila, canal, mapa de memória, *timer* e *mailbox* – suporta um conjunto de serviços para sincronização e comunicação entre tarefas, alocação de recursos, memória, etc.

Após a reconstrução do *loop*, técnicas de visualização 3D podem ser usadas para visualizar a estrutura tridimensional do arco magnético. Assim, uma interface gráfica de visualização 3D fornecida pelo ambiente de desenvolvimento IDL [6] foi adicionada ao projeto. O ambiente IDL oferece uma linguagem de programação e os recursos necessários para desenvolvimento de programas de visualização de dados.

O programa paralelo de reconstrução 3D e a interface gráfica de visualização são integrados no ambiente paralelo através de uma interface gráfica de usuário desenvolvida na linguagem IDL.

1.3 Organização da Dissertação

Os próximos capítulos apresentam os principais conceitos usados para desenvolvimento do projeto de mestrado, como interpolação 3D de imagens, metamorfose de imagens, reconstrução 3D e visualização de volume, além dos fundamentos básicos sobre sistemas paralelos e uma metodologia para desenvolvimento de programas paralelos.

No Capítulo 2 é apresentado o método de *Interpolação 3D de Imagens usando a Teoria de Estimação Bayesiana*, que gera uma imagem intermediária entre um par de imagens originais.

O Capítulo 3 apresenta uma técnica de metamorfose de imagens, geralmente empregada para gerar efeitos especiais, mas que pode ser muito útil em aplicações de reconstrução de volume.

O Capítulo 4 introduz a reconstrução de volume, apresentando alguns exemplos de reconstrução tomográfica usando métodos de interpolação e metamorfose de imagens.

No Capítulo 5 são abordadas as principais técnicas de visualização de volume.

O Capítulo 6 apresenta os fundamentos de sistemas paralelos e uma metodologia para desenvolvimento e análise de desempenho de programas paralelos.

O Capítulo 7 descreve o método de reconstrução 3D de arcos coronais e o Capítulo 8 apresenta o programa de reconstrução 3D de imagens tomográficas. Os capítulos seguintes apresentam os resultados obtidos, conclusões e trabalhos futuros.

2

Interpolação 3D de Imagens usando a Teoria de Estimação Bayesiana

O método de *Interpolação 3D de Imagens usando a Teoria de Estimação Bayesiana* foi desenvolvido por Mascarenhas *et al.* [14] para gerar uma imagem intermediária entre um par de imagens originais. Este método, baseado na Teoria de Estimação, usa um estimador linear de mínimo erro médio-quadrático para estimar o valor dos *pixels* da imagem intermediária. A Teoria de Estimação e o método de interpolação 3D de imagens são apresentados nas próximas seções. No final deste capítulo, são incluídos os resultados obtidos por Mascarenhas *et al.* na interpolação de uma imagem a partir de duas imagens tomográficas de raios-X do arco coronal e imagens de ressonância magnética de frutas.

O método de interpolação bayesiana é usado no projeto de reconstrução 3D descrito neste documento, para estimar a intensidade dos *pixels* das imagens intermediárias do arco coronal.

2.1 Teoria de Estimação Bayesiana

A Teoria de Estimação é amplamente usada em várias áreas da ciência e engenharia. Karl Frederich Gauss é conhecido como o progenitor da Teoria de Estimação. Fisher, Norbert Wiener, Rudolph E. Kalman e vários outros expandiram o legado de Gauss com vários métodos de estimação e algoritmos. A Teoria de Estimação é resultado da necessidade e tecnologia. Gauss, por exemplo, precisava prever os movimentos dos planetas e cometas por medidas telescópicas. Esta necessidade levou ao método dos mínimos quadrados.

2.1.1 Estimação de parâmetros

O problema de estimação consiste em determinar um parâmetro desconhecido que deve ser estimado em função de amostras disponíveis [28]. Esta técnica é conhecida como *estimação de parâmetros*. Exemplos de parâmetros escalares θ são: média μ e variância σ^2; exemplos de parâmetros vetoriais* $\boldsymbol{\theta} = [\theta_1 \ldots \theta_n]^T$ são: vetor média $\boldsymbol{\mu}$, matriz de covariância **K**, etc.

Definição 2.1. Um *estimador* $\hat{\Theta}$ é uma função do vetor de observações $\boldsymbol{X} = [X_1 \ldots X_n]^T$ que estima o parâmetro θ. O valor do estimador $\hat{\Theta}$ é denominado *estimativa* de θ.

Definição 2.2. Um estimador $\hat{\Theta}$ é chamado de *estimador linear de* θ se for uma função linear do vetor de observações $\mathbf{X} = [X_1 \ldots X_n]^T$, ou seja,

$$\hat{\Theta} = \mathbf{b}^T \mathbf{X}, \tag{2.1}$$

onde **b** é um vetor nx1 de coeficientes independentes de **X**.

Definição 2.3. Um estimador $\hat{\Theta}$ é chamado de *estimador sem-viés de* θ se

$$E(\hat{\Theta}) = \theta, \tag{2.2}$$

ou seja, o valor esperado da estimativa de θ é igual ao parâmetro θ que está sendo estimado.

* Ver Apêndice B: Vetores Aleatórios.

Definição 2.4. Um estimador $\hat{\Theta}$ é chamado de *estimador de mínimo erro médio-quadrático* (MEMQ) se

$$E[(\hat{\Theta} - \theta)^2] \leq E[(\hat{\Theta}' - \theta)^2], \tag{2.3}$$

onde $\hat{\Theta}'$ é qualquer outro estimador.

2.1.1.1 Estimação de média

Seja uma variável aleatória X com n observações $X_1,...,X_n$ igualmente distribuídas. Uma estimativa da média $\mu = E[X]$ é obtida por um *estimador de média amostral* definido como

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} X_i. \tag{2.4}$$

Calculando o valor esperado de $\hat{\mu}$, vemos que a equação (2.2) é satisfeita:

$$E[\hat{\mu}] = E\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] = \frac{1}{n}\sum_{i=1}^{n} E(X_i) = \frac{1}{n}(n).\mu = \mu,$$

ou seja, o estimador $\hat{\mu}$ é um estimador sem-viés para o parâmetro μ.

2.1.1.2 Estimação de variância

Dada uma variável aleatória X com n observações $X_1,...,X_n$ independentes e igualmente distribuídas, uma estimativa da variância $\sigma^2 = Var[X]$ pode ser obtida por um estimador definido como

$$\hat{\Theta} = \frac{1}{n}\sum_{i=1}^{n} (X_i - \hat{\mu})^2, \tag{2.5}$$

onde $\hat{\mu}$ é a média amostral definida na equação (2.4). Como o valor esperado de $\hat{\Theta}$ é

$$E(\hat{\Theta}) = \frac{1}{n}\left[\sum_{i=1}^{n} E(X_i - \mu)^2 - n.E\left(\hat{\mu} - \mu\right)^2\right] = \frac{1}{n}\left[\sum_{i=1}^{n} Var\left(X_i\right) - n.Var\left(\hat{\mu}\right)\right] =$$

$$= \frac{1}{n}\left[n.\sigma^2 - n\frac{\sigma^2}{n}\right] = \frac{n-1}{n}\sigma^2,$$

o estimador $\hat{\Theta}$ é viesado.

Para obter um estimador sem-viés, multiplica-se a equação $E(\hat{\Theta}) = \frac{n-1}{n}\sigma^2$ por $\frac{n}{n-1}$:

$$\frac{n}{n-1}E(\hat{\Theta}) = E\left[\frac{n}{n-1}\hat{\Theta}\right] = E\left[\frac{n}{n-1}.\frac{1}{n}\sum_{i=1}^{n}\left(X_i - \hat{\mu}\right)^2\right] = E\left[\frac{1}{n-1}\sum_{i=1}^{n}\left(X_i - \hat{\mu}\right)^2\right] = \sigma^2$$

Assim, um estimador sem-viés para a variância σ^2 é o *estimador de variância amostral*, definido por

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}\left(X_i - \hat{\mu}\right)^2. \tag{2.6}$$

2.1.1.3 Estimação de vetor média

Sejam N amostras aleatórias $\mathbf{X}_1,...,\mathbf{X}_N$ independentes e igualmente distribuídas, onde $\mathbf{X}_k$ é o k-ésimo vetor aleatório com n observações $X_1,...,X_n$, ou seja,

$$\mathbf{X}_1 = [X_{11}\ X_{i1}\ \ldots\ X_{n1}]^T$$

$$\mathbf{X}_k = [X_{1k}\ X_{ik}\ \ldots\ X_{nk}]^T$$

$$\vdots$$

$$\mathbf{X}_N = [X_{1N}\ X_{iN}\ \ldots\ X_{nN}]^T.$$

Um *estimador do vetor média* $\boldsymbol{\mu} = [\mu_1\ \mu_2 \ldots \mu_N]^T$ é definido por

$$\hat{\boldsymbol{\mu}} = \frac{1}{N}\sum_{k=1}^{N}\mathbf{X}_k, \tag{2.7}$$

onde $\hat{\boldsymbol{\mu}} = [\hat{\mu}_1\ \hat{\mu}_2 \ldots \hat{\mu}_N]^T$ é chamado de *vetor média amostral*.

2.1.1.4 Estimação de matriz de covariância

Sejam N amostras aleatórias $\mathbf{X}_1,...,\mathbf{X}_N$ independentes e igualmente distribuídas, onde $\mathbf{X}_k$ é o k-ésimo vetor aleatório com n elementos $X_1,...,X_n$. Se o vetor média $\boldsymbol{\mu}$ é conhecido, então o *estimador da matriz de covariância* $\mathbf{K}$ é definido por

$$\hat{\mathbf{K}} = \frac{1}{N}\sum_{k=1}^{N}\left(\mathbf{X}_k - \boldsymbol{\mu}\right)\left(\mathbf{X}_k - \boldsymbol{\mu}\right)^T. \tag{2.8}$$

Se o vetor média é estimado pela média amostral $\hat{\boldsymbol{\mu}}$ (equação 2.7), o estimador é

$$\hat{\mathbf{K}} = \frac{1}{N-1}\sum_{k=1}^{N}(\mathbf{X}_k - \hat{\boldsymbol{\mu}})(\mathbf{X}_k - \hat{\boldsymbol{\mu}})^T. \tag{2.9}$$

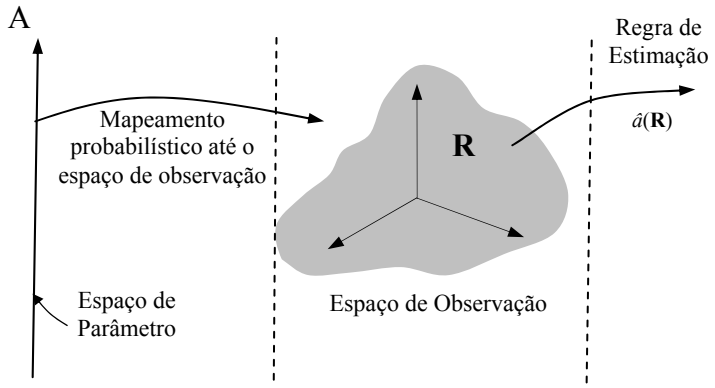
O elemento $\hat{c}_{ij}$ da *matriz de covariância amostral* $\hat{\mathbf{K}}$ é dado por

$$\hat{c}_{ij} = \frac{1}{N-1}\sum_{k=1}^{N}(X_{ik} - \hat{\mu}_i)(X_{jk} - \hat{\mu}_j) \tag{2.10}$$

onde X_{ik} é o *i*-ésimo componente da *k*-ésima amostra $\mathbf{X}_k$.

2.1.2 Estimação de Bayes

No problema de estimação, o mapeamento do espaço de observação em uma estimação é chamado de *regra de estimação* [31]. Depois de observar **R**, a estimação do parâmetro *a*, denominada $\hat{a}(\mathbf{R})$, é calculada pela regra de estimação. Um modelo geral do problema de estimação é mostrado na Figura 2.1.



Fonte: VAN TREES, 1968, p. 130.

FIGURA 2.1 – Modelo de Estimação.

É preciso determinar uma função de custo para todos os pares $(a, \hat{a}(\mathbf{R}))$, que é uma função de duas variáveis contínuas denotada por $C(a, \hat{a})$. A função de custo deve ser relacionada ao *erro de estimação.* Podemos definir este erro como

$$a_\varepsilon(\mathbf{R}) = \hat{a}(\mathbf{R}) - a \tag{2.11}$$

onde $a_\varepsilon(\mathbf{R})$ é o erro de estimação.

A função de custo é relacionada ao erro de estimação, então podemos expressá-la em termos do erro, $C(a, \hat{a}) = C(a_\varepsilon)$. Na Figura 2.2(a), a função de custo é o quadrado do erro,

$$C(a_\varepsilon) = a_\varepsilon^2 \tag{2.12}$$

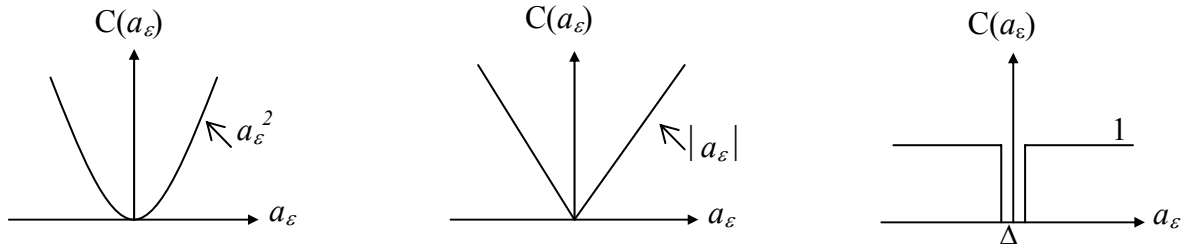
Na Figura 2.2(b), a função de custo é o valor absoluto do erro,

$$C(a_\varepsilon) = |a_\varepsilon| \tag{2.13}$$

Na Figura 2.2(c), assumimos custo zero para todos os erros menores que $\pm\frac{\Delta}{2}$.

Se $A_\varepsilon > \frac{\Delta}{2}$, assumimos um valor uniforme:

$$C(a_\varepsilon) = \begin{cases} 0, & |a_\varepsilon| \le \frac{\Delta}{2}, \\ 1, & |a_\varepsilon| > \frac{\Delta}{2}. \end{cases} \tag{2.14}$$



(a) função de custo de erro médio-quadrático.

(b) função de custo de erro absoluto.

(c) função de custo de erro uniforme.

Fonte: VAN TREES, 1968, p.55

FIGURA 2.2 – Tipos de funções de custo.

Devemos encontrar uma estimativa que minimize o valor esperado do custo (*risco*). Uma vez especificadas a função de custo e a densidade de probabilidade a priori $p_a(A)$, a expressão para o *risco* pode ser escrita:

$$\Re = E\{C[a, \hat{a}(\mathbf{R})]\} = \int_{-\infty}^{+\infty} dA \int_{-\infty}^{+\infty} C[A, \hat{a}(\mathbf{R})] p_{a,\mathbf{r}}(A, \mathbf{R}) d\mathbf{R} \tag{2.15}$$

Para custos que são funções de apenas uma variável,

$$\Re = \int_{-\infty}^{+\infty} dA \int_{-\infty}^{+\infty} C[A - \hat{a}(\mathbf{R})] p_{a,\mathbf{r}}(A, \mathbf{R}) d\mathbf{R} \tag{2.16}$$

O *estimador de Bayes* é o estimador que minimiza o risco para a função de custo dada. Para a função de custo da Figura 2.2(a), o *risco* corresponde ao erro médio quadrado. Denominamos o risco para o critério de erro médio-quadrático como R_{ms}. Substituindo a equação (2.12) em (2.16), temos

$$\Re_{ms} = \int_{-\infty}^{+\infty} dA \int_{-\infty}^{+\infty} d\mathbf{R}[A - \hat{a}(\mathbf{R})]^2 \, p_{a,\mathbf{r}}(A, \mathbf{R}) \tag{2.17}$$

Expressando a densidade conjunta pelo produto $p_{a,\mathbf{r}}(A, \mathbf{R}) = p_{\mathbf{r}}(\mathbf{R}) p_{a|\mathbf{r}}(A \mid \mathbf{R})$, temos

$$\Re_{ms} = \int_{-\infty}^{+\infty} d\mathbf{R} p_{\mathbf{r}}(\mathbf{R}) \int_{-\infty}^{+\infty} dA[A - \hat{a}(\mathbf{R})]^2 \, p_{\mathrm{a}|\mathbf{r}}(A|\mathbf{R}) \tag{2.18}$$

Agora a integral interna e $p_r(\mathbf{R})$ são não-negativas. Portanto, podemos minimizar $\Re_{ms}$ minimizando a integral interna. Denominamos esta estimativa $\hat{a}_{ms}(\mathbf{R})$. Para encontrá-la, diferenciamos a integral interna em relação a $\hat{a}(\mathbf{R})$:

$$\frac{d}{d\hat{a}} \int_{-\infty}^{+\infty} dA[A - \hat{a}(\mathbf{R})]^2 \, p_{\mathrm{a}|\mathbf{r}}\left(A|\mathbf{R}\right) = -2 \int_{-\infty}^{+\infty} A p_{\mathrm{a}|\mathbf{r}}\left(A|\mathbf{R}\right) dA + 2\hat{a}(\mathbf{R}) \int_{-\infty}^{+\infty} p_{\mathrm{a}|\mathbf{r}}\left(A|\mathbf{R}\right) dA \tag{2.19}$$

Igualando o resultado a zero e observando que a segunda integral é igual a 1, temos

$$\hat{a}_{ms}(\mathbf{R}) = \int_{-\infty}^{+\infty} A p_{\mathrm{a}|\mathbf{r}}\left(A|\mathbf{R}\right) dA \tag{2.20}$$

Para verificarmos se $\hat{a}_{ms}(\mathbf{R})$ é mínimo, precisamos obter a segunda derivada:

$$\frac{d^2}{d\hat{a}^2} \left\{ \int_{-\infty}^{+\infty} [A - \hat{a}(\mathbf{R})]^2 \, p_{\mathrm{a}|r}\left(A|\mathbf{R}\right) dA \right\} = 2 \tag{2.21}$$

A equação (2.20) é conhecida como a *média da densidade a posteriori* (ou esperança condicional ou média condicional) de a, dado $\mathbf{R} = \boldsymbol{r}$. Agora, se $\hat{a}(\mathbf{R})$ é a média condicional, a integral interna em (2.18) é a variância condicional de $\mathbf{R} = \boldsymbol{r}$. Com isso, o risco $\Re_{ms}$ é o valor esperado da variância condicional para todas as observações de $\mathbf{R}$.

Para encontrar a estimação de Bayes para o critério de valor absoluto na Figura 2.2(b), escrevemos:

$$\Re_{abs} = \int_{-\infty}^{+\infty} d\mathbf{R} p_r(\mathbf{R}) \int_{-\infty}^{+\infty} dA\left[\left|A - \hat{a}(\mathbf{R})\right|\right] p_{\mathrm{a}|r}\left(A|\mathbf{R}\right) \tag{2.22}$$

Como no caso do erro médio quadrático, a minimização da integral interna resulta na minimização de $\Re_{abs}$. Podemos reescrever a integral interna como

$$\int_{-\infty}^{+\infty} dA\left[\left|A-\hat{a}(R)\right|\right]p_{a|r}\left(A|R\right)=\int_{-\infty}^{\hat{a}(R)} dA\left[\hat{a}(R)-A\right]p_{a|r}\left(A|R\right)+\int_{\hat{a}(R)}^{+\infty} dA\left[A-\hat{a}(R)\right]p_{a|r}\left(A\mid R\right) \tag{2.23}$$

Diferenciando com relação a $\hat{a}(\mathbf{R})$:

$$\frac{d}{d\hat{a}}\int_{-\infty}^{\hat{a}(R)} dA[\hat{a}(R)-A]p_{a|r}\left(A|R\right)+\int_{\hat{a}(R)}^{+\infty} dA[A-\hat{a}(R)]p_{a|r}\left(A|R\right)=$$

$$=\hat{a}(R)\int_{-\infty}^{\hat{a}(R)} p_{a|r}\left(A|R\right)dA-\hat{a}(R)\int_{-\infty}^{\hat{a}(R)} p_{a|r}\left(A|R\right)dA \tag{2.24}$$

Igualando o resultado a zero, temos:

$$\int_{-\infty}^{\hat{a}_{abs}(R)} p_{a|r}\left(A|R\right)dA=\int_{\hat{a}_{abs}(R)}^{+\infty} p_{a|r}\left(A|R\right)dA \tag{2.25}$$

isto é, a definição da *mediana da densidade a posteriori* de *a*, dado **R** = ***r***. Logo, $\hat{a}_{abs}(\mathbf{R})$ é a *mediana* de *a*, dado **R** = ***r***.

O terceiro critério é a função de custo uniforme. A expressão do *risco* é

$$\Re_{unif}=\int_{-\infty}^{+\infty} dRp_r(R)\left[1-\int_{\hat{a}_{unif}(R)-\frac{\Delta}{2}}^{\hat{a}_{unif}(R)+\frac{\Delta}{2}} p_{a|r}\left(A|R\right)dA\right] \tag{2.26}$$

Neste caso, para minimizar $\Re_{unif}$ devemos maximizar a integral interna. Por isso, supomos que Δ seja arbitrariamente pequeno, mas diferente de zero.

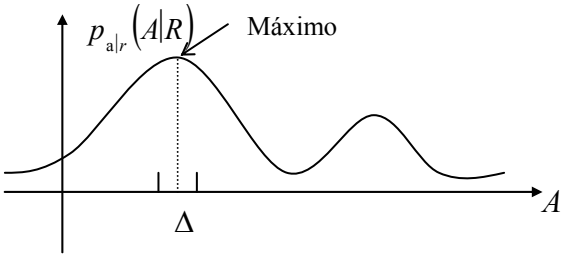


FIGURA 2.3 – Densidade a posteriori.

A Figura 2.3 mostra uma densidade *a posteriori* típica. Vemos que para delta pequeno a melhor escolha para $\hat{a}(\mathbf{R})$ é o valor de *A* onde a densidade *a posteriori* tem seu máximo. Denominamos a estimação para este caso especial como $\hat{a}_{map}(\mathbf{R})$, a *estimação máxima a posteriori*. Na seqüência usamos $\hat{a}_{map}(\mathbf{R})$ sem mais referências à função de custo uniforme.

Para encontrar $\hat{a}_{map}(\mathbf{R})$, devemos ter a localização do máximo de $p_{a|r}(A|R)$. Podemos encontrar a localização do máximo de $\ln p_{a|r}(A|R)$ facilmente, o que é mais conveniente. Se o máximo está dentro do limite permitido de A e $\ln p_{a|r}(A|R)$ tem uma derivada primeira contínua, então uma condição necessária, mas não suficiente para um máximo pode ser obtida diferenciando $\ln p_{a|r}(A|R)$ em relação a A e igualando o resultado a zero:

$$\left.\frac{\partial \ln p_{a|r}(A|R)}{\partial A}\right|_{A=\hat{a}(R)} = 0 \tag{2.27}$$

Referimo-nos à equação acima como a equação MAP (*máxima a posteriori*). Em cada caso devemos verificar para ver se a solução é o máximo absoluto. Devemos reescrever a expressão para $p_{a/r}(A \mid \mathbf{R})$ para separar a parte do vetor observado **R** e o conhecimento a priori:

$$p_{a|r}(A|R) = \frac{p_{r|a}(R|A)p_a(A)}{p_r(R)} \tag{2.28}$$

Usando logaritmos,

$$\ln p_{a|r}(A|R) = \ln p_{r|a}(R|A) + \ln p_a(A) - \ln p_r(R) \tag{2.29}$$

Para estimação MAP, estamos interessados apenas em encontrar o valor de A, onde o lado esquerdo é máximo. Como o último termo direito não é uma função de A, podemos considerar apenas a função

$$l(A) = \ln p_{r|a}(R|A) + \ln p_a(A) \tag{2.30}$$

O primeiro termo mostra a dependência probabilística de **R** em A e o segundo descreve o conhecimento a priori. A equação MAP pode ser escrita como

$$\left.\frac{\partial l(A)}{\partial A}\right|_{A=\hat{a}(R)} = \left.\frac{\partial \ln p_{r|a}(R|A)}{\partial A}\right|_{A=\hat{a}(R)} + \left.\frac{\partial \ln p_a(A)}{\partial A}\right|_{A=\hat{a}(R)} = 0 \tag{2.31}$$

2.1.3 Filtro de Wiener

Filtro de Wiener é um método de regularização estocástica que fornece a estimação linear de mínimo erro médio-quadrático do(s) parâmetro(s) aleatório(s) a partir dos dados. O propósito do filtro é produzir uma estimativa de um valor desejado, tal que o erro de estimação seja mínimo. O filtro de Wiener pode resolver o problema de minimizar o erro médio-quadrático, sob a restrição de operações lineares nos dados.

O problema da estimação linear de mínimo erro médio-quadrático é simplificado se é procurada uma estimativa X_0 por uma combinação linear de $X_1,..., X_n$. Neste caso, o problema é encontrar n constantes $a_1,..., a_n$ tal que o erro médio-quadrático

$$e = \mathrm{E}\{[X_0 - (a_1X_1 + ... + a_nX_n)]^2\} \tag{2.32}$$

seja mínimo. Estas constantes podem ser determinadas em termos dos segundos momentos

$$\mathrm{R_{ij}} = \mathrm{E}\{X_iX_j\} \tag{2.33}$$

das variáveis aleatórias dadas. Se $\mathrm{E}\{X_i\} = 0$, ou melhor, se as variáveis aleatórias tiverem média zero, então $\mathrm{R_{ij}}$ é a covariância de X_i e X_j. Esta suposição é feita sem perda de generalidade porque esta análise pode ser aplicada à variável aleatória

$$X_i - \mathrm{E}\{X_i\} \tag{2.34}$$

cuja média é igual a zero.

Pelo Princípio da Ortogonalidade, as constantes a_i que minimizam e são tais que o erro

$$X_0 - (a_1X_1 + ... + a_nX_n) \tag{2.35}$$

é ortogonal a $X_1,..., X_n$; isto é,

$$\mathrm{E}\{[X_0 - (a_1X_1 + ... + a_nX_n)]\ X_i\} = 0, \qquad \text{para } i = 1, ..., n \tag{2.36}$$

O erro mínimo médio-quadrático é dado por

$$e_\mathrm{m} = \mathrm{E}\{[X_0 - (a_1X_1 + ... + a_nX_n)]\ X_0\} = R_{00} - (a_1R_{01} + ... + a_nR_{0n}) \tag{2.38}$$

onde a_i são as soluções do problema.

Este sistema pode ser escrito na forma

$$\begin{cases} R_{11}a_1 + R_{12}a_2 + \ldots + R_{1n}a_n = R_{01} \\ R_{21}a_1 + R_{22}a_2 + \ldots + R_{2n}a_n = R_{02} \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ R_{n1}a_1 + R_{n2}a_2 + \ldots + R_{nn}a_n = R_{0n} \end{cases}$$

O sistema acima é o sistema de equações normais utilizado no método de interpolação 3D de imagens para obter os coeficientes necessários para a estimação da imagem intermediária.

2.2 Interpolação 3D de Imagens usando a Teoria de Estimação Bayesiana

Um novo método de *Interpolação 3D de Imagens usando a Teoria de Estimação Bayesiana*[*] foi desenvolvido por Mascarenhas *et al.* [14] para interpolar uma imagem intermediária entre um par de imagens originais. Este método de interpolação estima o valor dos *pixels* da imagem intermediária, usando um estimador linear de mínimo erro médio-quadrático. Para fazer a interpolação 3D usando a Teoria de Estimação Bayesiana sem uma imagem intermediária *a priori*, é necessário obter algumas informações como:

- informação estatística das imagens observadas;
- informação estatística sobre a dependência do *pixel* a ser estimado com os *pixels* das duas imagens observadas; e
- informação estatística *a priori* sobre a imagem a ser interpolada.

A informação *a priori* da imagem a ser estimada deve ser postulada. Todas as informações necessárias para interpolação são definidas na próxima seção.

[*] Dissertação do Programa de Pós-Graduação em Ciência da Computação da UFSCar (1999).

A interpolação da imagem é local, ou seja, valores de *pixels* em uma pequena janela (3x3, 5x5 ou 7x7) são observados nas imagens originais A e B. Na imagem interpolada, o *pixel* central da janela é estimado com os valores das janelas observadas (Figura 2.4). As janelas são simultaneamente deslizadas sobre as imagens observadas para obter o valor estimado de cada *pixel* da imagem interpolada.

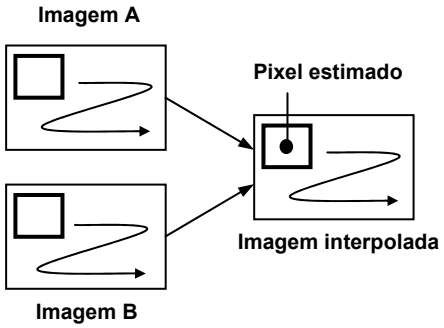


FIGURA 2.4 – Interpolação local com janelas deslizantes.

O método de interpolação 3D original, implementado em MATLAB, foi convertido para a linguagem IDL para ser usado neste projeto de mestrado. As seções seguintes descrevem o método de interpolação 3D e a implementação IDL.

2.2.1 Estimação de parâmetros estatísticos

As informações estatísticas necessárias para a interpolação 3D de imagens usando a Teoria de Estimação Bayesiana devem ser estimadas. Estas informações compreendem a *média amostral* e *variância amostral* das imagens observadas A e B, e a *matriz de covariância amostral* entre as duas imagens observadas. Estes parâmetros são descritos nas seções 2.2.1.1 e 2.2.1.2.

A informação estatística sobre a dependência do *pixel* central a ser estimado com os *pixels* das imagens observadas é o *vetor de covariâncias*, descrito na seção 2.2.1.3. Como não há uma imagem *a priori*, a informação estatística sobre a imagem interpolada foi definida como a *média* e a *variância*, calculadas em função das imagens observadas, conforme descrito na seção 2.2.1.1.

2.2.1.1 Média amostral e variância amostral das imagens observadas e da imagem a priori

Seja uma variável aleatória $\mathbf{X} = [X_1 \ldots X_n]^T$ de n observações denotadas pelas variáveis aleatórias X_i, com i = 1,...,n. A média amostral $\hat{\mu}$ e variância amostral S^2 de **X** foram definidas nas equações (2.4) e (2.6), respectivamente como

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} X_i \quad \text{e} \quad S^2 = \frac{1}{n-1}\sum_{i=1}^{n}\left(X_i - \hat{\mu}\right)^2, \qquad \text{com} \quad i = 1, \ldots, n.$$

Considere as imagens observadas como processos estocásticos estacionários, descritos em cada *pixel* pelas variáveis aleatórias A e B. O valor da *i*-ésima observação X_i corresponde à intensidade do *i*-ésimo *pixel* da imagem A ou B. No programa IDL*, a seqüência de comandos

```
mA = mean(A)                ; média da imagem A.
mB = mean(B)                ; média da imagem B.

vA = variance(A)            ; variância da imagem A.
vB = variance(B)            ; variância da imagem B.
```

calcula a média amostral e variância amostral sobre as imagens inteiras.

Como não existe uma imagem *a priori* da imagem interpolada, a informação *a priori* foi especificada pela média e variância da imagem, calculada em função das imagens observadas. Assim, a média *a priori* foi definida como a média aritmética das médias amostral das imagens observadas. A variância foi definida como a média geométrica das variâncias amostral das imagens observadas. Isto pode ser calculado em IDL, com

```
mC = (mA + mB) / 2          ; média da imagem a priori.
vC = sqrt(vA * vB)          ; variância da imagem a priori.
```

Os valores de média amostral devem ser subtraídos das imagens observadas para obter imagens de média zero e facilitar o cálculo da matriz de covariância. No final do processo de estimação, o valor de média amostral da imagem *a priori* deve ser adicionado à imagem interpolada.

* Ver Apêndice D: Código fonte IDL.

2.2.1.2 Matriz de covariância amostral entre imagens observadas

A matriz de covariância amostral **K** foi definida na equação (2.8) como

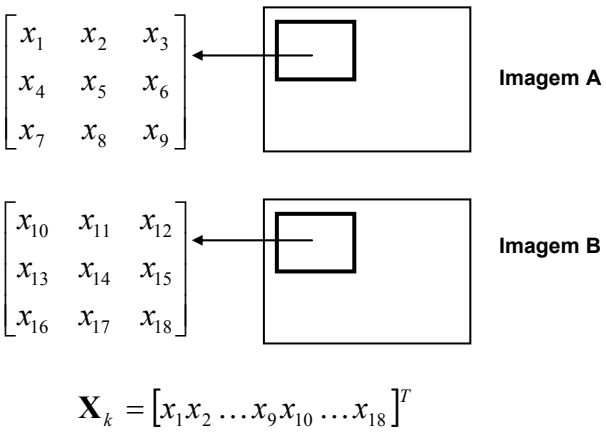
$$\hat{\mathbf{K}} = \frac{1}{N}\sum_{k=1}^{N}(\mathbf{X}_k - \hat{\boldsymbol{\mu}})(\mathbf{X}_k - \hat{\boldsymbol{\mu}})^T,$$

onde N é o número de vetores observados, $\mathbf{X}_k = [X_1 \ldots X_n]^T$ é o k-ésimo vetor observado, e $\hat{\boldsymbol{\mu}} = [\mu_1 \ldots \mu_n]^T$ é o vetor médio amostral de **X,** definido na equação (2.7).

Como o valor de média amostral já foi subtraído de cada imagem observada, o vetor médio amostral $\hat{\boldsymbol{\mu}}$ das observações **X** é um vetor nulo. Assim, a matriz de covariância amostral $\mathbf{K}_{xx}$ das imagens observadas pode ser expressa por

$$\mathbf{K}_{xx} = \frac{1}{N}\sum_{k=1}^{N}\mathbf{X}_k\mathbf{X}_k^T \tag{2.39}$$

Os valores X_i do vetor observado $\mathbf{X}_k = [X_1 \ldots X_n]^T$ são obtidos das janelas de observação das imagens A e B, lexicograficamente ordenados por colunas, linhas e imagens, como mostra a Figura 2.5. Por exemplo, para uma janela de observação 3x3 em cada uma das imagens, o vetor coluna $\mathbf{X}_k = [X_1 \ldots X_n]^T$ tem 18 variáveis aleatórias X_i. A variável X_i assume o valor de intensidade do *pixel* x_i, com i = 1,...,9 para *pixels* da janela A e i = 10,...,18 para *pixels* da janela B. Para janelas 5x5, o tamanho do vetor $\mathbf{X}_k$ é 50, sendo 25 *pixels* da janela A e 25 da janela B. Para janelas 7x7, o tamanho do vetor é 98, 49 *pixels* de cada janela.



$$\mathbf{X}_k = [x_1 x_2 \ldots x_9 x_{10} \ldots x_{18}]^T$$

FIGURA 2.5 – Janelas de observação 3x3.

Deslizando simultaneamente as janelas de observação sobre as imagens A e B, obtemos N vetores $\mathbf{X}_k$, com $k = 1,...,N$. O número de vetores de observação depende do tamanho das imagens e do tamanho das janelas de observação.

Dado os vetores de observação, podemos estimar a matriz de covariância usando a equação (2.39). A matriz de covariância amostral $\mathbf{K}_{xx}$ tem tamanho 18x18 para janelas 3x3. O cálculo resulta em uma matriz de covariância

$$\mathbf{K}_{xx} = \begin{bmatrix} vA & \text{cov}(x_1,x_2) & ... & \text{cov}(x_1,x_9) & corrAB & \text{cov}(x_1,x_{11}) & ... & \text{cov}(x_1,x_{18}) \\ \text{cov}(x_2,x_1) & vA & ... & \text{cov}(x_2,x_9) & \text{cov}(x_2,x_{10}) & corrAB & ... & \text{cov}(x_2,x_{18}) \\ ... & ... & ... & ... & ... & ... & ... & ... \\ \text{cov}(x_9,x_1) & ... & ... & vA & \text{cov}(x_9,x_{10}) & ... & ... & corrAB \\ corrAB & \text{cov}(x_{10},x_2) & ... & \text{cov}(x_{10},x_9) & vB & \text{cov}(x_{10},x_{11}) & ... & \text{cov}(x_{10},x_{18}) \\ \text{cov}(x_{11},x_1) & corrAB & ... & \text{cov}(x_{11},x_9) & \text{cov}(x_{11},x_{10}) & vB & ... & \text{cov}(x_{11},x_{18}) \\ ... & ... & ... & ... & ... & ... & ... & ... \\ \text{cov}(x_{18},x_1) & \text{cov}(x_{18},x_2) & ... & corrAB & \text{cov}(x_{18},x_{10}) & \text{cov}(x_{18},x_{11}) & ... & vB \end{bmatrix},$$

onde

$\text{cov}(x_i, x_i) = \text{vA},$ para $i = 1, ..., 9.$

$\text{cov}(x_i, x_i) = \text{vB},$ para $i = 10,...,18.$

$\text{cov}(x_i, x_j) = \text{cov}(x_j, x_i) = \text{corrAB},$ para $i = 1,..., 9$ e $j = 10,..., 18$, sendo i correspondente a j.

$\text{cov}(x_i, x_j) = \text{cov}(x_j, x_i),$ para os elementos fora das diagonais.

Assim, os valores dos elementos das diagonais do primeiro e segundo quadrante da matriz correspondem ao valor estimado da variância das imagens A e B, respectivamente. Os valores dos elementos das diagonais do segundo e terceiro quadrante correspondem à correlação espectral entre as imagens A e B. Todos os elementos fora da diagonal possuem um valor correspondente à covariância entre o *pixel* x_i e x_j, com $i, j = 1,...,18$.

O cálculo da matriz de covariância amostral foi implementado em IDL usando a seqüência de comandos

```
n = 0.0
for i = 1, nx - 4 do begin
   for j = 1, ny - 4 do begin
      A1 = A[i:i+2, j:j+2]        ; janela 3x3 da imagem A.
      B1 = B[i:i+2, j:j+2] ;      ; janela 3x3 da imagem B.
      A9 = reform(A1,9)           ; vetor linha com 9 pixels de A.
      B9 = reform(B1,9)           ; vetor linha com 9 pixels de B.
      X[0:8]  = A9                ; vetor de observação com 18 pixels.
      X[9:17] = B9
      Xt = transpose(X)           ; vetor coluna com 18 pixels.
      t  = Xt ## X                ; multiplicação de vetores.
      Kxx = Kxx + t               ; somatória de matrizes (18x18).
      n = n + 1.0                 ; número de vetores observados.
   end
end

Kxx = Kxx / n                     ; matriz de covariância amostral.
```

2.2.1.3 Vetor de covariâncias entre pixels estimados e pixels observados

A informação estatística sobre a dependência entre o *pixel* a ser estimado e os *pixels* observados nas imagens A e B foi definida por um vetor de covariâncias $\mathbf{k}_{YX}$. O vetor de covariâncias foi postulado para ser separável no domínio espacial e espectral. Os coeficientes de correlação espacial nas direções horizontal ρ_h e vertical ρ_v foram selecionados para valores entre 0,95 e 1,0. O coeficiente de correlação espectral 3D entre o *pixel* a ser estimado e os *pixels* correspondentes nas imagens observadas foram supostos como a raiz quadrada do coeficiente de correlação entre as imagens observadas, experimentalmente estimados por valores positivos. Em IDL, calculamos

```
corrAB = correlate(A, B)   ; Coeficiente de correlação espectral entre
                           ; imagens observadas.
corr3D = sqrt(corrAB)      ; Coeficiente de correlação espectral entre
                           ; uma imagem observada e a interpolada.
```

Como o coeficiente de correlação ρ_{ij} entre os elementos i e j pode ser expresso por

$$\rho_{ij} = \frac{\mathrm{cov}(i,j)}{\sqrt{v_i . v_j}},$$

então, a covariância entre uma imagem observada e a imagem interpolada é calculada por

```
covAC = corr3D * sqrt(vA*vC)      ; Covariância entre a imagem A e a
                                  ; imagem interpolada.
covBC = corr3D * sqrt(vB*vC)      ; Covariância entre a imagem B e a
                                  ; imagem interpolada.
```

Os coeficientes de correlação espacial são determinados pelas distâncias entre a posição do *pixel* a ser estimado e a posição correspondente dos *pixels* em cada uma das janelas de observação (Figura 2.5). Por exemplo, o *pixel* central y está distante do *pixel* x_1 (ou x_{10} na imagem B) um *pixel* na direção horizontal e um na direção vertical. Assim, a covariância entre y e x_1 (ou x_{10}) foi postulada como

$$\mathrm{cov}(y, x_1) = \mathrm{covAC} \,.\, \rho_h.\rho_v$$

$$\mathrm{cov}(y, x_{10}) = \mathrm{covBC} \,.\, \rho_h.\rho_v$$

Desta forma, o vetor de covariâncias para uma janela de observação 3x3 fica determinado por

$$\mathbf{k}_{YX} = \left[\mathrm{cov}(y,x_1) \quad \ldots \quad \mathrm{cov}\,AC \quad \ldots \quad \mathrm{cov}(y,x_9) \quad \mathrm{cov}(y,x_{10}) \quad \ldots \quad \mathrm{cov}\,BC \quad \ldots \quad \mathrm{cov}(y,x_{18})\right],$$

onde cov(y, x_i) é especificada pela covariância espectral e espacial, de acordo com a distância entre o *pixel* central y e o *pixel* x_i nas imagens observadas. Em IDL, o cálculo do vetor de covariâncias foi implementado como segue

```
roh = 1.0                  ; coeficiente de correlação horizontal.
rov = 1.0                  ; coeficiente de correlação vertical.

cAh  = covAC*roh
cAv  = covAC*rov
cAhv = covAC*roh*rov
cBh  = covBC*roh
cBv  = covBC*rov
cBhv = covBC*roh*rov

; Vetor de covariâncias entre uma imagem observada e o
; ponto a ser estimado.
kyx = [ cAhv, cAv, cAhv, cAh, covAC, cAh, cAhv, cAv, cAhv,
        cBhv, cBv, cBhv, cBh, covBC, cBh, cBhv, cBv, cBhv ]
```

2.2.2 Interpolador linear médio-quadrático

Seja **X** o vetor de observações nas janelas 3x3 das imagens A e B, e Y a variável aleatória escalar do *pixel* a ser estimado no centro da janela. De acordo com o princípio da ortogonalidade, o *estimador linear de mínimo erro médio-quadrático* de Y, baseado no vetor observado **X,** é dado por

$$Y = a^T \mathbf{X}, \tag{2.40}$$

onde

$a^T = \mathbf{k}_{YX}(\mathbf{K}_{XX})^{-1}$ é um vetor coluna de 18 coeficientes que minimizam o erro da estimação,

$\mathbf{k}_{YX} = E[Y\mathbf{X}^T]$ é o vetor de covariâncias entre o *pixel* a ser estimado e as imagens observadas,

$\mathbf{K}_{XX}^{-1} = E[\mathbf{X}\mathbf{X}^T]^{-1}$ é a matriz inversa da matriz de covariância amostral entre as imagens observadas.

Os coeficientes *a* foram calculados em IDL com

```
invKxx = invert(Kxx)        ; matriz inversa de Kxx.
ax = kyx ## invKxx          ; coeficientes procurados.
```

Depois de encontrados os coeficientes, os *pixels* da imagem interpolada devem ser estimados usando a equação (2.40). Deslizando as janelas de observações nas imagens A e B, obtemos *N* vetores **X** para interpolar cada *pixel* central das *N* janelas de observação.

A maneira mais eficiente de fazer isso em IDL é transformando o vetor de coeficientes *a* em duas matrizes 3x3. Os primeiros nove elementos do vetor *a* são multiplicados ponto a ponto pelos elementos da janela A e os últimos nove elementos de *a* são multiplicados pelos elementos da janela B. Para completar o processo de estimação, a média amostral da imagem interpolada é adicionada à imagem.

```
a1 = reform(ax,9,2)         ; transforma vetor linha 1x18 em matriz 2x9.
a2 = a1[*,0]                ; vetor linha com primeiros 9 elementos.
a3 = a1[*,1]                ; vetor linha com últimos 9 elementos.
a4 = reform(a2,3,3)         ; transforma vetor linha 1x9 em matriz 3x3.
a5 = reform(a3,3,3)

ITP = convol(A,a4) + convol(B,a5) ; multiplicação ponto a ponto
                                  ; pelos coeficientes 3x3.

ITP = ITP + mC              ; imagem interpolada final.
```

O erro médio-quadrático da estimação é obtido por $\mathrm{MSE} = \mathrm{var}(Y) - \mathbf{k}_{YX}(\mathbf{K}_{XX})^{-1}(\mathbf{k}_{YX})^T$, e calculado por

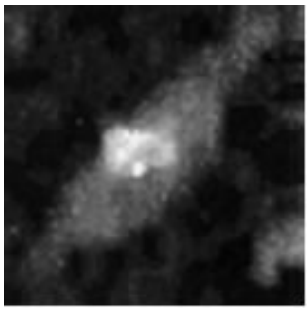
```
erro = vC - (kyx ## invKxx ## TRANSPOSE(kyx)) ; erro de interpolação.
```

2.3 Aplicações de Interpolação 3D de Imagens

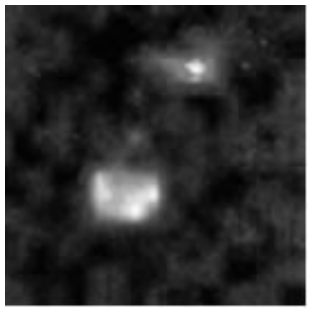
O método de interpolação 3D foi testado por Mascarenhas *et al*. para interpolar uma imagem intermediária entre as imagens originais de um arco coronal e de um tomate. Para usar este método em reconstrução 3D de imagens, seria necessário gerar várias imagens intermediárias.

2.3.1 Imagens de raios-X de um arco coronal

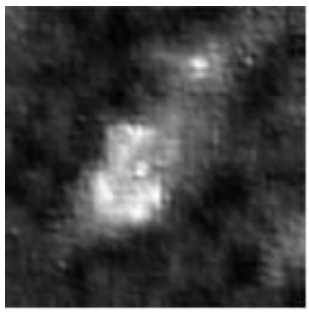
Imagens do *loop* em duas diferentes profundidades da atmosfera solar foram interpoladas usando uma janela de observação 3x3, com coeficientes de correlação espacial igual a 0,99 ou 1,0. A Figura 2.6 mostra as imagens originais do topo e da base do *loop* e as imagens interpoladas.



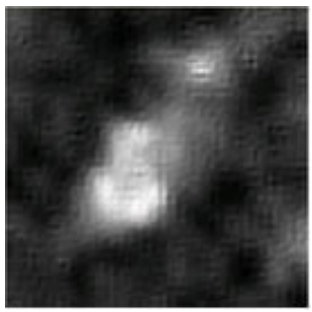
(a) Imagem A (topo do *loop*).



(b) Imagem B (base do *loop*).



(c) Interpolada com coeficiente 0,99



(d) Interpolada com coeficiente 1,0.

FIGURA 2.6 – Imagens originais e interpoladas com janela de observação 3x3.

A média a priori da imagem interpolada é a média aritmética das médias das imagens observadas e, a variância a priori é a média geométrica das variâncias das imagens observadas.

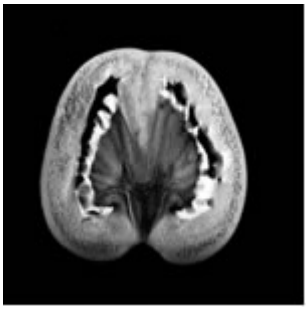
Imagem	Média	Variância
A	43	1603
B	39	1153
Interpolada	41	1360

Os valores da média, variância e erro das imagens interpoladas são:

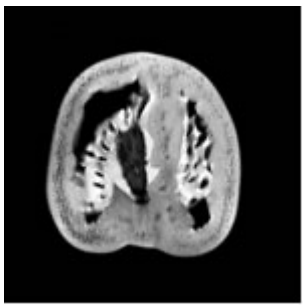
Imagem interpolada com coeficiente de correlação	Média	Variância
0,99	41	674
1,00	41	707

2.3.2 Imagens de ressonância magnética de um tomate

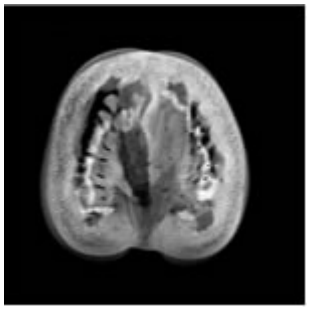
As duas imagens do tomate, cedidas pela Embrapa Instrumentação Agropecuária (São Carlos), foram interpoladas com janelas de observação 3x3 e coeficiente de correlação igual a 0,99 e 1,0.



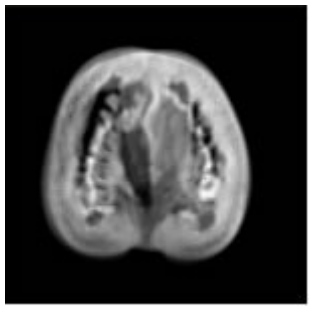
(a) Imagem original A.



(b) Imagem original B.



(c) Interpolada com coeficiente 0,99.



(d) Interpolada com coeficiente 1,0.

FIGURA 2.7 – Imagens originais de ressonância magnética de um tomate e imagens interpoladas.

As médias e variâncias das imagens são:

Imagem	Média	Variância
A	49	5271
B	53	6302
Interpolada	51	5764

Os valores da média e variância das imagens interpoladas são:

Imagem interpolada com coeficiente de correlação	Média	Variância
0,99	52	4775
1,0	52	4931

O método de interpolação de imagens apresentado neste capítulo baseia-se na interpolação da intensidade dos *pixels*. No próximo capítulo, um método de metamorfose de imagens para interpolação de forma e cor dos pixels é apresentado.

3

Metamorfose de Imagens

Metamorfose de imagens ou *morphing* é uma técnica usada para obter a transformação de uma imagem em outra [35]. As técnicas de *morphing* de imagens foram largamente usadas na indústria de entretenimento para criar efeitos especiais em comerciais de televisão, vídeo clips e filmes. O processo de *morphing* consiste de uma combinação de deformação de imagens e interpolação de cor para gerar uma seqüência de *n* imagens intermediárias entre a imagem inicial A e final B (Figura 3.1). Quando as imagens intermediárias são vistas seqüencialmente, a imagem inicial se transforma na imagem final, com uma mudança progressiva de forma e cor. Nas próximas seções, é apresentado um método simples de interpolação de imagens, além de técnicas de deformação e metamorfose de imagens.

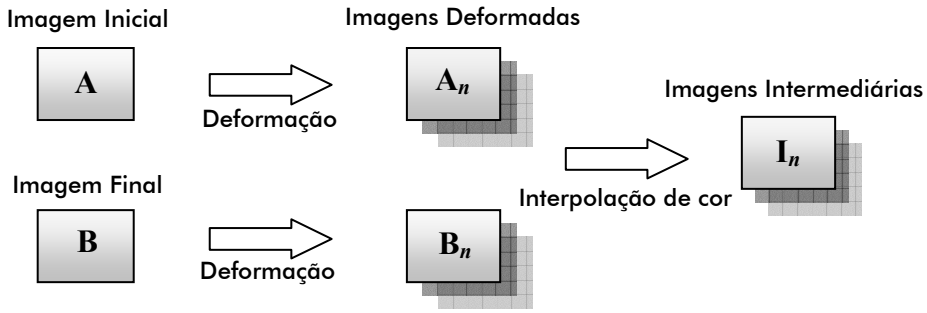
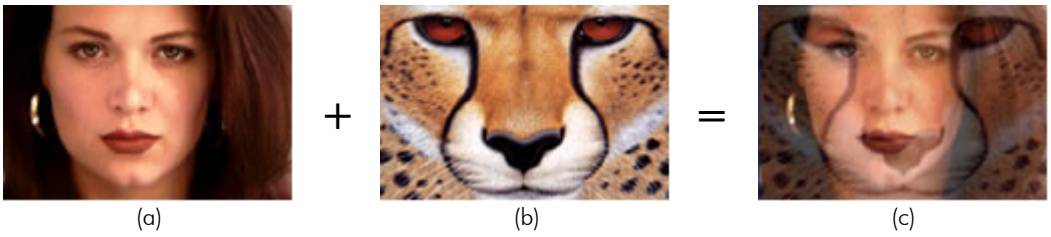


FIGURA 3.1– Processo de metamorfose de imagens.

3.1 Interpolação de Imagens

Uma técnica simples de interpolação de imagem gera uma imagem de saída através da mistura de cores das imagens de entrada. O valor de cada *pixel* na imagem de saída é uma combinação dos valores dos *pixels* correspondentes nas imagens de entrada. Os coeficientes da combinação de cor definem a proporção da contribuição de cada imagem. Estas proporções são aplicadas de forma que os valores dos *pixels* não excedam o valor máximo. Dadas as imagens A e B nas Figuras 3.2(a) e 3.2(b), uma nova imagem com 50% de cor da imagem A e 50% da imagem B é calculada por $I(x, y) = 0{,}5.A(x, y) + 0{,}5.B(x, y)$. A Figura 3.2(c) mostra o resultado da interpolação.



Fonte: GOMES, 1998.

FIGURA 3.2 – Combinação de cor das imagens.

Uma técnica de animação bastante simples, chamada *cross-dissolve*, gera uma seqüência de imagens intermediárias com a mistura de cores das imagens inicial A e final B, para obter um efeito de transição suave entre as duas imagens [5].

As imagens intermediárias são geradas por uma combinação $I_t(x, y) = f(t).A(x, y) + g(t).B(x, y)$, tal que $f(0) = 1$, $g(0) = 0$, $f(1) = 0$, $g(1) = 1$ e $f(t) + g(t) = 1$, para todo t variando de 0 a 1. Uma operação típica é o *cross-dissolve* com interpolação linear definido por $I_t(x, y) = (1- t).A(x, y) + t.B(x, y)$.

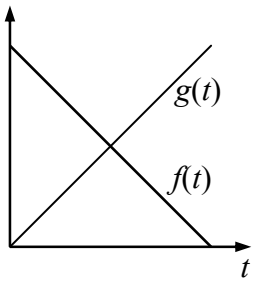
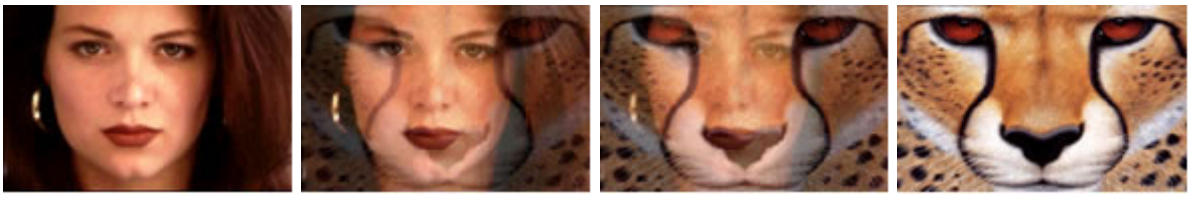


FIGURA 3.3 – Funções de cross-dissolve linear.

No *cross-dissolve* linear, as proporções da mistura entre as imagens A e B mudam linearmente de 100% de A e 0% de B para 0% de A e 100% de B. Desta forma, quando as imagens intermediárias são mostradas seqüencialmente, a imagem A é continuamente atenuada, enquanto a imagem B aumenta em intensidade. A Figura 3.4 mostra um exemplo de animação usando *cross-dissolve* linear. As imagens 3.4(a) e 3.4(d) representam as imagens A e B, respectivamente. A imagem 3.4(b) foi gerada com 60% da informação de cor da imagem A e 40% da imagem B; a imagem 3.4(c) foi gerada com 40% de A e 60% de B.



(a) (b) (c) (d)

Fonte: GOMES, 1998.

FIGURA 3.4 – Seqüência de imagens geradas com cross-dissolve linear.

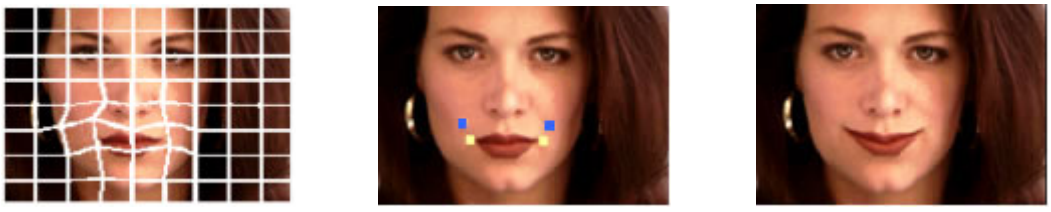
A técnica *cross-dissolve* foi usada por décadas pela indústria de filme e vídeo, para obter efeitos de transição entre dois objetos diferentes com formas similares. No entanto, esta e outras técnicas de interpolação de intensidade ou cor dos *pixels* da imagem, como o método de *Interpolação 3D de Imagens usando a Teoria de Estimação Bayesiana*, não apresentam um resultado realístico quando as duas imagens têm formas diferentes. Isto pode ser observado na seqüência de imagens acima. Neste caso, técnicas baseadas em interpolação de forma, como *morphing* de imagens, podem ser usadas para melhorar a qualidade da animação.

Uma técnica de metamorfose de imagem consiste de uma progressiva deformação das imagens inicial e final, seguida de interpolação de cor das imagens deformadas, para gerar as imagens intermediárias da seqüência de animação. A seguir, os conceitos básicos de deformação de imagens são apresentados.

3.2 Deformação de Imagens

Deformação de imagens ou *warping 2D* é uma transformação geométrica bidimensional que gera a distorção de uma imagem [36]. Este tipo de transformação possui diversas aplicações, como por exemplo, em sensoriamento remoto para corrigir distorções causadas por dispositivos de imageamento por satélite e, principalmente, na indústria de entretenimento para gerar efeitos especiais em filmes e comerciais.

Uma deformação de imagem pode ser diretamente especificada por uma função de mapeamento que estabelece a correspondência espacial entre todos os pontos da imagem de entrada e os pontos da imagem deformada. Cada *pixel* da imagem de entrada é mapeada para uma nova posição, gerando uma imagem deformada. Existem duas classificações de técnicas de deformação: *deformação baseada em malha* e *deformação baseada em característica*. Na deformação baseada em malha, uma malha irregular define a transformação de deformação. Na deformação por características, somente as características da imagem são especificadas. Um caso importante de *warping* baseado em características é o da especificação baseada em pontos, onde cada característica é indicada por pares de pontos de controle na imagem. A Figura 3.5(c) mostra o resultado da deformação de uma imagem, usando uma especificação por malha ou pontos, como mostram as Figuras 3.5(a) e 3.5(b).



(a) (b) (c)

FIGURA 3.5 – Métodos de deformação de imagem baseados em malha ou pontos.

Em técnicas de metamorfose de imagens, a deformação baseada em malhas ou características é usada numa etapa preliminar da interpolação de cor, para geração das imagens intermediárias.

3.3 Metamorfose de Imagens

Metamorfose ou *morphing* de imagens é uma técnica usada para fazer a metamorfose de uma imagem em outra [35]. A idéia é gerar uma seqüência de imagens intermediárias que representam uma transformação suave de forma e cor de uma imagem para outra. O processo de *morphing* consiste de deformação, de modo que as duas imagens tenham a mesma forma, seguida de interpolação de cor das imagens deformadas [9, 25]. A Figura 3.6 ilustra o processo de deformação de duas imagens, combinada com interpolação de cor das imagens deformadas para gerar uma nova imagem.

Fonte: GOMES, 1998.

FIGURA 3.6 – Deformação de imagens e interpolação de cor das imagens deformadas.

A seqüência de imagens da animação é gerada por uma deformação que distorce progressivamente a primeira imagem na segunda. Uma deformação inversa distorce a segunda imagem na primeira. Simultaneamente ao processo de deformação, um *cross-dissolve* entre a primeira imagem, gradualmente distorcida e desvanecida, e a segunda imagem, no início totalmente distorcida e desvanecida, gera as imagens intermediárias finais. Desta forma, as contribuições de cor das imagens deformadas são amenizadas quando estão muito distorcidas. Quando vistas em seqüência, as imagens intermediárias produzem uma animação da primeira imagem se transformando suavemente na segunda (Figura 3.7).

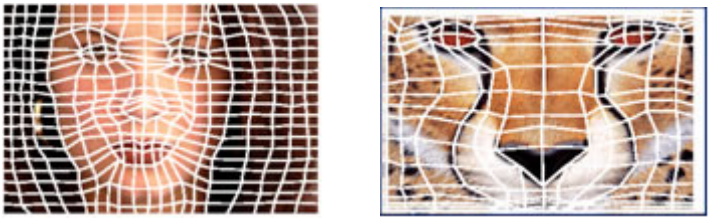
Fonte: GOMES, 1998.

FIGURA 3.7 – Seqüência de imagens geradas com morphing.

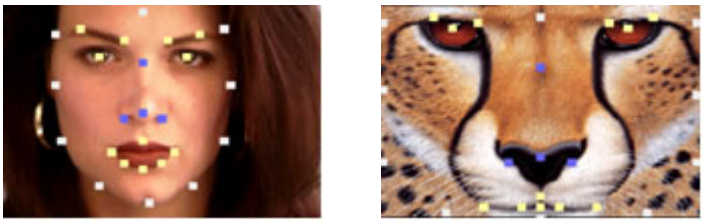
O problema principal de *morphing* de imagens é, basicamente, como as imagens deformadas são efetivamente geradas. Dependendo do método de deformação, as técnicas de *morphing* podem ser classificadas em: métodos baseados em malha (*mesh-based*) e métodos baseados em característica (*feature-based*).

Nos métodos baseados em malha [5], a deformação é especificada por duas malhas não-uniformes. As duas malhas têm o mesmo tamanho e cada ponto em uma malha corresponde a um ponto na outra malha. A deformação usando malhas pode não ser eficientemente controlada em imagens muito complexas, porque com o aumento da malha, muitos pontos podem não corresponder às características nas imagens.

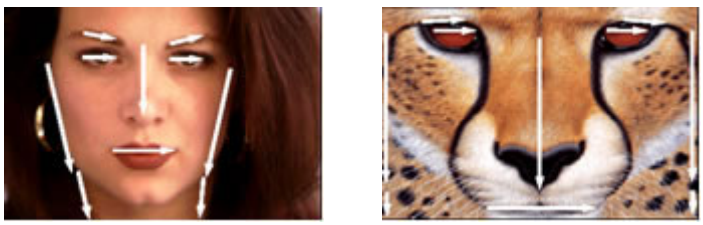
Métodos baseados em característica solucionam o problema de métodos baseados em malha, especificando as características em uma imagem com um conjunto de pontos ou segmentos de linha [2, 8]. A Figura 3.8 ilustra as especificações de deformação nos métodos baseados em malha e métodos baseados em característica com pontos ou linhas.



(a) *Morphing* baseado em malhas.



(b) *Morphing* baseado em pontos.



(c) *Morphing* baseado em linhas.

Fonte: GOMES, 1998.

FIGURA 3.8 – Métodos de morphing de imagens baseados em malha ou característica.

Nos métodos baseados em característica [2], uma deformação é gerada pela correspondência entre dois conjuntos de pontos ou linhas. Para cada ponto ou linha no conjunto de características de uma imagem, o ponto ou linha correspondente na outra imagem especifica a posição na imagem distorcida. Para aumentar o controle de deformação em determinada área, basta adicionar novos pares de pontos ou linhas nas características das imagens.

3.3.1 Metamorfose de imagens baseada em pontos

Para obter a metamorfose de imagens, usando o método de deformação baseada em pontos, um conjunto de pares de pontos de correspondência $\{(\mathbf{p}_i, \mathbf{q}_i) \mid \mathbf{p}_i, \mathbf{q}_i \in \mathbb{R}^2, i \in 1,\ldots,n\}$ deve ser especificado para definir a correspondência entre as características das imagens, onde $\mathbf{p}_i$ indica uma posição na imagem inicial que corresponde a uma posição $\mathbf{q}_i$ na imagem final.

As novas posições de todos os *pixels* nas imagens intermediárias devem ser calculadas, tal que a posição dos *pixels* nos pontos de controle da imagem inicial sejam progressivamente deslocados para a posição dos pontos de controle correspondente na imagem final, e todos os outros *pixels* sejam movidos de uma maneira consistente para que as relações de vizinhança entre *pixels* permaneçam inalteradas.

Este problema de deformação geométrica pode ser tratado como um problema de interpolação de dados esparsos (*scattered data interpolation*) [26]. Existem vários métodos de deformação baseada em métodos de interpolação de dados esparsos, tais como, métodos baseados em triangulação, interpolação ponderada pela distância inversa e funções base radial, entre outros [25]. O método de deformação usando interpolação ponderada pela distância inversa é apresentado a seguir.

3.3.2 Deformação com interpolação ponderada pela distância inversa

Um método de interpolação ponderada pela distância inversa, originalmente proposto por Shepard [27], pode ser aplicado no problema de deformação de imagem, conforme descrito a seguir.

Dado n pares ($\mathbf{p}_i$, $\mathbf{q}_i$) de pontos de controle, uma função de mapeamento $f: \mathbb{R}^2 \rightarrow \mathbb{R}^2$ deve ser encontrada, com $f(\mathbf{p}_i) = \mathbf{q}_i$ e $i = 1,\ldots,n$. Esta função de mapeamento especifica a nova posição para todos os *pixels* na imagem deformada [8, 9].

Para cada par de pontos de controle, um interpolador local $f_i: \mathbb{R}^2 \rightarrow \mathbb{R}^2$ com $f_i(\mathbf{p}_i) = \mathbf{q}_i$, é determinado. Interpoladores locais lineares são normalmente usados para deslocar os pontos de controle com uma transformação linear. O deslocamento dos *pixels* vizinhos é controlado pela função de mapeamento que depende da distância do *pixel* para cada um dos pontos de controle da imagem.

A função de mapeamento $f: \mathbb{R}^2 \rightarrow \mathbb{R}^2$ é uma média ponderada dos interpoladores locais, com pesos w_i dependendo da distância do *pixel* aos pontos de controle. Assim, a função ponderada pela distância inversa é dada por

$$f(x,y) = \sum_{i=1}^{n} w_i(x,y) f_i(x,y), \quad \text{com } f_i(\mathbf{p}_i) = \mathbf{q}_i, \ i = 1,...,n.$$

A função peso $w_i: \mathbb{R}^2 \rightarrow \mathbb{R}$ depende da distância inversa de um *pixel* $\mathbf{p}(x, y)$ ao ponto de controle $\mathbf{p}_i(x_i, y_i)$, normalizada pela soma das distâncias inversas de todos os pontos de controle, sendo definida por

$$w_i(x,y) = \frac{\sigma_i(x,y)}{\sum_{j=1}^{n} \sigma_j(x,y)}, \qquad \text{com } \sigma_i(x,y) = \frac{1}{d_i(x,y)},$$

onde $d_i(x, y)$ é a distância euclidiana entre o *pixel* $\mathbf{p}(x, y)$ e o ponto de controle $\mathbf{p}_i(x_i, y_i)$.

A função peso deve satisfazer as seguintes condições:

$$(1)\ w_i \geq 0 \ \text{ e } \ \sum_{i=1}^{n} w_i(x,y) = 1;$$

$$(2)\ w_i(x_i,y_i) = 1, \ \ w_i(x_j,y_j) = 0, \ \text{com } i \neq j, \ i,j = 1,...,n.$$

Estas condições garantem que o *pixel* no ponto de controle $\mathbf{p}_i$ seja deslocado para a posição $\mathbf{q}_i$, uma vez que $f(\mathbf{p}_i) = f_i(\mathbf{p}_i) = \mathbf{q}_i$, para todo $i = 1,\ldots,n$. Os *pixels* mais distantes de um ponto de controle $\mathbf{p}_i$ não são afetados pelo deslocamento deste ponto, pois o peso w_i correspondente é pequeno.

A Figura 3.9 ilustra o processo de mapeamento de um *pixel* $\mathbf{p}(x, y)$ para uma nova posição $\mathbf{p'}(x', y')$. A posição (x', y') é calculada por uma função de mapeamento definida por $f(x, y) = w_0 f_0 + w_1 f_1 + w_2 f_2$, com $f_0(\mathbf{p}_0) = \mathbf{q}_0$, $f_1(\mathbf{p}_1) = \mathbf{q}_1$, $f_2(\mathbf{p}_2) = \mathbf{q}_2$, sendo que $w_1 > w_0 > w_2$ e $w_0 + w_1 + w_2 = 1$. Para cada *pixel* da imagem, uma função de mapeamento é determinada com pesos correspondentes as distâncias d_0, d_1 e d_2 do *pixel* aos pontos de controle p_0, p_1 e p_2.

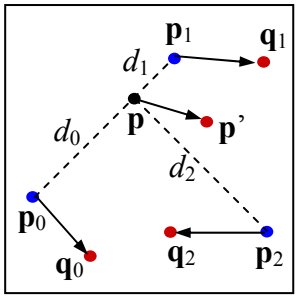


FIGURA 3.9 – Deslocamento de um pixel com interpolação ponderada pela distância inversa.

3.3.3 Metamorfose com interpolação ponderada pela distância inversa

No processo de metamorfose, uma deformação parcial das imagens inicial e final é realizada progressivamente para gerar pares de imagens deformadas. A imagem inicial é gradualmente deformada na imagem final e, simultaneamente, uma deformação inversa deforma a imagem final na imagem inicial. Ao mesmo tempo, uma interpolação de cor entre cada par de imagens deformadas gera as imagens intermediárias entre as imagens inicial e final.

Para deformação parcial das imagens, pontos de controle intermediários entre os pontos de controle ($\mathbf{p}_i$, $\mathbf{q}_i$) são determinados para cada imagem intermediária. Normalmente, os pontos de controle intermediários são

calculados por um interpolador local linear, dado pela inclinação da reta entre os pontos $\mathbf{p}_i$ e $\mathbf{q}_i$, como mostra a Figura 3.10.

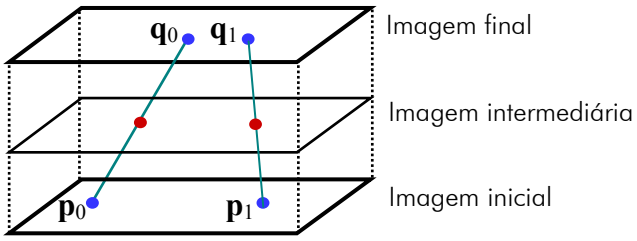


FIGURA 3.10 – Interpolação linear para deformação progressiva das imagens.

Cada *pixel* na posição (x, y) na imagem inicial é movido para uma nova posição $(x_k, y_k) = (x, y) + k.\ \frac{f(x,y)}{N-1}$ na imagem intermediária k, sendo $f(x, y)$ a função linear de interpolação ponderada pela distância inversa, $N-1$ o número total de imagens e $k = 0,..,N$. Um processo inverso é usado para deformar a imagem final na imagem inicial, com $(x_k, y_k) = (x, y) - (N-k).\ \frac{f(x,y)}{N-1}$.

Finalmente, para completar o processo de metamorfose de imagens, basta interpolar uma imagem entre cada par de imagens deformadas para gerar as imagens intermediárias da seqüência de animação. O método de metamorfose de imagens também pode ser usado em aplicações de reconstrução de volume, como discutido no próximo capítulo.

4

Reconstrução de Volume

Em muitas aplicações é necessário usar técnicas de visualização de volume para visualizar a estrutura tridimensional de um objeto, a partir do conjunto de imagens tomográficas de seções transversais deste objeto. O processo básico de visualização volumétrica compreende a fase de formação do volume e a fase de visualização de dados. A fase de formação do volume envolve, principalmente, a aquisição dos dados e a reconstrução do volume.

Em aplicações médicas, uma pilha de imagens de seções transversais de um objeto pode ser adquirida por Tomografia de Ressonância Magnética (MRI), Tomografia Computadorizada de Raios-X (CT) e outras tecnologias de imageamento. Geralmente, a distância entre as imagens tomográficas (fatias do volume) é muito maior que a resolução espacial dentro de cada fatia. Conseqüentemente, a visualização tridimensional do objeto pode apresentar um resultado visual insatisfatório, devido à falta de informação entre as fatias. Neste caso, é necessário um método de reconstrução de volume, onde técnicas de interpolação de imagens são usadas para gerar imagens intermediárias entre cada par de imagens originais.

Após a reconstrução 3D, o volume de dados é representado por um *array* tridimensional de *voxels*, cujos valores de intensidade correspondem a medidas

de densidade do objeto. Finalmente, técnicas de visualização de volume permitem visualizar a estrutura tridimensional do objeto, sob qualquer ângulo de observação no espaço.

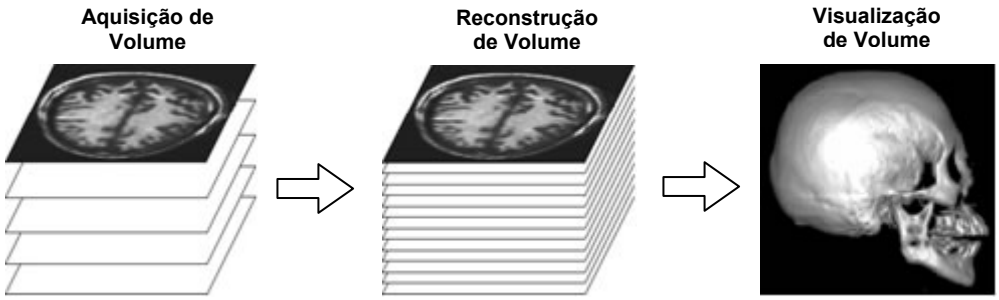


FIGURA 4.1 – Etapas de reconstrução e visualização volumétrica.

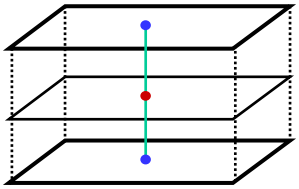
Neste capítulo, são abordados alguns exemplos de reconstrução de volume usando as técnicas de interpolação de imagens *cross-dissolve* e *morphing 2D*. Algumas das técnicas mais usadas de visualização 3D são apresentadas no próximo capítulo.

4.1 Reconstrução 3D de Imagens Tomográficas de Ressonância Magnética

Nesta seção, imagens tomográficas de ressonância magnética de frutas, cedidas pela Embrapa Instrumentação Agropecuária (São Carlos), são reconstruídas usando métodos de interpolação de imagens. Para reconstruir as imagens tomográficas através do método de *Interpolação 3D de Imagens usando a Teoria de Estimação Bayesiana*, seria necessário aprimorar o método de interpolação para gerar várias imagens intermediárias entre cada par de imagens. Assim, apenas o método *cross-dissolve* foi empregado para reconstrução de volume.

O processo de *cross-dissolve* aplicado em reconstrução 3D é ilustrado na Figura 4.2, onde um *pixel* da imagem inferior corresponde ao *pixel* na mesma posição da imagem superior. Desta forma, a cor do *pixel* da imagem intermediária é uma combinação ou mistura das cores dos *pixels* correspondentes das imagens tomográficas.

As imagens intermediárias I_n, com valores de n entre 0 e 1, são obtidas por $I_n = (1 - n)A + n.B$, onde $I_0 = A$ é a imagem inferior e $I_1 = B$, a imagem superior.



$$I_n = (1-n)A + n.B$$

FIGURA 4.2 – Cross-dissolve para reconstrução de volume.

Três imagens originais de seções transversais de um tomate, obtidas por um tomógrafo de ressonância magnética da Embrapa, são mostradas na Figura 4.3. As fatias do tomate têm 256x256 *pixels* com resolução espacial de 1 mm². A distância entre essas fatias é de 3 mm.

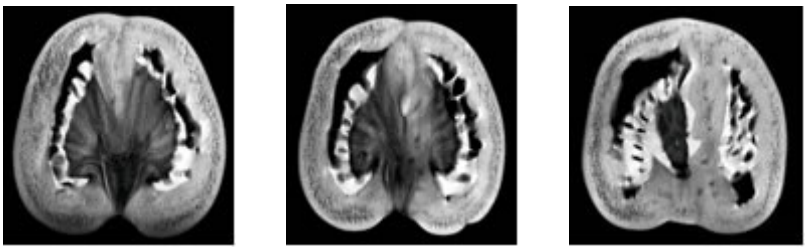


FIGURA 4.3 – Imagens originais de tomografia de ressonância magnética de um tomate.

Para reconstrução 3D do tomate foram geradas cinco fatias entre cada par de imagens originais usando o método *cross-dissolve*[*]. Após a reconstrução, técnicas de visualização de volume permitem visualizar o resultado da reconstrução do tomate.

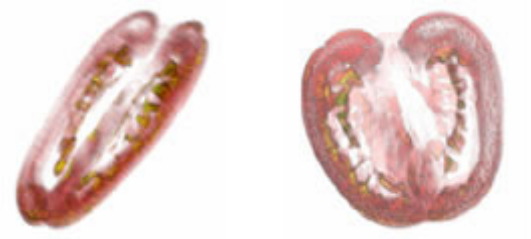


FIGURA 4.4 – Visualização 3D do tomate após reconstrução de volume.

[*] Ver as imagens intermediárias no Apêndice E.

Da mesma forma, um conjunto de 22 imagens de tomografia de ressonância magnética de uma manga foi reconstruído usando *cross-dissolve* entre cada par de imagens originais.[Ŧ]

FIGURA 4.5 – Algumas imagens originais de ressonância magnética da manga.

O resultado da reconstrução 3D da manga é mostrado na Figura 4.6. Entre cada par de imagens foram geradas oito fatias intermediárias.

FIGURA 4.6 – Visualização após reconstrução volumétrica de imagens tomográficas da manga.

Nos dois casos acima, a reconstrução 3D usando um método simples de interpolação de cor, apresenta um resultado visual satisfatório, porque não há uma mudança significativa de forma entre as imagens originais.

[Ŧ] Ver as imagens originais da manga no Apêndice E.

4.2 Reconstrução 3D de Imagens Tomográficas de Raios-X do Arco Coronal

O resultado da reconstrução 3D de um arco coronal da atmosfera solar usando o método *cross-dissolve* é mostrado na Figura 4.7. Foram geradas 70 imagens intermediárias entre as imagens tomográficas de raios-X da base e topo do arco.§ A estrutura tridimensional em forma de arco não é eficientemente reconstruída por métodos de interpolação de cor, devido ao número insuficiente de imagens disponíveis para reconstrução 3D e a necessidade de uma mudança de forma entre as imagens originais.

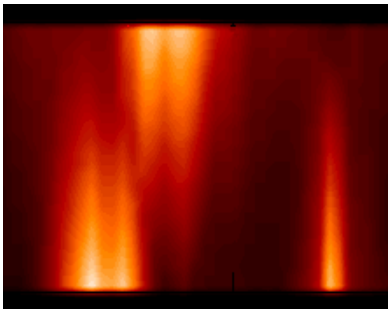


FIGURA 4.7 – Reconstrução 3D do arco coronal usando cross-dissolve.

Uma alternativa é a reconstrução 3D do arco coronal através de uma técnica de metamorfose baseada em pontos, usando o método de interpolação ponderada pela distância inversa. Pares de pontos de controle devem ser especificados nas imagens da base e topo para definir as deformações das imagens (Figura 4.8). Cada par de pontos de controle deve especificar características correspondentes nas imagens da base e topo.

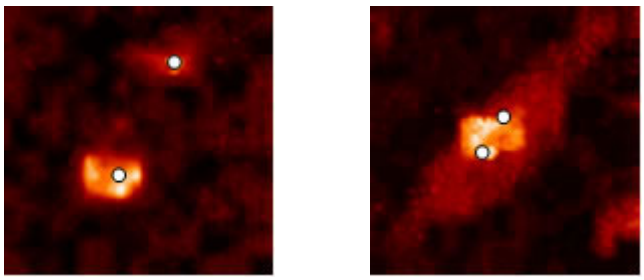


FIGURA 4.8 – Especificação de pontos de controle nas imagens tomográficas do arco coronal para deformação das imagens da base e topo.

§ Ver algumas imagens intermediárias no Apêndice E.

A Figura 4.9 mostra o esquema do método de metamorfose tradicional, que realiza uma deformação linear das imagens da base e topo. Pontos de controle intermediários são criados pela interpolação linear entre os pares de pontos de controle correspondentes. Os *pixels* nos pontos de controle da imagem da base (topo) são deslocados para as posições dos pontos de controle na imagem intermediária *n*, gerando a imagem deformada da base (topo). Os *pixels* mais distantes dos pontos de controle na imagem da base (topo) sofrem um deslocamento menor, ponderado pelas distâncias inversas do *pixel* aos pontos de controle. Desta forma, um par de imagens deformadas da base e topo é gerado para cada imagem intermediária.

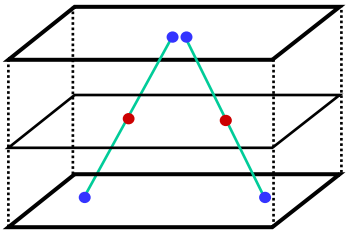


FIGURA 4.9 – Metamorfose com deformação linear.

Cada imagem intermediária, I_n, é determinada pela interpolação de cor das imagens deformadas da base A_n e topo B_n. Desta forma, a cor do *pixel* no ponto de controle da imagem intermediária *n* é uma mistura dos *pixels* que estão nos pontos de controle correspondentes nas imagens da base e topo. O resultado da reconstrução 3D do *loop* é mostrado na Figura 4.10.§

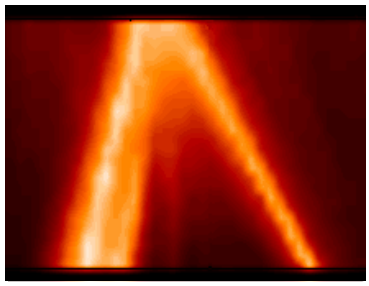


FIGURA 4.10 – Reconstrução 3D do arco coronal usando metamorfose linear.

§ Ver algumas imagens intermediárias no Apêndice E.

Um método de metamorfose, adaptado para reconstrução 3D das imagens do *loop*, foi implementado usando uma função de mapeamento x^2 (e x^4) para controlar a deformação das imagens. O processo de deformação e interpolação de cor é semelhante ao caso linear. Os resultados nas Figuras 4.12(a) e 4.12(b), embora sejam mais realísticos que a deformação linear, não representam todas as formas possíveis de arcos coronais.

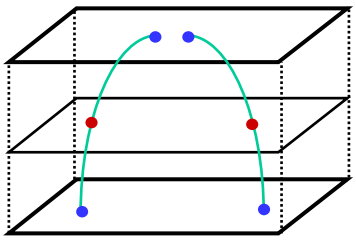
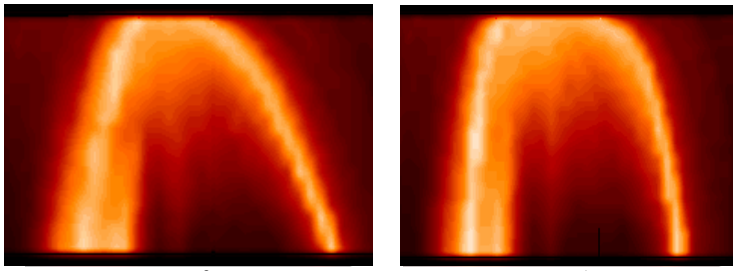


FIGURA 4.11 – Metamorfose com deformação quadrática.



(a) x^2 (b) x^4

FIGURA 4.12 – Reconstrução 3D do arco coronal usando metamorfose quadrática.

Para resolver este problema é necessário especificar uma curva polinomial mais complexa que aproxime o arco magnético e desenvolver um método de metamorfose controlado por esta curva.

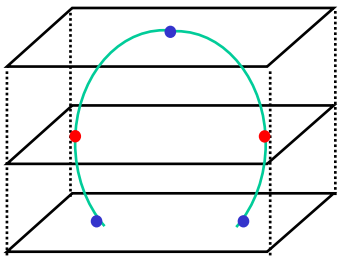


FIGURA 4.13 – Metamorfose com curva polinomial.

Diante desta necessidade, foi desenvolvido um método de reconstrução 3D de arcos coronais, baseado em metamorfose de imagens, controlado por uma curva Bezier. Neste método, a *Interpolação 3D de Imagens usando a Teoria de Estimação Bayesiana* é usada para estimar a intensidade dos *pixels* de cada imagem intermediária a partir dos *n* pares de imagens deformadas da base e topo do *loop*. Este método de reconstrução 3D de arcos coronais, desenvolvido no trabalho de mestrado, será descrito no Capítulo 7.

No próximo capítulo são abordados os principais conceitos de visualização volumétrica usados para visualização das imagens tomográficas de ressonância magnética de frutas e de imagens tomográficas de raios-X do arco coronal.

5

Visualização de Volume

Visualização de volume é um conjunto de técnicas que permitem visualizar a estrutura tridimensional de um objeto a partir de um conjunto de dados de volume [12]. Em aplicações médicas, os dados volumétricos são geralmente adquiridos por tomógrafos que geram uma pilha de imagens. O volume obtido é representado por um *array* tridimensional de voxe*ls*, onde um *voxel* é o menor elemento de dados do volume, correspondente ao conceito de *pixel* na imagem. Em tomografia computadorizada, os *voxels* representam coeficientes de absorção de raios-X, cujos valores correspondem à densidade do material dentro do objeto – como ar, músculo, osso, etc.

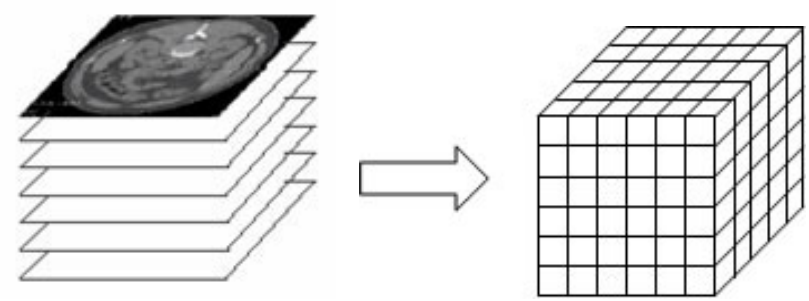


FIGURA 5.1 – Representação do volume.

Os algoritmos de visualização volumétrica podem ser classificados em: algoritmos de *renderização de superfície* e algoritmos de *renderização de volume*. Nos algoritmos de renderização de superfície, os dados volumétricos são convertidos para uma representação geométrica de superfície. Neste caso, métodos tradicionais de renderização de polígonos são usados para visualizar as superfícies do volume. Nos algoritmos de renderização de volume, a visualização é realizada a partir dos dados volumétricos, sem nenhuma representação intermediária. Para visualizar gases e fluidos que não apresentam uma superfície bem definida, a renderização de volume é mais apropriada.

5.1 Renderização de Superfície

Um algoritmo de renderização de superfície, *Marching Cubes* [10], permite extrair isosuperfícies de densidade constante a partir de um conjunto de dados de volume. Este algoritmo se baseia em dois passos básicos: geração de uma malha de polígonos para aproximar a superfície correspondente a um isovalor (*trhesholding*); e cálculo das normais dos polígonos. Após a definição da isosuperfície, métodos tradicionais de renderização de polígonos são utilizados para visualizar o volume. Uma desvantagem dos métodos de renderização de superfícies é que somente a representação geométrica intermediária pode ser visualizada, dificultando a compreensão de detalhes do volume.

(a) *Trhesholding* igual a 80. (b) *Trhesholding* igual a 200.

FIGURA 5.2 – Visualização do mesmo volume de dados com diferentes isosuperfícies.

5.2 Renderização de Volume

Nos algoritmos de renderização de volume [3, 12], os dados são visualizados sem nenhuma representação intermediária. As fases principais dos algoritmos de renderização de volume são: *classificação de dados* e *projeção do volume*. A fase de classificação é um processo que permite selecionar a estrutura do volume a ser visualizada. A fase de projeção do volume envolve o lançamento de raios da posição do observador em direção ao volume e cálculo de composição dos *voxels* interceptados pelo raio, para determinar a cor dos *pixels* no plano de visualização.

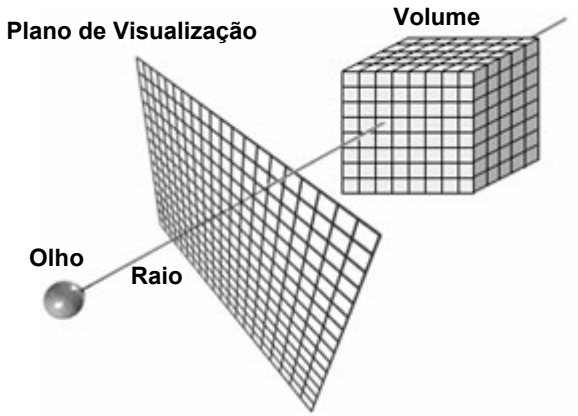


FIGURA 5.3 – Projeção do volume no plano de visualização.

No algoritmo mais simples de renderização de volume, *Maximum Intensity Projection* (MIP), o *voxel* com a maior intensidade ao longo do raio é projetado no plano de visualização. No algoritmo *Ray Casting*, as contribuições de cor e opacidade dos *voxels* interceptados pelo raio são acumuladas para determinar a cor final do *pixel*.

5.2.1 Classificação dos dados

A classificação é um processo interativo que permite selecionar a estrutura tridimensional do volume a ser visualizado. Na fase de classificação é feito um mapeamento dos valores de intensidade dos *voxels* em valores de cor e opacidade. A cor permite diferenciar o material representado pelo *voxel*, enquanto a opacidade permite visualizar estruturas internas do volume. Os valores de opacidade variam

entre 0 e 1, representando *voxels* totalmente transparentes a *voxels* totalmente opacos. A cor e opacidade dos voxels são determinadas através de *funções de transferência*, evitando o armazenamento da informação de cor e opacidade para cada *voxel* do volume.

5.2.1.1 Função de transferência

A função de transferência de opacidade mapeia os valores de intensidade dos *voxels* em valores de opacidade. Uma função de transferência de cor usando o modelo RGB deve ser definida para cada componente de cor *R*, *G* e *B*. A Figura 5.4 mostra exemplos de funções de transferência de cor e opacidade. O eixo horizontal representa os valores possíveis de intensidade do *voxel*. O eixo vertical representa os valores de opacidade, entre 0 e 1, ou os valores de cor (*R*, *G* ou *B*), entre 0 e 255.

Para acelerar o processamento, as funções de transferência são implementadas com tabelas de cor e opacidade. Desta forma, o valor de intensidade do *voxel* é usado como índice das tabelas, para determinar a cor e a opacidade correspondente do *voxel*.

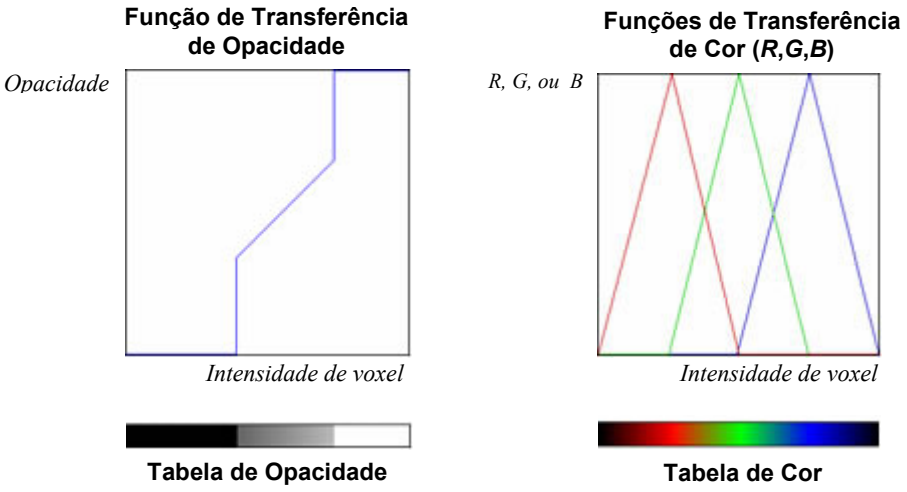


FIGURA 5.4 – Funções de transferência e tabelas de opacidade e cor (R, G, B).

As funções de transferência são específicas para cada aplicação, pois dependem dos valores de intensidade dos *voxels*, do material representado por estes *voxels*, e das estruturas que se deseja visualizar. A Figura 5.5 representa o mesmo

conjunto de dados de volume visualizado com diferentes funções de transferência de opacidade.

FIGURA 5.5 – Visualização de volume com diferentes funções de transferência de opacidade.

5.2.1.2 Histograma do volume

Não existe uma regra geral para o projeto de funções de transferência. No entanto, o *histograma do volume* é uma ferramenta que auxilia a especificação interativa da função de transferência. O histograma do volume representa o número de vezes que ocorrem *voxels* de cada um dos valores possíveis de intensidade. A Figura 5.6 mostra um exemplo de histograma, com os valores de intensidade de *voxels* representados no eixo horizontal e o número de ocorrência, ou freqüência, na vertical. Assim, os valores dos *voxels* que representam a estrutura que se deseja visualizar podem ser determinados, facilitando o projeto da função de transferência.

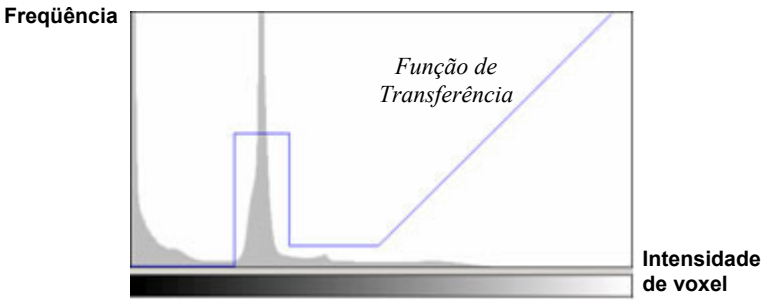


FIGURA 5.6 – Histograma do volume e função de transferência de opacidade.

Após a classificação, o volume pode ser projetado no plano de visualização usando um algoritmo de composição de *voxels*.

5.2.2 Projeção do volume

Nos métodos de renderização de volume, raios são lançados de cada *pixel* da imagem em direção ao volume, e os *voxels* interceptados pelo raio são projetados no plano de visualização.

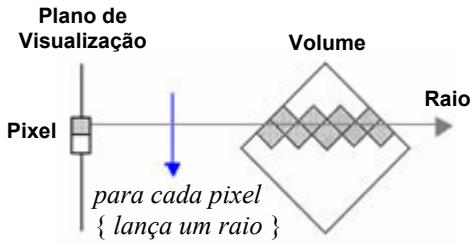


FIGURA 5.7 – Projeção do volume no plano de visualização.

As etapas principais da projeção do volume são: lançamento de raios, determinação dos raios que interceptam o volume, determinação dos *voxels* interceptados pelo raio e composição dos *voxels* ao longo do raio. Os raios que realmente interceptam o volume são lançados e um algoritmo de composição dos *voxels* interceptados pelo raio é usado para determinar a cor final do *pixel*. Vários métodos de renderização foram propostos, com diferentes algoritmos de composição de *voxels*. A seguir, são apresentados dois métodos.

5.2.2.1 Algoritmo Maximum Intensity Projection

Maximum Intensity Projection (MIP) [12,16] é um algoritmo de renderização de volume usado para visualizar estruturas de alta intensidade. Em aplicações médicas, MIP é usado para visualizar a estrutura de vasos sanguíneos (angiografia). Para cada raio lançado através do volume, o *voxel* com maior valor de intensidade é projetado no plano de visualização.

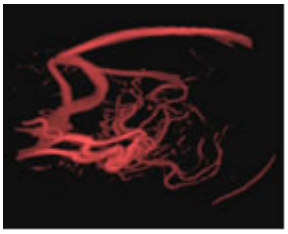


FIGURA 5.8 – Renderização de volume usando MIP.

5.2.2.2 Algoritmo Ray-Casting

O algoritmo de renderização de volume *ray casting* foi originalmente proposto por Levoy [10,11] em 1988. Posteriormente, surgiram várias adaptações deste algoritmo. A Figura 5.9 mostra um volume visualizado com o algoritmo *ray-casting*.

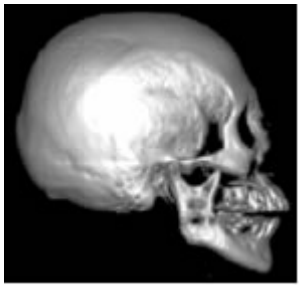


FIGURA 5.9 – Renderização de volume usando ray-casting.

No algoritmo *ray casting*, todas as contribuições de cor C_i e opacidade α_i, de cada voxel *i* interceptado por um raio, são acumuladas para determinar a cor *C* final do *pixel*. A cor C_{out} e a opacidade α_{out} de um raio após atravessar um *voxel i* são calculadas em função da cor C_{in} e da opacidade α_{in}, antes de atravessar o *voxel* e da cor C_i e da opacidade α_i, do próprio *voxel*.

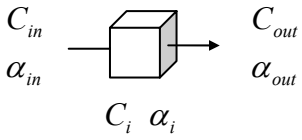


FIGURA 5.10 – Cor e opacidade após atravessar um voxel.

O cálculo de cor e opacidade de um raio após atravessar um *voxel i* pode ser expresso por

$$\hat{C}_{out} = \hat{C}_{in} + \hat{C}_i.[1-\alpha_{in}]$$

$$\alpha_{out} = \alpha_{in} + \alpha_i.[1-\alpha_{in}]$$

onde

$$\hat{C}_{in} = C_{in}.\alpha_{in}$$

$$\hat{C}_{out} = C_{out}.\alpha_{out}$$

$$\hat{C}_i = C_i\alpha_i$$

Depois de calcular todas as N contribuições dos *voxels*, com $k = 1,..., N$, a cor final C do *pixel* é dada por

$$C = \frac{\hat{C}_{out}}{\alpha_{out}}$$

A equação de composição pode ser expressa por

$$\begin{aligned} C = & C(1) + \\ & C(2).[1-\alpha(1)] + \\ & \cdots \\ & C(N).[1-\alpha(1)]. \cdots .[1-\alpha(N-1)] \\ = & \sum_{k=1}^{N} \left(C(k).\prod_{i=1}^{k-1}[1-\alpha(i)] \right) \end{aligned}$$

O custo computacional do algoritmo *ray casting* é alto devido ao tempo de processamento do cálculo de composição de todos os *voxels* do volume. O tempo de processamento cresce linearmente com o aumento do tamanho do conjunto de dados. Além disso, qualquer modificação no posicionamento ou orientação do volume exige que todos os cálculos de projeção do volume sejam novamente processados. Várias otimizações do algoritmo *ray casting* e algoritmos paralelos foram propostos para reduzir o tempo de renderização.

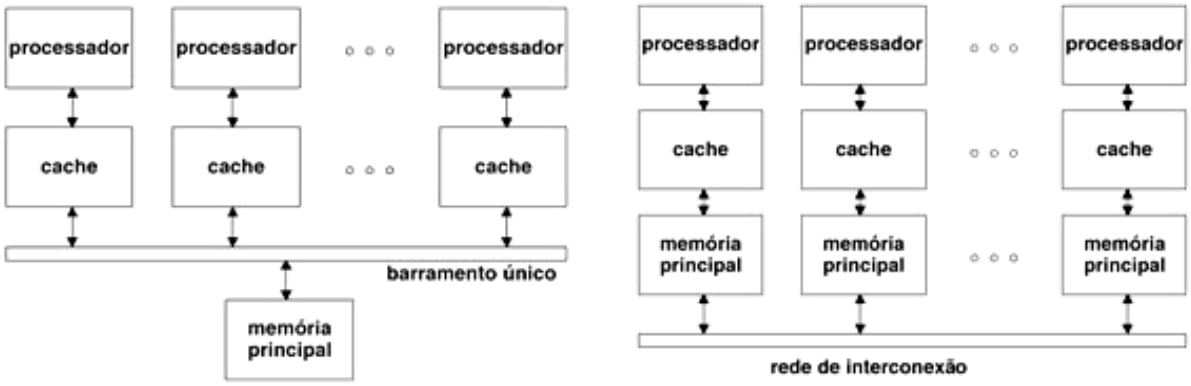
6

Sistemas Paralelos

Neste capítulo, são apresentados os conceitos básicos de sistemas paralelos, além de uma metodologia para desenvolvimento de programas paralelos e análise de desempenho, empregados para desenvolver o programa paralelo de reconstrução 3D de arcos coronais.

6.1 Arquitetura de Computadores Paralelos

Um computador paralelo é composto de vários processadores que operam simultaneamente. Os computadores paralelos podem ser agrupados de acordo com o fluxo de dados em duas categorias básicas: SIMD (*Single Instruction Multiple Data*) e MIMD (*Multiple Instruction Multiple Data*) [4]. Nas máquinas SIMD, todos os processadores executam a mesma instrução em diferentes conjuntos de dados simultaneamente, enquanto nas máquinas MIMD, cada processador executa um programa independente operando sobre diferentes conjuntos de dados. As máquinas MIMD são classificadas de acordo com a organização da memória em máquinas de *memória compartilhada* ou *memória distribuída*. A Figura 6.1 mostra a arquitetura geral dos modelos de máquina MIMD de memória compartilhada e distribuída.



(a) Memória Compartilhada (b) Memória Distribuída

FIGURA 6.1 – Modelos de máquina MIMD de memória compartilhada e memória distribuída.

No modelo de memória compartilhada, todos os processadores têm acesso a uma memória compartilhada global. Por outro lado, no modelo de memória distribuída, cada processador possui uma memória local, podendo acessar diretamente somente os dados armazenados em sua própria memória local. O modelo de máquina MIMD com memória distribuída também é chamado de *multiprocessador*.

6.2 Conceitos de Processamento Paralelo

Em sistemas paralelos, uma aplicação pode ser dividida em diversas tarefas, ou seja, módulos de programa seqüencial que executam uma função ou um conjunto de funções bem definidas. Cada tarefa é uma unidade de execução independente que pode interagir com outras tarefas. Em sistemas com um único processador, um conceito de *multitarefa* é empregado para distribuir o tempo do processador entre várias tarefas concorrentes. A multitarefa usa o conceito de *concorrência* para simular a execução simultânea das tarefas, uma vez que um processador pode executar somente uma tarefa seqüencial de cada vez. Em sistemas multiprocessadores, o conceito de *multiprocessamento* ou *paralelismo* é empregado para executar as tarefas simultaneamente, uma vez que elas são efetivamente executadas em paralelo por mais de um processador. Neste caso, a multitarefa é também usada para execução

concorrente das tarefas que compartilham um determinado processador e os recursos do sistema.

Nos sistemas paralelos de tempo real, o tempo de execução das tarefas deve satisfazer limites rígidos definidos de acordo com os requisitos de um problema específico. Neste caso, um *Sistema Operacional de Tempo Real* (RTOS) é necessário para gerenciar os recursos do sistema, controlar a execução concorrente e simultânea das tarefas e assegurar que as tarefas críticas sejam executadas dentro do limite de tempo. Os SOs modernos normalmente são baseados em uma arquitetura formada por um *kernel* (núcleo) e serviços de gerenciamento dos recursos do sistema, tais como, memória, arquivos e dispositivos de I/O. O *kernel* de tempo real é responsável principalmente pela execução das tarefas e oferece mecanismos de sincronização/comunicação de tarefas através de primitivas de serviço.

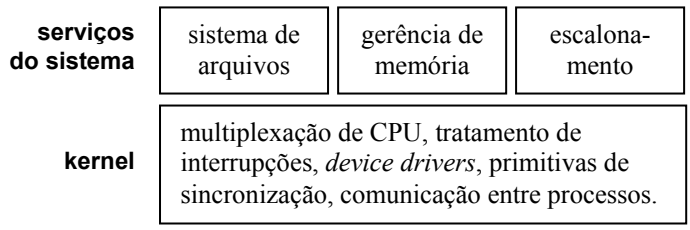


FIGURA 6.2 – Arquitetura de um sistema operacional de tempo real.

Uma tarefa só pode ser executada se os recursos necessários – processador, memória e dispositivos – estiverem disponíveis. Como várias tarefas podem estar concorrendo por um mesmo recurso, é necessário um critério para determinar qual a ordem de execução das tarefas para que cada uma obtenha acesso ao processador. O critério mais utilizado consiste na associação de uma prioridade para cada tarefa indicando a sua importância em relação a outras tarefas no sistema. O escalonador é responsável por decidir qual tarefa deve ser processada pelo processador. Num algoritmo de escalonamento preemptivo baseado em prioridade, a tarefa de prioridade mais alta é selecionada para ser executada no processador. Quando uma tarefa de prioridade mais alta está pronta para ser processada, a tarefa de prioridade mais baixa em execução é suspensa pelo escalonador e o processador é,

então, cedido para a tarefa de prioridade mais alta. As tarefas também são suspensas quando esperam a ocorrência de um evento ou um recurso se tornar disponível. Desta forma, as tarefas podem passar por vários estados durante a execução do programa. Os principais estados de tarefas são:

Estado de Execução: uma tarefa se encontra no estado de execução quando está sendo executada pelo processador. Em sistemas multiprocessadores, várias tarefas podem estar sendo executadas ao mesmo tempo.

Estado de Pronto: indica que a tarefa aguarda uma oportunidade para ser executada pelo processador. O escalonador interrompe uma tarefa em execução, colocando-a no estado de pronto, para que o processador execute outra tarefa de prioridade mais alta no estado de pronto. Uma lista de tarefas no estado de pronto é mantida em ordem de prioridade para que o processador execute primeiro as tarefas mais prioritárias.

Estado Bloqueado: quando uma tarefa necessita de um recurso do sistema que não se encontra disponível, ela deve esperar a liberação do recurso permanecendo no estado bloqueado. Quando o recurso é liberado, a tarefa é colocada no estado de pronto e aguarda o escalonador selecioná-la para entrar em execução.

Estado de Espera: uma tarefa se encontra no estado de espera quando está aguardando um evento para poder prosseguir. Por exemplo, quando uma tarefa em execução começa a realizar uma operação de entrada/saída, ela automaticamente entra no estado de espera liberando o processador para executar outras tarefas. Ao terminar a operação, ela é colocada no estado de pronto e deve aguardar sua vez de ser executada. Assim, um processo em estado de espera aguarda pela conclusão de uma operação em um recurso que já está garantido.

As transições entre os estados de uma tarefa podem ser representadas em um grafo: execução/pronto, pronto/execução, execução/bloqueado, bloqueado/pronto, execução/espera, espera/pronto.

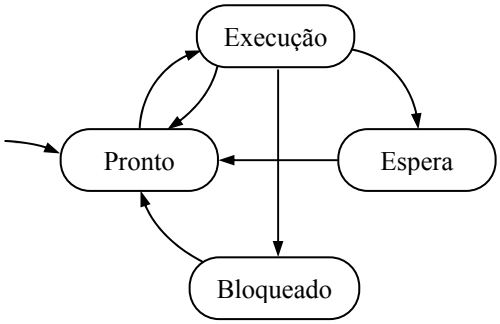


FIGURA 6.3 – Grafo de transições entre estados de tarefas.

6.3 Metodologia de Desenvolvimento de Programas Paralelos

A programação paralela introduz uma complexidade adicional ao modelo de programação seqüencial, devido à necessidade de gerenciamento de recursos compartilhados, sincronização e comunicação entre tarefas [4]. Para garantir uma programação eficiente é necessária uma metodologia de desenvolvimento de programas que permita satisfazer os requisitos de um programa paralelo. Os requisitos fundamentais de projeto de um programa paralelo são: *paralelismo*, *escalabilidade* e *localidade*.

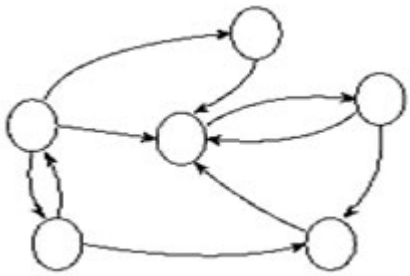
No programa paralelo, a aplicação é dividida em várias tarefas que são executadas simultaneamente. Assim, as possibilidades de execução simultânea devem ser identificadas na fase de projeto do programa paralelo. No entanto, como as tarefas interagem através de mecanismos de comunicação, o programa deve obedecer aos princípios de localidade, isto é, alta taxa de acesso à memória local comparado com o acesso a dados remotos, diminuindo assim os custos de comunicação. Além disso, a programação paralela exige mecanismos de sincronização de tarefas para garantir o acesso aos recursos compartilhados. Outra questão que deve ser considerada na fase de projeto é o desenvolvimento de

programas escaláveis. Desta forma, é possível aumentar o número de processadores do sistema, sem perder eficiência e desempenho na execução do programa paralelo.

Finalmente, os requisitos de *abstração* e *modularidade* são tão importantes no modelo de programação paralela quanto paralelismo, escalabilidade e localidade. Na próxima seção, será apresentado um modelo de programação paralela baseado em tarefas e canais, que simplifica o desenvolvimento de programas modulares e escaláveis. A abstração tarefa/canal é apropriada para desenvolvimento de programas paralelos em sistemas multiprocessadores.

6.3.1 Modelo de programação paralela

No modelo de programação baseado em tarefas e canais, um programa paralelo consiste de uma ou mais tarefas que executam concorrentemente e simultaneamente nos diversos processadores [4]. Além de acessar sua própria memória local, uma tarefa pode interagir com outras tarefas através de serviços de sincronização e comunicação. Neste modelo de programação, a comunicação entre as tarefas é realizada através de canais.

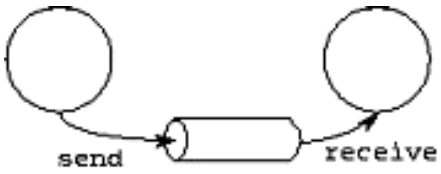


Fonte: FOSTER, 1995.

FIGURA 6.4 – Modelo de programação paralela task/channel. Um programa consiste de um conjunto de tarefas (representadas por círculos) conectadas por canais (setas).

Um canal é uma fila de mensagens na qual uma tarefa pode enviar mensagens para outra tarefa, de forma *síncrona* ou *assíncrona*. Na comunicação síncrona, uma tarefa envia uma mensagem e fica esperando, até que a tarefa receptora receba a mensagem; ou, uma tarefa receptora permanece esperando, até que uma tarefa envie alguma mensagem no canal. A comunicação síncrona introduz

o conceito de *dependência de dados*, ou seja, a computação em uma tarefa requer dados de outra tarefa para continuar a execução.



Fonte: FOSTER, 1995.

FIGURA 6.5 – Comunicação e sincronização de tarefas usando canal: uma tarefa emissora envia dados para uma tarefa receptora, que pode ficar em espera até que uma mensagem chegue pelo canal.

Um mecanismo de sincronização entre tarefas também pode ser implementado através de semáforos. Uma tarefa fica parada esperando um semáforo ser sinalizado, até que outra tarefa sinalize o semáforo. Por outro lado, uma tarefa pode sinalizar um semáforo e ficar esperando, até que outra tarefa verifique o sinal do semáforo.

Algumas propriedades do modelo de programação tarefa/canal são: *desempenho*, *modularidade* e *independência de mapeamento*.

Desempenho: Se duas tarefas que compartilham um canal são mapeadas para processadores diferentes, a conexão do canal é implementada como uma comunicação interprocessador; caso contrário, algum mecanismo mais eficiente de comunicação intraprocessador pode ser usado.

Modularidade: No projeto modular, vários módulos independentes são combinados para obter um programa completo, reduzindo a complexidade do programa e facilitando o reuso de código. Uma tarefa é um modulo independente que encapsula os dados e o código que opera nestes dados. Interações entre módulos são restritos a interfaces bem definidas, isto é, canais, por onde as tarefas enviam e recebem mensagens. Assim, a implementação do módulo pode ser alterada sem modificar outros módulos do programa.

Independência de Mapeamento: Algoritmos devem ser projetados com um número maior de tarefas do que processadores. Esta é uma forma de adquirir escalabilidade: quando o número de processadores aumenta, o número de tarefas por processador é reduzido, mas o algoritmo não é modificado. Uma vez que tarefas interagem usando o mesmo mecanismo de comunicação (canal), programas não dependem de onde as tarefas executam e podem ser implementadas sem levar em consideração o número de processadores.

6.3.2 Projeto de algoritmos paralelos

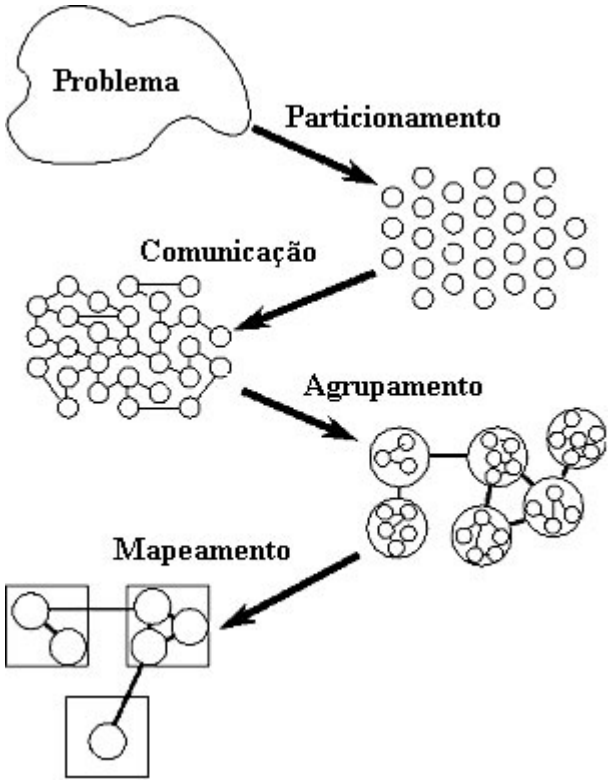
Nesta seção, é mostrado como a especificação de um problema é traduzida num algoritmo que exibe paralelismo, escalabilidade e localidade. Uma metodologia PCAM estrutura o processo de projeto em quatro etapas distintas: *particionamento*, *comunicação*, *agrupamento* e *mapeamento* [4]. Nas duas primeiras etapas, a ênfase é o paralelismo e a escalabilidade. Na terceira e quarta etapas, a atenção é dirigida para a localidade e outros temas relacionados com a eficiência. As quatro etapas são ilustradas na Figura 6.6 e resumidas como segue:

Particionamento: O processamento e os dados são decompostos em pequenas tarefas. Os aspectos práticos são ignorados, como por exemplo, o número de processadores. A atenção é dirigida para a identificação de oportunidades de execução simultânea.

Comunicação: Os requisitos de comunicação necessários para coordenar a execução das tarefas são determinados e as estruturas de comunicação são definidas.

Agrupamento: Várias tarefas pequenas executando em paralelo (grão-fino) aumentam o custo de comunicação. Assim, as estruturas de comunicação e as tarefas, definidas nas duas primeiras etapas, são avaliadas e, se for necessário, as tarefas são combinadas em tarefas maiores (grão-grosso) para diminuir o custo de comunicação e aumentar o desempenho.

Mapeamento: Cada tarefa é atribuída a um processador de forma a tentar satisfazer o objetivo de balanceamento de carga e minimização dos custos de comunicação. O balanceamento possibilita a distribuição da carga de processamento entre os diversos processadores para maximização da utilização dos processadores e aumento do desempenho do sistema.



Fonte: FOSTER, 1995.

FIGURA 6.6 – Metodologia de projeto de programas paralelos PCAM.

A partir da especificação de um problema, a metodologia de desenvolvimento de programas paralelos é empregada com as etapas de particionamento, comunicação, agrupamento e mapeamento, que devem ser constantemente revisadas e aperfeiçoadas.

6.4 Análise de Programas Paralelos

Uma aplicação paralela não pode ter um desempenho pior do que a mesma aplicação executada em um único processador. Assim, o objetivo na programação paralela é otimizar as funções de custo específicas do problema, como tempo de execução, requisitos de memória, custos de comunicação, etc. Na análise do projeto, é necessário considerar não apenas o rendimento, como também a escalabilidade do algoritmo paralelo.

6.4.1 Análise de rendimento

O *rendimento* pode ser usado para comparar a eficiência de diferentes algoritmos, para avaliar a escalabilidade e para identificar congestionamentos e outras ineficiências. As métricas para medir o rendimento podem ser tão diversas, como: o tempo de execução, a eficiência e o *througput* (número de tarefas executadas em um determinado intervalo de tempo). A seguir serão apresentados apenas dois aspectos do rendimento de algoritmos: o tempo de execução e a eficiência.

6.4.1.1 Tempo de execução

Os modelos de rendimento especificam uma métrica, como o tempo de execução T, em função do tamanho do problema N, do número de processadores P, do número de tarefas e outras características.

$$T = f(N, P, ...)$$

O tempo de execução de um programa paralelo é o tempo que decorre desde que o primeiro processador inicia a execução do problema, até que o último processador complete a execução. Durante a execução, cada processador está em computação, em comunicação ou ocioso, como ilustra a Figura 6.7. T_{comp}, T_{comm}, T_{idle} são os tempos de computação, comunicação e ócio, respectivamente.

Tempo de Computação: O tempo de computação de um algoritmo (T_{comp}) é o tempo gasto ao efetuar processamento sem contar a comunicação e ociosidade. O tempo de computação depende, normalmente, do tamanho do problema e das características dos processadores.

Tempo de Comunicação: O tempo de comunicação de um algoritmo (T_{comm}) é o tempo que as tarefas gastam enviando e recebendo mensagens. O tempo de comunicação pode diferir entre comunicações *interprocessador* e *intraprocessador*. Na comunicação interprocessador, as duas tarefas em comunicação estão localizadas em diferentes processadores. Na comunicação intraprocessador, as duas tarefas estão no mesmo processador.

Tempo de Ócio: O tempo de ócio (T_{idle}) é mais difícil de determinar, uma vez que depende da ordem de execução das operações. Um processador pode estar ocioso devido à falta de processamento ou falta de dados para prosseguir o processamento.

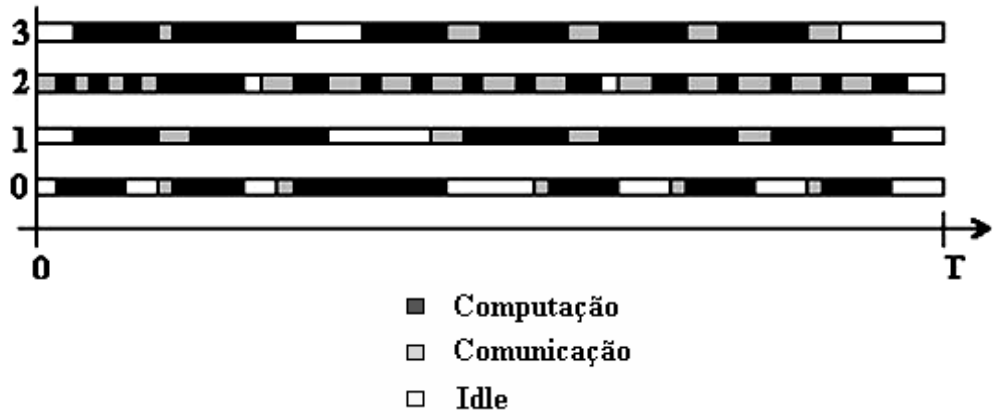


FIGURA 6.7 – Representação de execução de um programa paralelo em quatro processadores. Cada processador gasta o seu tempo em computação, em comunicando ou à espera. T é o tempo total de execução.

O tempo total de execução pode ser definido como a soma dos tempos de computação, de comunicação e de ócio de um processador *j* arbitrário,

$$T = T_{comp}^{j} + T_{comm}^{j} + T_{idle}^{j} \tag{6.1}$$

ou, como a soma desses termos em todos os processadores dividida pelo número de processadores,

$$T = \frac{1}{P}(T_{comp} + T_{comm} + T_{idle})$$

$$= \frac{1}{P}(\sum_{i=0}^{P-1} T_{comp}^{i} + \sum_{i=0}^{P-1} T_{comm}^{i} + \sum_{i=0}^{P-1} T_{idle}^{i}). \quad (6.2)$$

6.4.1.2 Eficiência e Ganho

O tempo de execução nem sempre é a métrica mais conveniente de avaliação do rendimento de um algoritmo paralelo. Como o tempo de execução tende a variar com o tamanho do problema, os tempos de execução têm de ser normalizados para comparação do rendimento do algoritmo para diferentes tamanhos de um problema. A *eficiência* - a fração de tempo que os processadores gastam fazendo trabalho útil – é uma métrica que pode fornecer uma medida mais conveniente de um algoritmo paralelo, independente do tamanho do problema. A eficiência E do programa em P processadores é definida como

$$E_P = \frac{S_P}{P}, \quad (6.3)$$

onde S_P se refere ao ganho (*speedup*) medido em P processadores. O ganho ou *speedup* é o fator de redução do tempo de execução do programa em P processadores, dado por

$$S_P = \frac{T_1}{T_P}, \quad (6.4)$$

onde T_1 é o tempo de execução do programa num sistema com um único processador e T_P é o tempo de execução em P processadores. A Figura 6.8 mostra os gráficos de eficiência e ganho por número de processadores. A eficiência geralmente diminui com o aumento do número de processadores. Um ganho linear e eficiência constante seriam ideais, mas na prática, isto não é possível devido ao aumento de *overhead* e gargalos de comunicação.

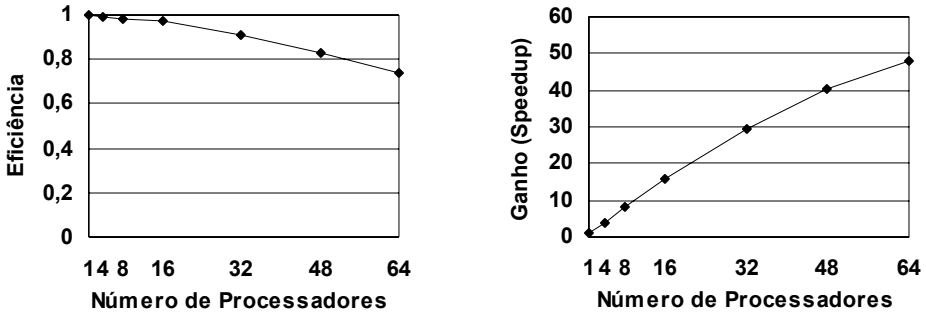


FIGURA 6.8 – Gráficos de eficiência x número de processadores e ganho (speedup) x número de processadores.

6.4.2 Análise de escalabilidade

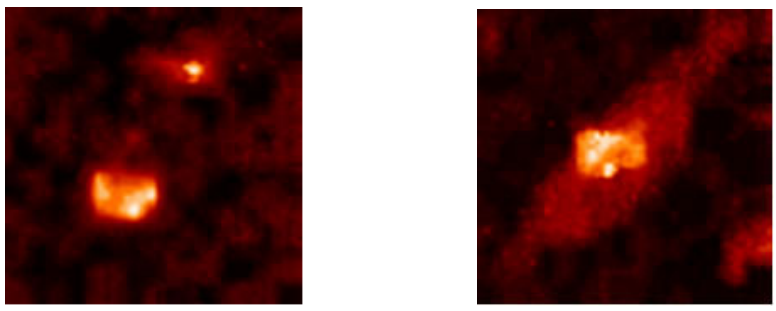
As métricas de rendimento são úteis para análise de escalabilidade de um algoritmo paralelo. A análise de escalabilidade avalia quão efetivamente pode-se usar um número crescente de processadores. Uma forma de quantificar a escalabilidade é determinar como o tempo de execução T e a eficiência E variam com o aumento do número de processadores P, para um problema fixo em tamanho e parâmetros da máquina. A eficiência E, em geral, diminui monotonamente com o aumento do número de processadores. A análise de escalabilidade permite determinar o número máximo de processadores que podem ser usados quando se pretende manter a eficiência em um determinado valor. No entanto, para a análise de escalabilidade é necessário considerar tanto E como T, pois o tempo de execução pode crescer com o aumento do número de processadores. Nestes casos, pode não ser produtivo usar mais que um número máximo de processadores, para um determinado tamanho de problema e parâmetros de máquina.

Todos os conceitos necessários para desenvolvimento do trabalho de mestrado, como interpolação de imagens, metamorfose de imagens, reconstrução e visualização 3D, além de sistemas paralelos, foram abordados nos capítulos anteriores. Nos próximos capítulos serão apresentados o método e o programa paralelo de reconstrução 3D desenvolvido neste trabalho.

7

Reconstrução 3D de Arcos Coronais

O objetivo do trabalho de mestrado é a reconstrução da estrutura 3D de um arco coronal a partir de duas imagens tomográficas de raios-X observadas pelo satélite japonês *Yohkoh* (Figura 7.1). Para reconstruir a estrutura magnética tridimensional é necessário gerar imagens intermediárias entre as imagens da base e topo do *loop*. A forma das seções transversais do *loop* é determinada pela distribuição espacial da energia liberada e pelo transporte desta energia dentro do *loop*. Neste capítulo é apresentado o método de reconstrução 3D baseado em metamorfose de imagens, onde a deformação das imagens da base e topo é controlada por uma curva Bezier* em forma de arco.



(a) Base do *loop*. (b) Topo do *loop*.

FIGURA 7.1 – Imagens de raios-X do loop.

* Ver Apêndice C: Curva Bezier.

7.1 Definição da Estrutura 3D do Loop com Curva Bezier

A estrutura tridimensional do *loop* é aproximada por uma curva Bezier 3D, definida por quatro pontos de controle. Na Figura 7.2, os pontos P_0 e P_3 correspondem aproximadamente aos *footpoints* do *loop*, observados na imagem da base. Os pontos P_1 e P_2 são ajustados para definir a forma do arco magnético.

A função polinomial paramétrica, $P(t)$, de uma curva Bezier cúbica [20] é expressa por

$$P(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t) P_2 + t^3 P_3, \tag{7.1}$$

onde t é o parâmetro normalizado da curva, com valores variando entre 0 e 1; P_0, P_1, P_2 e P_3 são os pontos de controle com coordenadas cartesianas x, y e z.

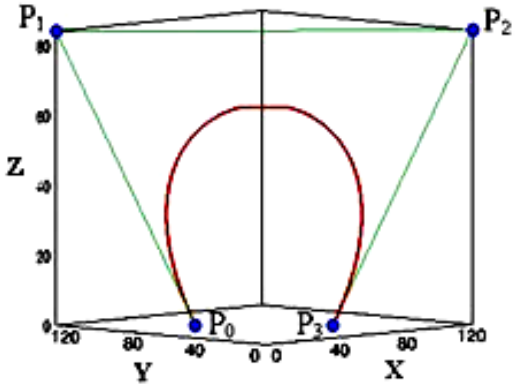


FIGURA 7.2 – Curva Bezier 3D para aproximação da estrutura tridimensional do loop.

A função da curva Bezier pode ser expressa na forma de uma matriz, como

$$P(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix} \tag{7.2}$$

e, a primeira derivada paramétrica, $P'(t)$, é definida por

$$P'(t) = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}. \tag{7.3}$$

Para cada ponto $P(t)$ na curva, o vetor tangente à curva é

$$\vec{P}(t) = \frac{P'(t)}{\|P'(t)\|}. \tag{7.4}$$

Depois da especificação da curva Bezier, são realizadas deformações parciais das imagens da base e topo do *loop*.

7.2 Deformação de Imagens controlada pela Curva Bezier

A deformação de imagens mapeia cada *pixel* de uma imagem em uma nova posição na outra imagem. Um conjunto de pontos de controle é especificado para fazer o mapeamento da posição dos *pixels*. O movimento dos *pixels* vizinhos é controlado por uma função que depende da distância do *pixel* a cada ponto de controle da imagem. Neste trabalho, o método de interpolação ponderada pela distância inversa [26] foi adaptado para deformação de imagens controlada pela curva Bezier.

Definida a estrutura 3D do *loop*, pelos pontos P_0, P_1, P_2 e P_3 com coordenadas (x_0, y_0, z_0), (x_1, y_1, z_1), (x_2, y_2, z_2) e (x_3, y_3, z_3), respectivamente, um par de pontos da curva Pc_0 e Pc_1 deve ser determinado para cada fatia n, para realizar a deformação parcial das imagens da base e topo. A Figura 7.3 mostra os pontos Pc_0 e Pc_1 da fatia n e outros parâmetros descritos nesta seção.

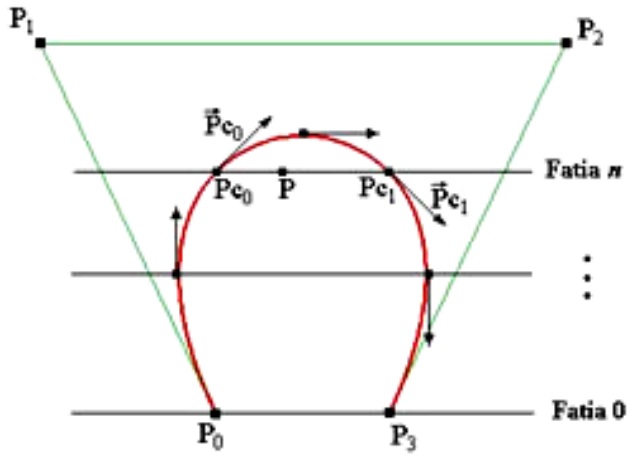


FIGURA 7.3 – Parâmetros para deformação com curva Bezier.

Os pontos da curva, Pc_0 e Pc_1, de uma fatia n têm a mesma coordenada $z = n$. Pela Equação 7.1 ou 7.2, tendo a coordenada z de uma determinada fatia e as coordenadas z_i dos quatro pontos que definem a curva Bezier, as raízes t_0 e t_1 da função polinomial da curva Bezier são determinadas pelo método de Laguerre [18], resolvendo-se a equação

$$z = (1-t)^3 z_0 + 3t\,(1-t)^2 z_1 + 3t^2 (1-t)\, z_2 + t^3 z_3.$$

Agora, para encontrar as coordenadas x e y dos pontos de controle $Pc_0\,(px_0, py_0, z)$ e $Pc_1\,(px_1, py_1, z)$, a função polinomial da curva Bezier deve ser calculada com o parâmetro $t = t_0$ e $t = t_1$. Desta forma,

$$\begin{aligned}
px_0 &= (1-t_0)^3 x_0 + 3t_0 (1-t_0)^2 x_1 + 3t_0^2 (1-t_0)\, x_2 + t_0^3 x_3,\\
py_0 &= (1-t_0)^3 y_0 + 3t_0 (1-t_0)^2 y_1 + 3t_0^2 (1-t_0)\, y_2 + t_0^3 y_3,\\
px_1 &= (1-t_1)^3 x_0 + 3t_1 (1-t_1)^2 x_1 + 3t_1^2 (1-t_1)\, x_2 + t_1^3 x_3,\\
py_1 &= (1-t_1)^3 y_0 + 3t_1 (1-t_1)^2 y_1 + 3t_1^2 (1-t_1)\, y_2 + t_1^3 y_3.
\end{aligned}$$

Para cada ponto de controle Pc_i de uma determinada fatia, um interpolador local $f_i : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ é especificado. O interpolador f_i foi definido com base no vetor tangente à curva Bezier no ponto de controle Pc_i. Assim, os interpoladores f_0 e f_1 nos pontos de controle Pc_0 e Pc_1 são especificados por

$$f_0 = 1 - \vec{\mathrm{Pc}}_0 \quad \text{e} \quad f_1 = 1 - \vec{\mathrm{Pc}}_1$$

onde $\vec{\mathrm{Pc}}_0$ e $\vec{\mathrm{Pc}}_1$ são os vetores tangentes à curva Bezier nos pontos Pc_0 e Pc_1, calculados pela Equação 7.4, como

$$\vec{\mathrm{Pc}}_0 = \frac{\mathrm{Pc}'_0}{\left\|\mathrm{Pc}'_0\right\|} = \frac{(dx_0, dy_0, dz_0)}{\sqrt{dx_0^2 + dy_0^2 + dz_0^2}} = (tx_0, ty_0, tz_0)$$

$$\vec{\mathrm{Pc}}_1 = \frac{\mathrm{Pc}'_1}{\left\|\mathrm{Pc}'_1\right\|} = \frac{(dx_1, dy_1, dz_1)}{\sqrt{dx_1^2 + dy_1^2 + dz_1^2}} = (tx_1, ty_1, tz_1)$$

onde Pc'_0 e Pc'_1 são as primeiras derivadas nos pontos Pc_0 e Pc_1. As derivadas Pc'_0 e Pc'_1 são calculadas usando a Equação 7.3, com $t = t_0$ e $t = t_1$, respectivamente.

Finalmente, os interpoladores locais f_0 e f_1 nos pontos Pc_0 e Pc_1 são determinados por

$$fx_0 = 1 - tx_0 \qquad fx_1 = 1 - tx_1$$

$$fy_0 = 1 - ty_0 \qquad fy_1 = 1 - ty_1$$

Agora uma função de interpolação $f(x, y)$ deve ser encontrada para mapear cada *pixel* da imagem da base (ou topo) em uma nova posição, realizando assim a deformação da imagem da base (ou topo). A função de interpolação ponderada pela distância inversa $f(x, y)$ é uma média ponderada das funções locais f_0 e f_1, com pesos w_0 e w_1 dependendo da distância do *pixel* aos respectivos pontos de controle. A função $f(x, y)$ é calculada por

$$f(x, y) = w_0(x, y) f_0(x, y) + w_1(x, y) f_1(x, y).$$

A função peso $w_i : \mathbb{R}^2 \to \mathbb{R}$ depende da distância inversa de um *pixel* $P(x, y)$ ao ponto de controle Pc_i, normalizado pela soma das distâncias inversas de todos os pontos de controle. Os pesos w_0 e w_1 são calculados por

$$w_0(x, y) = \frac{d_0(x, y)}{d_0(x, y) + d_1(x, y)} \quad \text{e} \quad w_1(x, y) = \frac{d_1(x, y)}{d_0(x, y) + d_1(x, y)},$$

onde $d_0(x, y)$ e $d_1(x, y)$ são as distâncias inversas do *pixel* $P(x, y)$ aos pontos de controle Pc_0 e Pc_1, respectivamente. A distância euclidiana inversa $d_i(x, y)$ de um *pixel* $P(x, y)$ para o ponto de controle $Pc_i(x_i, y_i)$ é

$$d_i(x, y) = \frac{1}{\|P - Pc_i\|} = \frac{1}{\sqrt{(x - x_i)^2 + (y - y_i)^2}}.$$

Para gerar as imagens da base e topo do *loop*, cada *pixel* na posição (x, y) da base (ou topo) é movido para uma nova posição $(x', y') = (x, y) + \alpha(\Delta x, \Delta y)$ na fatia n, sendo α a função de interpolação ponderada pela distância inversa, e $(\Delta x, \Delta y)$ um deslocamento que depende do ponto de controle mais próximo.

O deslocamento $(\Delta x, \Delta y)$ de cada *pixel* da fatia é definido como a distância xy entre o ponto de controle mais próximo na fatia n e o ponto de controle correspondente na imagem da base, para deformação da imagem da base (ou o ponto de controle correspondente na imagem do topo, para deformação da imagem do topo). Assim, para deformação da imagem da base, a posição de cada *pixel* (x, y) mais próximo do ponto Pc_0 é calculada por $(x', y') = (x, y) + (fx_0, fy_0).(\Delta x, \Delta y)$, com $(\Delta x, \Delta y) = P_0 - Pc_0 = (x_0 - px_0, y_0 - py_0)$. O mesmo é feito para os *pixels* mais próximos do ponto Pc_1 que do ponto Pc_0, com $\alpha = (fx_1, fy_1)$ e $(\Delta x, \Delta y) = P_3 - Pc_1 = (x_3 - px_1, y_3 - py_1)$.

Conforme ilustrado na Figura 7.4, o ponto de controle P_0 na imagem da base é mapeado para uma nova posição na fatia n, com um deslocamento Δxy. Os *pixels* que estão mais distantes dos pontos de controle são deslocados com Δxy menor, ponderado pela função de interpolação. O mesmo procedimento é realizado para deformação da imagem do topo.

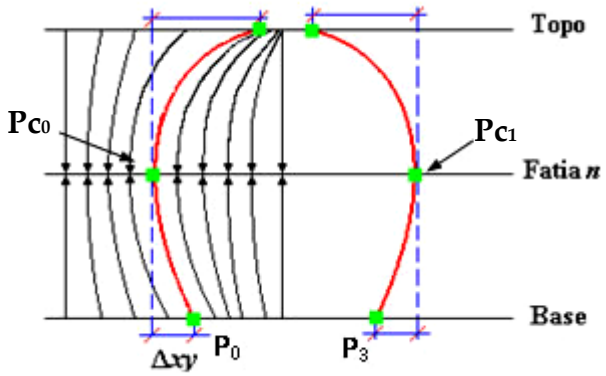


FIGURA 7.4 – Mapeamento de pixels no processo de deformação com curva Bezier.

Agora as imagens intermediárias do *loop* são geradas com *morphing* das imagens da base e topo.

7.3 Reconstrução 3D de Loops usando Metamorfose de Imagens

O processo de metamorfose de imagens para reconstrução 3D do *loop* consiste de um estágio de deformação das imagens da base e topo. A Figura 7.5 mostra algumas fatias deformadas e seus respectivos pontos de controle de deformação. Simultaneamente, um método de interpolação de imagens é usado para determinar o valor do *pixel* da imagem intermediária a partir das imagens deformadas.

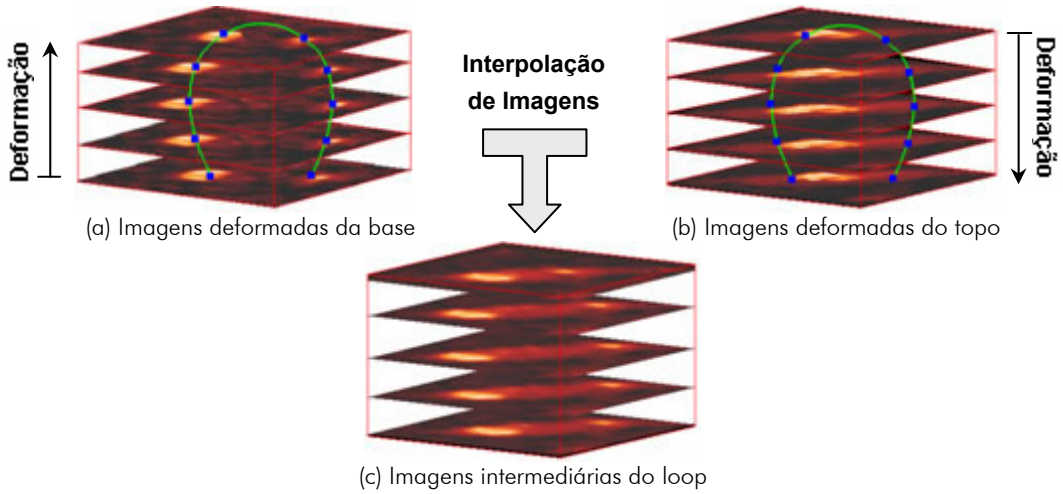


FIGURA 7.5 – Processo de metamorfose de imagens: deformação + interpolação de imagens.

Neste projeto, o método de interpolação *cross-dissolve* e o método de *Interpolação 3D de Imagens usando a Teoria de Estimação Bayesiana* são usados para gerar uma imagem intermediária entre cada par de imagens deformadas da base e topo. A Figura 7.6 mostra o resultado da visualização 3D do *loop* a partir de 100 imagens intermediárias.

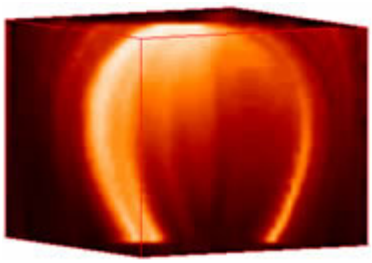


FIGURA 7.6 – Resultado da reconstrução 3D do loop usando morphing de imagens.

No próximo capítulo é apresentado o programa de reconstrução 3D de *loops* implementado neste trabalho de mestrado.

8

Programa de Reconstrução 3D de Arcos Coronais

O projeto *Reconstrução 3D de Imagens Tomográficas de Raios-X de Arcos Coronais em Ambiente Paralelo* foi implementado para ser executado no sistema Atlas, composto atualmente de um *host* PC Pentium com Windows NT e quatro processadores ADSP-21160[†].

Este projeto é constituído de um programa paralelo para deformação de imagens e um programa para interpolação de imagens. Além disso, um programa principal oferece uma interface gráfica de usuário e uma interface para visualização de volume, integrando todas as fases de reconstrução e visualização 3D do arco coronal, de forma transparente para o usuário.

O programa paralelo de deformação de imagens, implementado no *Ambiente Visual de Desenvolvimento de Programas Paralelos de Tempo Real*, é executado nos DSPs da máquina paralela e gerenciado pelo *kernel* Virtuoso.

O método original de *Interpolação 3D de Imagens usando a Teoria de Estimação Bayesiana*, implementado em MATLAB, foi convertido para a linguagem de programação IDL (Interactive Data Language).

[†] Ver a descrição do ambiente de desenvolvimento do projeto no Apêndice A: Sistema Paralelo Atlas com *kernel* Virtuoso, Ambiente Visual de Desenvolvimento de Programas Paralelos de Tempo Real para Virtuoso e ambiente de desenvolvimento IDL.

Para implementar a interface gráfica de usuário foram usados os recursos do ambiente de desenvolvimento IDL.

O programa principal IDL, interpretado e executado no *host* PC da máquina paralela, solicita um serviço enviando uma mensagem para o programa paralelo que responde à solicitação enviando os dados solicitados. Esta interação através de mensagens, baseada no modelo *cliente-servidor*, possibilita a implementação e execução da aplicação e do servidor de paralelismo, independentemente da linguagem de programação usada e do local de execução.

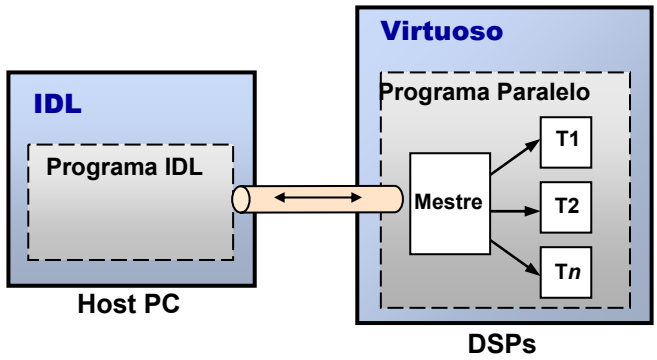


FIGURA 8.1 – Estrutura do programa de reconstrução 3D de loops.

Para permitir a troca de mensagens, o programa paralelo e o programa principal IDL são integrados através de um *pipe* de comunicação bidirecional. O programa IDL envia as imagens originais do topo e base do *loop* para uma tarefa mestre. Esta tarefa distribui as imagens para diversas tarefas, que geram simultaneamente em cada DSP uma seqüência de pares de imagens deformadas da base e do topo. Enquanto as tarefas de deformação geram pares de imagens deformadas, estas imagens são enviadas diretamente para o programa IDL. Então, o programa IDL interpola cada par de imagens deformadas, usando interpolação bayesiana, para gerar as imagens intermediárias do *loop*. Estas imagens são armazenadas num *array* tridimensional de *voxels* para gerar o volume de dados. Finalmente, o usuário pode visualizar as imagens tomográficas reconstruídas e a estrutura 3D do *loop*, usando os recursos de visualização de dados 2D e 3D oferecidos pela linguagem IDL.

8.1 Programa Paralelo de Reconstrução 3D

O processo de reconstrução 3D consiste inicialmente na geração de uma seqüência de pares de imagens deformadas da base e topo do *loop*. Considerando que a deformação da imagem é um processo independente, a reconstrução 3D pode ser facilmente paralelizada.

O programa de deformação pode ser particionado com vários níveis de granularidade, de forma que cada processador calcule a nova posição de um *pixel* da imagem deformada, ou calcule uma parte da imagem deformada, ou então calcule a deformação da imagem inteira (Figura 8.2). Assim, o programa de reconstrução pode ser dividido em várias tarefas independentes que são distribuídas entre os diversos processadores. O grau de paralelismo pode ser aumentado, uma vez que o número de *pixels* ou imagens é muito maior que o número de processadores disponíveis.

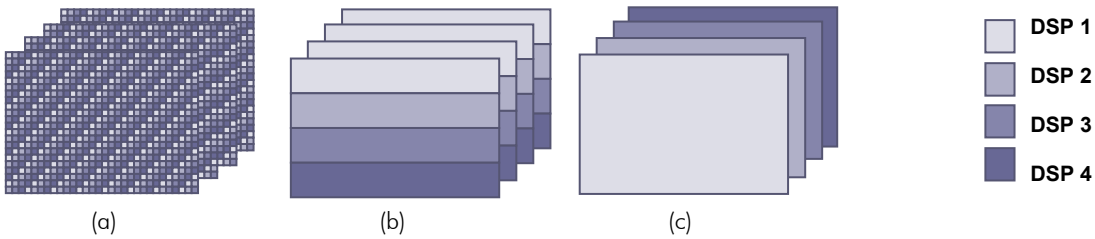


FIGURA 8.2 – Particionamento para deformação das imagens.

No entanto, devido ao aumento do custo de comunicação, as tarefas foram agrupadas em tarefas maiores, onde cada tarefa é responsável por calcular um subconjunto de imagens deformadas a partir das imagens da base e topo do *loop*, conforme ilustrado na Figura 8.3.

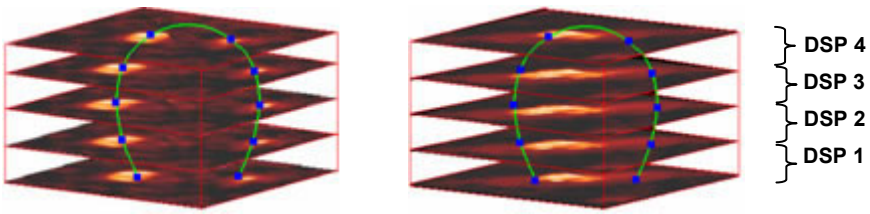


FIGURA 8.3 – Balanceamento de carga para deformação das imagens da base e topo.

O paralelismo foi obtido com *replicação de código*, uma vez que todas as tarefas de deformação executam a mesma atividade. A tarefa mestre determina o número de imagens e quais imagens dentro do conjunto de volume cada tarefa deve calcular. Com o aumento do número de processadores, novas tarefas podem ser facilmente adicionadas, diminuindo o número de imagens processadas em cada processador, até que cada DSP seja responsável por deformar apenas uma imagem, atingindo o nível máximo de escalabilidade.

O programa paralelo de reconstrução 3D, implementado no modelo de programação baseado em tarefas e canais, é dividido em cinco tarefas (MASTER, WARP1, WARP2, ... , WARP4) interconectadas por canais (PIPE, CHANNEL1, ... , CHANNEL4). O canal externo, PIPE, permite a comunicação bidirecional entre o *host* PC e os DSPs da máquina paralela. Desta forma, é possível a troca de mensagens entre o programa IDL e a tarefa MASTER e, entre cada tarefa WARP e o programa IDL. Os outros canais fazem a comunicação interprocessador ou intraprocessador, entre as tarefas MASTER e WARPS.

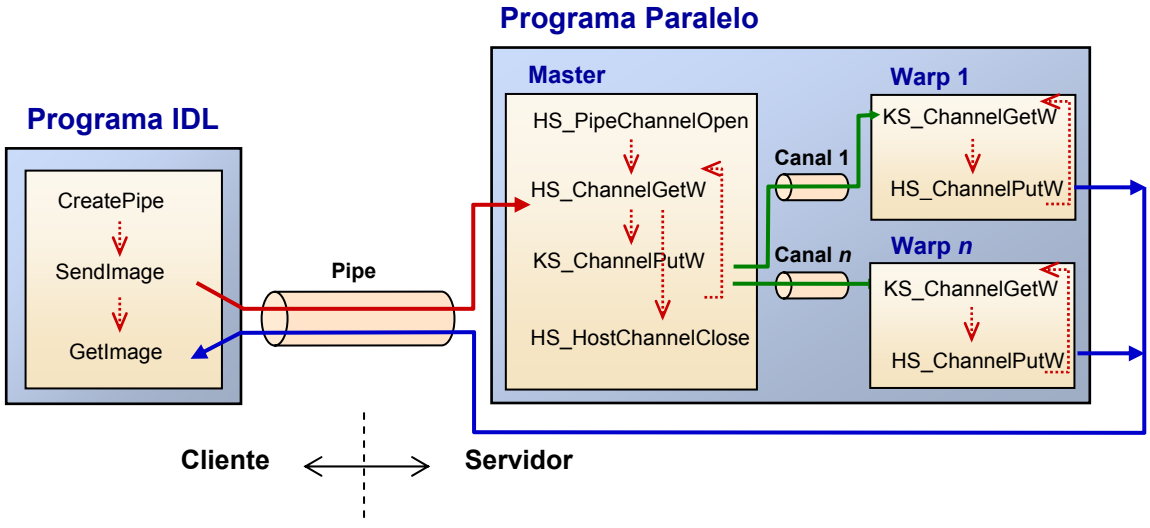


FIGURA 8.4 – Estrutura de comunicação do programa de reconstrução 3D de loops.

A comunicação entre tarefas é realizada através de primitivas de serviço fornecidas pelo *kernel* Virtuoso. As tarefas enviam dados para outras tarefas usando a primitiva de canal *KS_ChannelPutW*() e recebem dados através da primitiva

KS_ChannelGetW(). Desta forma, a comunicação é síncrona, ou seja, ao enviar dados para o canal, a tarefa espera até que o dado enviado seja recebido por outra tarefa. De maneira análoga, uma tarefa receptora espera uma tarefa enviar dados para o canal. A dependência de dados do programa de reconstrução 3D, bem como a comunicação externa entre programas e interna entre tarefas, são ilustradas na Figura 8.4.

Quando o programa IDL é iniciado pelo usuário, uma função IDL *CreatePipe*() cria um *pipe* de comunicação externo. Um programa de gerenciamento de programas paralelos do Virtuoso, *Virtuoso Host Server*, é automaticamente executado para carregar e executar o programa paralelo.

Todas as tarefas são inicialmente executáveis e rodam em *loop* contínuo para responder a várias solicitações do usuário. Inicialmente, a tarefa MASTER abre o *pipe* de comunicação usando a primitiva *HS_PipeChannelOpen*(). Em seguida, a tarefa MASTER executa a primitiva *KS_ChannelGetW*() e aguarda até que o programa IDL envie as imagens originais da base e topo do *loop.*

Quando o usuário solicita uma reconstrução 3D, o programa IDL envia as imagens da base e topo para a tarefa MASTER, que distribui as imagens entre as tarefas WARP usando os canais internos. Cada tarefa WARP aguarda até receber as imagens da base e topo, para gerar um subconjunto de pares de imagens deformadas do *loop*. Durante o processo de geração das imagens deformadas, as tarefas WARP enviam as imagens deformadas para o programa IDL através do *pipe* de comunicação externo.

O programa IDL recebe os pares de imagens deformadas da base e topo de cada tarefa e gera as imagens intermediárias com interpolação de imagens. No final do processo, as imagens intermediárias e o *loop* tridimensional podem ser visualizados. Se desejar, o usuário pode executar uma nova reconstrução 3D, caso contrário, ao encerrar o programa IDL, uma mensagem é enviada ao programa paralelo, que fecha o *pipe* de comunicação externo com a primitiva *HS_HostChannelClose*() e encerra a execução.

8.1.1 Implementação do programa paralelo

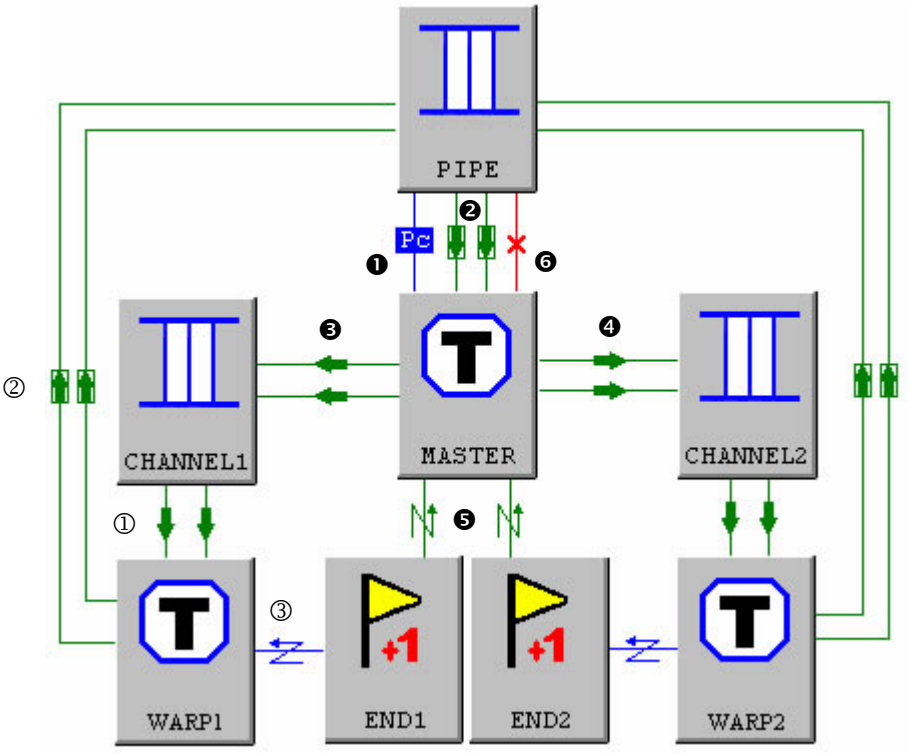
O programa paralelo foi implementado usando o *Ambiente Visual de Desenvolvimento de Programas Paralelos de Tempo Real*.‡ Neste ambiente, os objetos de *microkernel* – tarefas, semáforos, canais, etc. – e os serviços de comunicação e sincronização são representados graficamente, por retângulos e linhas de conexão, respectivamente. O código correspondente das conexões, ou seja, primitivas de serviço oferecidas pelo Virtuoso, são automaticamente geradas pelo ambiente visual. Além disso, códigos complementares específicos das tarefas podem ser escritos em linguagem C/C++, no editor de texto do ambiente. A Figura 8.5 mostra uma representação gráfica simplificada do programa paralelo no ambiente visual e trechos de código das tarefas MASTER e WARP1.§

A representação gráfica permite uma rápida compreensão do programa paralelo. Na Figura 8.5, podemos observar o processamento paralelo e o fluxo de dados, desde a chegada das imagens originais no *pipe* externo, a distribuição das imagens entre as tarefas, o processamento nas tarefas WARP, até o envio das imagens deformadas para o programa externo. A comunicação síncrona entre a tarefa MASTER e WARP1 é feita por pares de primitivas C*hannelPutW*() e *ChannelGetW*() conectadas ao mesmo canal, como por exemplo, nas linhas de código 3 (da tarefa MASTER) e 1 (da tarefa WARP1).

Além dos serviços de comunicação síncrona, é necessário um mecanismo de alocação de memória para armazenar as imagens originais da base, do topo e das duas imagens deformadas. Dois blocos de memória são alocados na tarefa MASTER e quatro blocos em cada tarefa WARP.

‡ Ver Apêndice A.

§ A representação gráfica completa do programa paralelo encontra-se no Apêndice F.



MASTER

```
❶ HS_PipeChannelOpen(PIPE)
  do{
❷   HS_ChannelGetW(PIPE, base_image)
    HS_ChannelGetW(PIPE, top_image)
    ....
❸   KS_ChannelPutW(CHANNEL1, base_image)
    KS_ChannelPutW(CHANNEL1, top_image)
    ....
❹   KS_ChannelPutW(CHANNEL2, base_image)
    KS_ChannelPutW(CHANNEL2, top_image)
    ....
❺   KS_SemaTestW(END1)
    KS_SemaTestW(END2)
  }while()
❻ HS_HostChannelClose(PIPE)
```

WARP 1

```
while(true){
①   KS_ChannelGetW(CHANNEL1, base_image)
    KS_ChannelGetW(CHANNEL1, top_image)
    for(...){      // Gera n imagens deformadas
        ....
②       HS_ChannelPutW(PIPE, warped_base)
        HS_ChannelPutW(PIPE, warped_top)
    }
③   KS_SemaSignal(END1)
}
```

FIGURA 8.5 – Representação gráfica do programa paralelo no Ambiente Visual.

Blocos de memória de tamanho variável são alocados e desalocados dinamicamente na memória externa local do processador onde a tarefa está mapeada, usando os serviços do objeto *Memory Pool* do Virtuoso. Como a memória da máquina paralela é distribuída, uma tarefa não pode acessar diretamente a memória de outra tarefa, nem mesmo se estas tarefas estiverem localizadas no mesmo processador. Assim, as tarefas devem usar mecanismos de comunicação independentes de mapeamento, enviando mensagens por *mailboxes*, filas ou canais.

Um mecanismo de sincronização entre tarefas é necessário para garantir o funcionamento correto do programa paralelo de reconstrução 3D. Conforme ilustrado na Figura 8.5, a tarefa MASTER é sincronizada com as tarefas WARP através de semáforos (END1,..., END4). Quando termina de gerar o subconjunto de imagens deformadas, cada tarefa WARP sinaliza o semáforo com a primitiva *KS_SemaSignal*(), para indicar o término de processamento. A tarefa MASTER verifica o sinal de cada semáforo com a primitiva *KS_SemaTestW*(), aguardando até que todos os semáforos tenham sido sinalizados. A sincronização é necessária, porque uma tarefa WARP poderia enviar imagens para o *pipe* externo, enquanto a tarefa MASTER espera uma imagem original pelo mesmo *pipe*. Desta forma, a imagem deformada seria recebida incorretamente, como imagem original do *loop*. Este problema foi resolvido com a sincronização, uma vez que, somente quando todas as tarefas terminam de usar o *pipe*, a tarefa MASTER pode receber novas imagens originais. Uma alternativa seria criar dois *pipes* externos de comunicação unidirecional.

Depois de definir todos os objetos necessários e escrever o código da aplicação paralela, as tarefas e objetos podem ser facilmente distribuídos entre os processadores da máquina paralela, usando o editor de propriedades de objeto oferecido pelo ambiente de desenvolvimento. As tarefas MASTER e WARP1 foram mapeadas para o DSP1, enquanto as tarefas WARP2, WARP3 e WARP4 foram mapeadas para os DSP2, DSP3 e DSP4, respectivamente. Os blocos de memória devem ser

mapeados obrigatoriamente nos processadores das tarefas correspondentes. Os canais são mapeados de acordo com os *links* de comunicação entre processadores.

No sistema Atlas, os *links* de comunicação devem ser adicionados, dependendo da necessidade de transmissão e recepção de dados. O programador pode especificar se os *links* lógicos são unidirecionais ou bidirecionais, definir o sentido da comunicação para *links* unidirecionais e a velocidade de cada *link*. Na Figura 8.6, se duas tarefas estão localizadas no DSP1 e DSP4, o canal de comunicação pode ser alocado no DSP1, DSP3 ou DSP4. Se as tarefas estão na mesma placa, o canal deve ser mapeado em um dos dois processadores daquela placa. No entanto, o tempo de comunicação será menor se o canal for alocado no mesmo processador da tarefa destino.

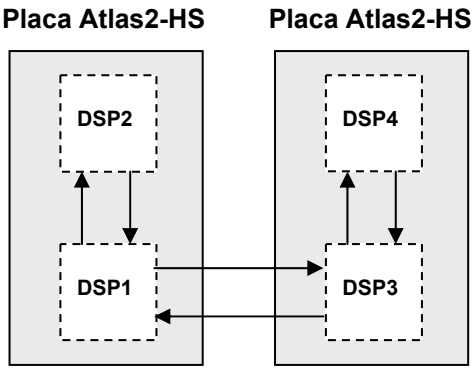


FIGURA 8.6 – Links de comunicação interprocessadores.

A seguir, são apresentados a interface principal de usuário, a interface de execução do programa paralelo de reconstrução e da interpolação de imagens, a interface de visualização e o mecanismo de integração entre o programa IDL e o programa paralelo.

8.2 Programa Principal

O programa principal permite que o usuário selecione as imagens originais da base e topo que devem ser enviadas para o programa paralelo, e execute todas as operações necessárias para reconstrução 3D, como especificação da curva Bezier, execução do programa paralelo e visualização de dados. A Figura 8.7 mostra a janela principal da interface gráfica de usuário.

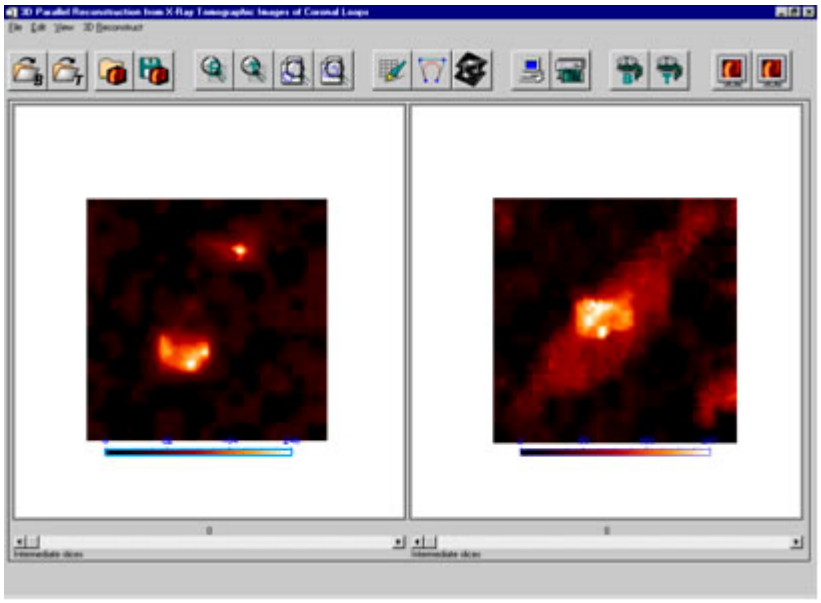


FIGURA 8.7 – Interface gráfica principal.

8.2.1 Especificação e visualização da curva Bezier 3D

O primeiro passo para reconstrução 3D é a especificação das coordenadas de quatro pontos de controle da curva Bezier. A curva Bezier 3D pode ser visualizada numa interface gráfica.

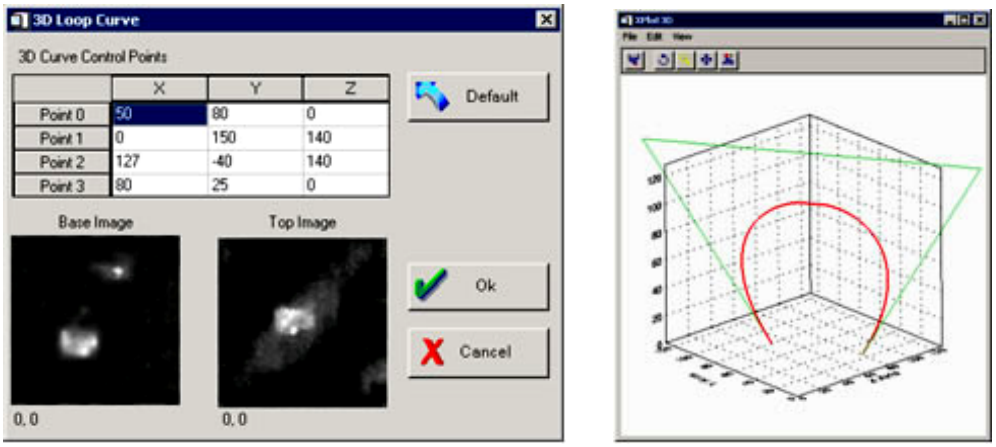


FIGURA 8.8 – Interface de especificação e visualização da curva Bezier.

8.2.2 Interface de execução do programa paralelo

O programa paralelo de reconstrução 3D é executado pelo usuário através do menu do programa principal. As imagens originais da base e topo são enviadas ao programa paralelo, que retornam as imagens deformadas. Uma barra de progressão de cada tarefa, atualizada sempre que um par de imagens deformadas é recebido, mostra o tempo de execução de cada tarefa nos quatro processadores. A partir das imagens deformadas, as imagens intermediárias são geradas no programa IDL e visualizadas na interface do usuário.

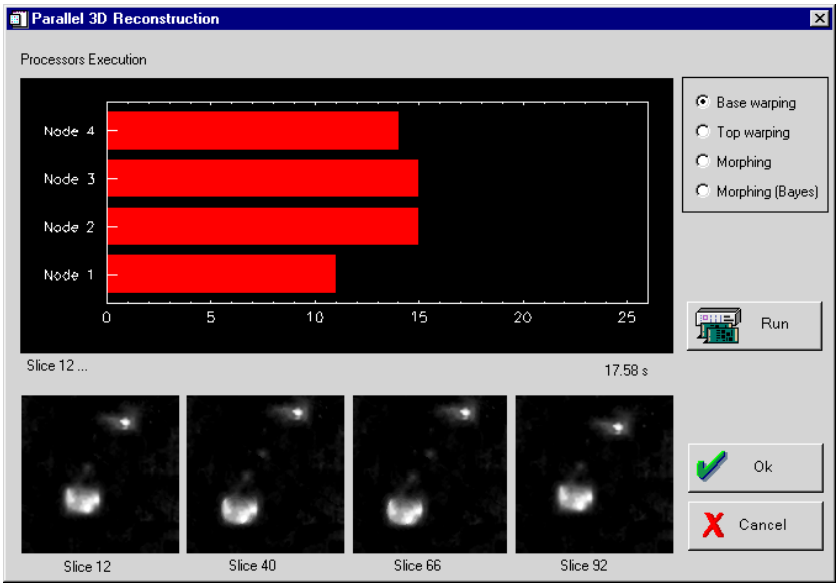


FIGURA 8.9 – Interface de execução do programa paralelo de reconstrução 3D.

Como as imagens intermediárias são geradas em ordem aleatória entre os subconjuntos de fatias do volume, é necessário que cada tarefa envie as imagens deformadas com um identificador de tarefa e número de seqüência da imagem. Desta forma, o *array* tridimensional de *voxels* pode ser armazenado corretamente.

8.2.3 Interface de visualização 3D de loops

A linguagem IDL oferece uma interface gráfica para visualização e manipulação interativa de volumes e iso-superfícies. O *loop* tridimensional pode ser visualizado com o método MIP, *ray-casting* ou iso-superfície. A Figura 8.10 mostra a interface gráfica de visualização 3D.

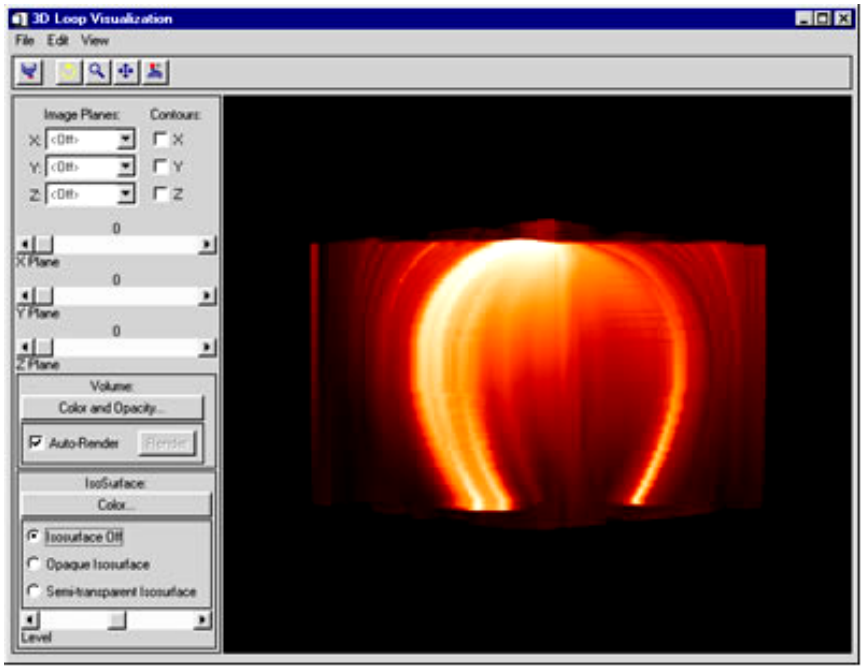


FIGURA 8.10 – Interface de visualização 3D de loops.

8.2.4 Integração de programas entre IDL&Virtuoso

A linguagem IDL permite que funções e procedimentos escritos em linguagem C/C++ sejam adicionados ao conjunto de rotinas da linguagem IDL, através de bibliotecas compartilhadas [7]. Desta forma, uma DLL com as funções *CreatePipe*(), *SendImage*() e *GetImage*() foi gerada para fazer a comunicação com o programa Virtuoso.

As funções *SendImage*() e *GetImage*() usam funções *ReadFile*() e *WriteFile*() de C++, para leitura e escrita em arquivos e *pipes*. Estas funções são compatíveis com as primitivas *HS_ChannelGet*() e *HS_ChannelPut*() do Virtuoso.

Quando o ambiente de desenvolvimento IDL é iniciado, estas rotinas são automaticamente adicionadas ao conjunto de rotinas da linguagem IDL, permitindo que o programador use estas funções sem se preocupar com detalhes de implementação.

No próximo capítulo são apresentados os resultados obtidos com o método de reconstrução 3D de *loops*, e uma análise do programa paralelo de reconstrução.

9

Resultados

Nesta seção são apresentados os resultados da reconstrução 3D de *loops* e a análise de desempenho do programa paralelo de deformação de imagens.

9.1 Análise do Programa Paralelo

Para fazer a análise do programa paralelo de deformação de imagens, o programa foi executado em uma máquina paralela com quatro processadores, gerando um número fixo de 103 imagens deformadas da base e topo.

Os tempos de execução foram medidos executando-se o programa paralelo em um número crescente de processadores. O ganho (*speedup*) e a eficiência são determinados por

$$S_P = \frac{T_1}{T_P} \qquad \text{e} \qquad E_P = \frac{S_P}{P},$$

onde T_1 é o tempo de execução em um processador e T_p, em P processadores.

A Tabela 1 mostra o tempo de execução, a eficiência e o *speedup* do programa paralelo por número de processadores. A Figura 9.1 mostra os gráficos de tempo de execução, eficiência e *speedup* por número de processadores.

TABELA 1 – Tempo de execução (em segundos), eficiência, e speedup do programa paralelo.

Número de Processadores	Tempo (s)	Eficiência (%)	Speedup
1	75,86	100,00	1,0
2	38,36	98,88	1,98
3	25,63	98,67	2,96
4	19,28	98,37	3,93

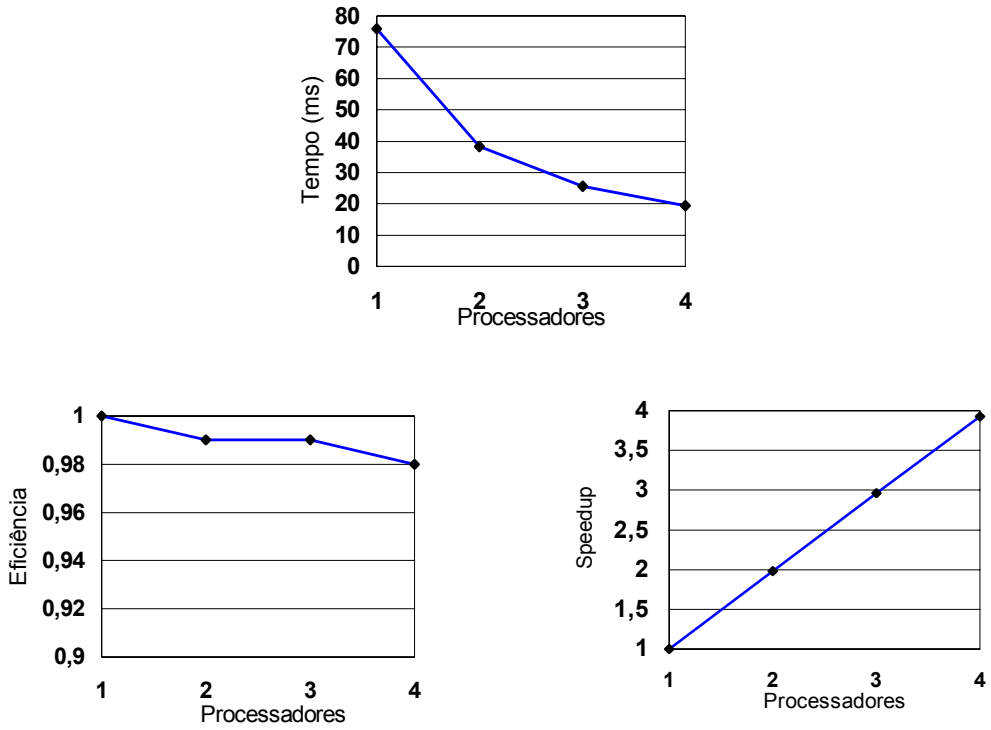


FIGURA 9.1 – Gráficos de tempo de execução, eficiência e speedup por número de processadores.

A eficiência se mantém praticamente constante e o ganho cresce linearmente com o aumento do número de processadores, o que significa que o algoritmo é escalável. Para verificar se o programa paralelo é escalável para números maiores de processadores, seria necessário estimar o tempo de execução para determinar a eficiência e *speedup*. Com isso, poderíamos determinar o número máximo de processadores para manter a eficiência acima de um determinado valor.

9.2 Análise da Reconstrução 3D do Loop

O *loop* foi reconstruído com um número total de 103 fatias de 128x128 *pixels.* O tamanho do *loop* é determinado pela altura da curva Bezier mostrada na Figura 9.2.

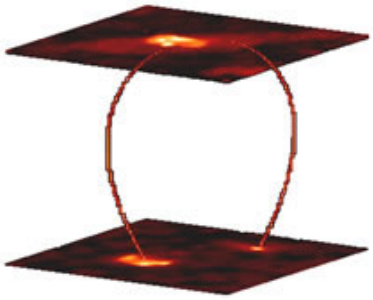
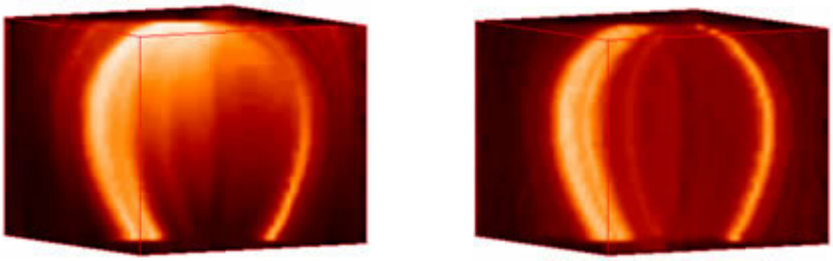


FIGURA 9.2 – Estrutura do loop.

A Figura 9.3 mostra uma comparação da reconstrução da estrutura tridimensional do *loop*, usando os métodos *cross-dissolve* e interpolação bayesiana para gerar as imagens intermediárias no processo de metamorfose de imagens.



(a) Com *cross-dissolve* (b) Com interpolação bayesiana

FIGURA 9.3 – Loop 3D reconstruído com 103 imagens intermediárias de 128 x 128 pixels.

Uma vantagem do método de reconstrução 3D é a capacidade de obter *loops* de várias formas e tamanhos, bastando apenas definir quatro pontos de controle para a curva Bezier. Desta forma, um *loop* torcido com uma forma sigmoidal pode ser facilmente reconstruído a partir das imagens da base e topo. Quando as imagens de raios-X mostram *loops* torcidos, com aparência de S ou S invertido, é muito provável ocorrer uma explosão solar em poucos dias. A Figura 9.4 mostra um *loop* sigmoidal capturado pelo satélite *Yohkoh* em 7 de junho de 1998.

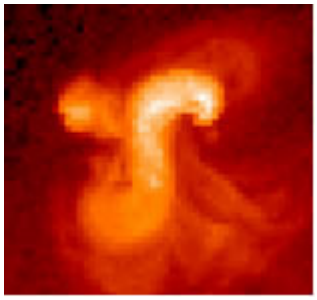


FIGURA 9.4 – Um loop sigmoidal capturado pelo Yohkoh.

A Figura 9.5 mostra uma curva Bezier que define a forma sigmoidal do *loop*. O resultado da reconstrução do *loop* sigmoidal a partir das imagens tomográficas de raios-X da base e topo do *loop* é mostrado na Figura 9.6.

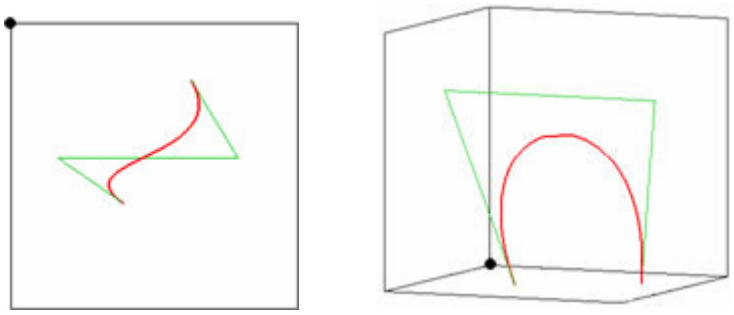


FIGURA 9.5 – Curva Bezier com forma sigmoidal.

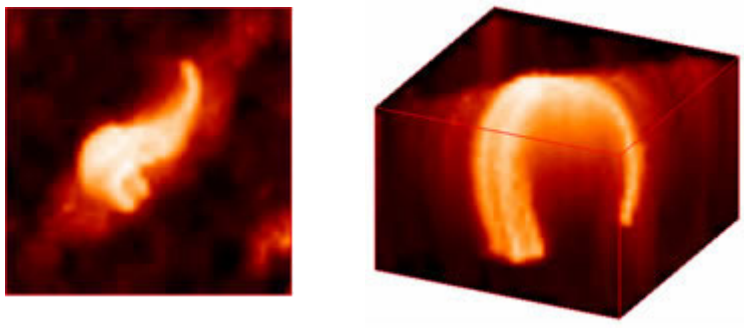


FIGURA 9.6 – Loop reconstruído com forma sigmoidal.

Desta forma, apesar do número insuficiente de imagens tomográficas para reconstrução 3D do *loop*, o método de reconstrução desenvolvido neste trabalho apresenta um resultado visual satisfatório, permitindo a reconstrução de *loops* de diferentes formas e tamanhos.

10

Conclusões

Imagens tomográficas de raios-X da coroa solar permitem observar os arcos coronais em duas profundidades da atmosfera solar. Devido à necessidade de extrair informações a partir da estrutura tridimensional dos arcos coronais para previsão de explosões solares, métodos de reconstrução 3D vêm sendo desenvolvidos.

As seções transversais do arco coronal mudam de forma devido à distribuição espacial e transporte de plasma magnético em tubos com formato de arco. Desta forma, um método de reconstrução 3D foi desenvolvido para gerar imagens intermediárias entre as imagens tomográficas da base e topo do arco coronal, usando técnicas de metamorfose de imagens. A reconstrução 3D do arco coronal é realizada através de um processo de metamorfose que consiste de deformação e interpolação de imagens para transformar gradualmente a imagem da base na imagem do topo. A deformação progressiva das imagens da base e topo é controlada por uma curva Bezier em forma de arco, para que as duas imagens tenham a mesma forma. Um método de *Interpolação 3D de Imagens usando a Teoria de Estimação Bayesiana* é usado para gerar as imagens intermediárias entre cada par de imagens deformadas. Este método de reconstrução 3D apresentou bons resultados, permitindo a reconstrução do arco com várias formas e tamanhos. Com isso, os cientistas espaciais poderiam definir a estrutura 3D mais adequada para cada arco coronal.

Devido à necessidade de obter a reconstrução 3D em tempo razoável para estimar os parâmetros envolvidos na previsão de explosões solares, um programa paralelo de reconstrução 3D foi implementado para executar no sistema Atlas com quatro processadores. Atualmente, apenas o processo de deformação das imagens da base e topo é executado em paralelo. Análises preliminares mostram que o algoritmo é escalável, o que nos permite expandir a capacidade de processamento do sistema, aumentando o número de processadores, sem a necessidade de grandes modificações no projeto. Para determinar o número máximo de processadores, uma análise mais detalhada do algoritmo deveria ser realizada com uma estimativa do tempo de execução para números maiores de processadores, de acordo com os tempos de computação, comunicação e *idle* do algoritmo paralelo.

Como os arcos coronais são constituídos de gás aquecido a altas temperaturas, sem uma superfície bem definida, técnicas de renderização de volume, que são mais apropriadas para visualização de fluidos, foram usadas para visualizar os arcos coronais após a reconstrução. O método *ray casting* possibilita a visualização de estruturas de densidades diferentes dentro do volume, através de funções de transferência de opacidade, tendo apresentado bons resultados para visualização das imagens de ressonância magnéticas de frutas. No caso dos arcos coronais, o resultado visual não foi satisfatório, uma vez que definir uma função de opacidade é uma tarefa muito mais complexa. A técnica *maximum intensity projection* apresentou um resultado visual mais realístico sem a necessidade de definir funções de transferência. No entanto, as técnicas de renderização de volume apresentam um alto custo computacional, dependendo do tamanho do volume.

Como proposta de trabalho futuro, podemos destacar a paralelização dos métodos de interpolação de imagens e visualização de volume, para reconstrução e visualização em tempo real de uma série de imagens tomográficas em evolução no tempo.

Referências Bibliográficas

[1] ATLAS System: *user guide*, Atlas2HS-V1.1. Eonic Solutions GmbH, 2001.

[2] BEIER, T., NEELY, S. Feature-based image metamorphosis. *Computer Graphics*, *Proceedings* SIGGRAPH, 1992. v. 26, n. 2, p. 35-42.

[3] DREBIN, R. A., CARPENTER, L., HANRAHAN, P. Volume rendering. *Computer Graphics, Proceedings of SIGGRAPH*, 1988. v.22, n.4, p.65-74.

[4] FOSTER, I. *Designing and building parallel programs*. Addison Wesley, 1995. Disponível em http://www.mcs.anl.gov/dbpp.

[5] GOMES, J. et al. *Warping and morphing of graphical objects*. Morgan Kaufmann Publishers Inc., São Francisco, CA, 1998.

[6] IDL-Building IDL applications, version 5.4. Research Systems, Inc., 2000.

[7] IDL-External development guide, version 5.4. Research Systems, Inc., 2000.

[8] LEE, S. et al. Image metamorphosis with scattered feature constraints. *IEEE Trans. Visualization and Computer Graphics*. v. 2, n. 4, p. 337-354, 1996.

[9] LEE, S. et al. Image morphing using deformation techniques. *The Journal of Visualization and Computer Animation*. v. 7, n. 1, p. 3-23, 1996.

[10] LEVOY, M. Display of surfaces from volume data. *IEEE Computer Graphics and Applications,* v.8, n.3, p.29-37, 1988.

[11] LEVOY, M. Efficient ray tracing of volume data. *Computer Graphics. Proceedings of SIGGRAPH*. p. 157-67, 1990.

[12] LICHTENBELT, B., CRANE, R., NAQVI, S. *Introduction to volume rendering*. Hewlett-Packard Professional Books. Prentice Hall PTR, 1998.

[13] LUO, B., HANCOCK, E. R. Slice interpolation using the distance transform and morphing. *Proceedings IEEE 13th International Conference on Digital Signal Processing*, p. 1145-1149, 1997.

[14] MASCARENHAS, N. D. A. et al. An estimation theoretic approach to 3D image interpolation. In: 13TH BRAZILIAN SYMPOSIUM ON COMPUTER GRAPHICS AND IMAGE PROCESSING, 2000, Gramado, RS. *Proceedings SIBGRAPI*. p.163-9.

[15] MROZ, L., GRÖLLER, E., KÖNIG, A. Real-time maximum intensity projection, *Proc. Joint Eurographics - IEEE TCCG Symposium on Visualization*, VISSYM'99, p.135-144, 1999.

[16] MROZ, L., HAUSER, H., GRÖLLER, E., Interactive high-quality maximum intensity projection, *Proc. Eurographics* 2000, v.19, n.3, p.C-341-350, 2000.

[17] PAPOULIS, A. *Probability, random variables and stochastic processes.* McGraw Hill, 1965.

[18] PRESS, W.H. et al. "Roots of polynomials", *Numerical recipes in C: the art of scientific computing.* 2.ed., Cambridge University Press, p.371-374, 1992.

[19] RIBEIRO, J. R. P., SILVA, N. C., MORON, C. E. A visual environment for the development of parallel real-time programs. In: *Lecture Notes in Computer Science*, 1998. v.1388, p.994-1014. http://www.dc.ufscar.br/~tev.

[20] ROGERS, D.F. e ADAMS, J.A. "Space curves", *Mathematical elements for computer graphics.* McGraw-Hill, 2.ed., 1990, p.289-305.

[21] ROSA, R. R., SHARMA, A. S. and VALDIVIA, J. A. Characterization of Asymmetric Fragmentation Patterns in Spatially Extended Systems. *Int'l J. Modern Physics C*, v.10, n. 1, 1999, p. 147-163.

[22] ROSA, R. R. et al. Emission measurement density curves for regularization of coronal loop tomography. *Proceedings of HESSE*, NASA, 1999.

[23] ROSA, R. R et al. Normalized emission measurement density curves for reconstruction of X-ray loop tomography. *ASP Conference Series*, v.206, p.293-296, 2000.

[24] ROSA, R. R. et al. Phenomenological dynamics of coronal loops using a neural network approach. *Advances in Space Research*, v.25, n.9, p.1917-21, 2000.

[25] RUPRECHT, D., MÜLLER, H. Deformed cross-dissolves for image interpolation in scientific visualization. *The Journal of Visualization and Computer Animation*. v. 5, n. 3, p.167-181, 1994

[26] RUPRECHT, D., MÜLLER, H. Image warping with scattered data interpolation, *IEEE Computer Graphics and Applications*, v.15, n.2, p.37-43, 1995.

[27] SHEPARD, D. A two dimensional interpolation function for irregularly spaced data. In Proc. 23 Nat. Conf. ACM, p. 517-524, 1968.

[28] STARK, H., WOODS, J. W. *Probability, random processes, and estimation theory for engineers.* 2.ed. Prentice-Hall, 1994.

[29] TSUNETA, S. and LEMEN, J. R. Dynamics of the solar coronal observed with the Yohkoh soft X-ray telescope. *Physics of Solar and Stellar Coronae*, 1993, p. 113-130.

[30] UCHIDA, Y. New Aspects of Solar Coronal Physics Revealed by Yohkoh. *Physics of Solar and Stellar Coronae*, 1993, p. 97-112.

[31] VAN TREES, H. L. *Detection, estimation and modulation theory,* Party I. Wiley, 1968.

[32] VIRTUOSO user guide for version 4.2, Wind River Systems, Inc. 2001.

[33] WATT, A. *3-D Computer graphics.* 2.ed. Addison-Wesley, 1993. p.313-329.

[34] WATT, A., WATT, M. *Advanced animation and rendering techniques.* Addison-Wesley, 1992. p.297-321.

[35] WOLBERG, G., Image morphing: a survey, *The Visual Computer Journal,* 1998 v.14, p.360-372.

[36] WOLBERG, G., Digital Image Warping, *IEEE Computer Society Press*, 1990.

Apêndices

Parte I

Ambiente de Desenvolvimento

Os recursos computacionais de hardware e software usados para implementação do projeto de mestrado e execução do programa de reconstrução 3D de arcos coronais são apresentados neste capítulo.

A.1 Sistema Paralelo Atlas™

O sistema paralelo Atlas™ [1] da Eonic Solutions GmbH inclui hardware e software para implementar e executar aplicações que necessitam de alto desempenho. Este sistema tem, atualmente, um *host* PC Pentium rodando Windows NT e duas placas Atlas2-HS. Cada placa possui dois processadores ADSP-21160 (Hammerhead SHARC ™) da Analog Devices, 72 MHz. Para cada DSP há 2 MB de memória SBSRAM.

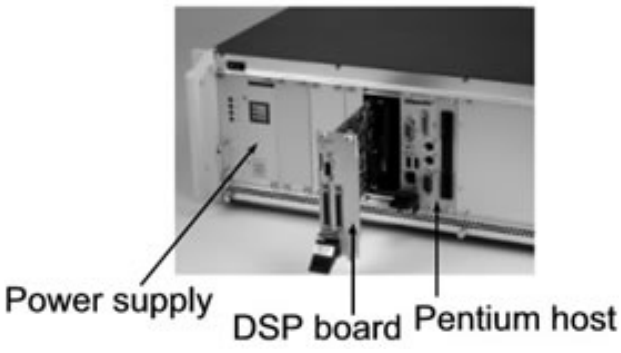


FIGURA A.1 – Sistema paralelo Atlas.

A.2 Virtuoso™ – Sistema Operacional de Tempo Real

O sistema operacional de tempo real Virtuoso™ da Wind River Systems gerencia a execução do programa paralelo nos processadores do sistema Atlas. O Virtuoso™ (*Virtual Single Processor Programming System*) [32] é um *kernel* paralelo que oferece suporte ao desenvolvimento de aplicações de tempo real. As aplicações desenvolvidas no Virtuoso são divididas em tarefas que podem interagir com outras tarefas através de serviços de comunicação e sincronização. Nos sistemas multiprocessadores, as tarefas podem ser distribuídas entre os diversos processadores. Em cada processador, o Virtuoso usa o conceito de multitarefa para realizar a execução concorrente das tarefas.

O Virtuoso concentra apenas os serviços e objetos de *microkernel* necessários para o desenvolvimento de aplicações paralelas. Os principais objetos de *microkernel* – tarefas, semáforos, eventos, *mailboxes*, filas de mensagens, canais, recursos e memória – são apresentados a seguir. Cada objeto possui atributos específicos e suporta um conjunto de serviços.

Tarefa: Tarefas são módulos independentes de programa, que interagem com outras tarefas através dos serviços de sincronização e/ou comunicação. Os principais serviços de tarefas são:

KS_TaskStart	Inicia o processamento da tarefa
KS_TaskSuspend	Suspende o processamento da tarefa
KS_TaskResume	Reinicia o processamento da tarefa
KS_TaskSleep	Atrasa o processamento da tarefa

Semáforo: Semáforos são usados para sincronizar duas tarefas. Uma tarefa sinaliza um semáforo, enquanto outra tarefa espera o semáforo ser sinalizado. Se o semáforo não estiver sinalizado, a tarefa pode retornar, esperar um sinal (W) ou esperar o sinal com *timeout* (WT).

KS_SemaSignal	Sinaliza o semáforo, incrementando o contador
KS_SemaTest[W|WT]	Testa o semáforo, decrementando o contador

Evento: Este objeto é similar a um semáforo, mas tem apenas dois estados (0 ou 1).

KS_EventSignal	Sinaliza um evento
KS_EventTest[W]	Testa um sinal do evento

Mailbox: Tarefas enviam e recebem mensagens de tamanho arbitrário através de *mailboxes*. Uma prioridade pode ser associada às mensagens.

KS_MsgGet[W|WT]	Lê uma mensagem do mailbox
KS_MsgPut[W|WT]	Envia uma mensagem para mailbox

Fila: Uma fila de mensagens é usada para transferir pequenas quantidades de dados de tamanho fixo, usando *buffers*. Nenhuma sincronização entre tarefas é necessária.

KS_FIFOGet[W|WT]	Obtém uma entrada da fila
KS_FIFOPut[W|WT]	Coloca uma entrada na fila

Canal: Canais são *pipes* que permitem uma tarefa enviar e receber dados de tamanho arbitrário. Além disso, os canais permitem a comunicação entre uma tarefa e um programa externo executado no host PC.

KS_ChannelGet[W|WT]	Lê uma mensagem do canal
KS_ChannelPut[W|WT]	Envia uma mensagem para canal

Recurso: Recurso é um dispositivo lógico usado para controlar o acesso a um recurso físico compartilhado (memória, dispositivos de I/O, etc). Uma tarefa bloqueia um recurso para impedir que outra tarefa de prioridade mais alta use o recurso antes dela terminar a operação. Uma tarefa deve desbloquear o recurso após a utilização do mesmo.

KS_ResLock[W|WT]	Bloqueia um recurso
KS_ResUnlock	Desbloqueia o recurso

Memória: Memória é um recurso pelo qual as tarefas concorrem para armazenar dados. Um ou mais blocos de memória de tamanho fixo podem ser alocados pelas tarefas.

KS_PoolGetBlock[W|WT]	Aloca um bloco de memória
KS_PoolFreeBlock	Desaloca um bloco de memória

A.3 Ambiente Visual de Desenvolvimento de Programas Paralelos

O *Ambiente Visual de Desenvolvimento de Programas Paralelos de Tempo Real*[*] [19] oferece suporte ao *kernel* de tempo real Virtuoso. Este ambiente oferece uma ferramenta para desenvolvimento de programas paralelos, onde as aplicações são representadas graficamente, com objetos de *microkernel* denotados por retângulos e serviços de *microkernel* denotados por conexões entre estes objetos. Através de um editor de texto, as informações da representação gráfica podem ser complementadas com código fonte C/C++, de acordo com a finalidade específica da aplicação. A partir destas informações, o código fonte da aplicação é automaticamente gerado, compilado e *linkado*, para ser executado na máquina paralela.

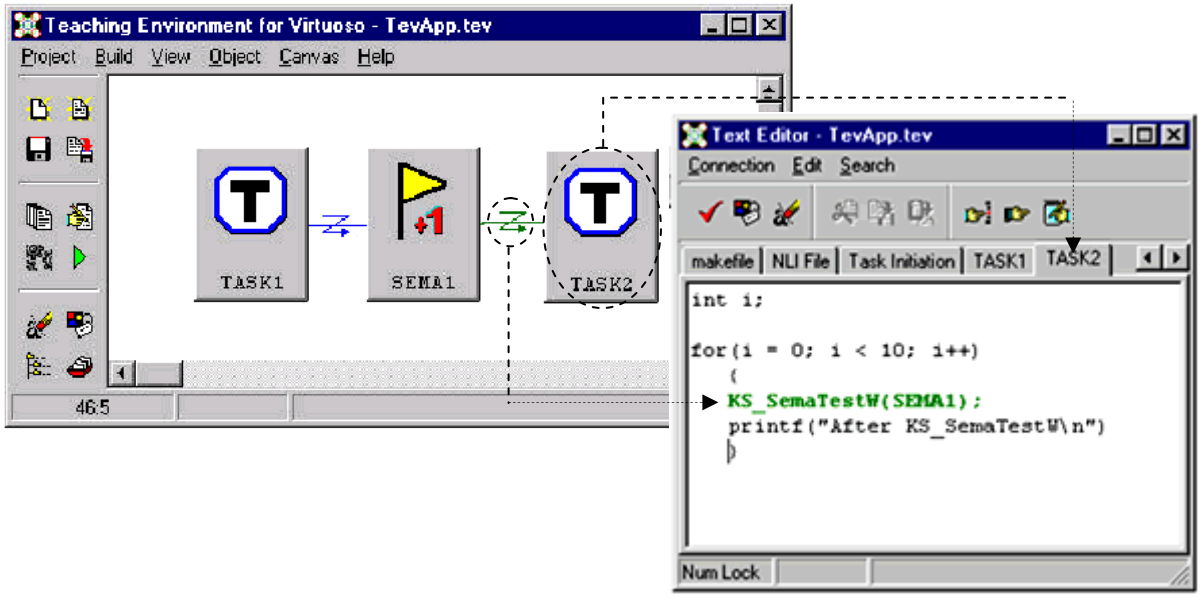


FIGURA A.2 – Gerador de Programas do Ambiente Visual de Desenvolvimento de Programas Paralelos de Tempo Real.

As propriedades dos objetos de *microkernel* e o mapeamento destes objetos nos diversos processadores disponíveis podem ser especificados num editor de propriedades.

[*] Projeto desenvolvido no Departamento de Computação/UFSCAR em cooperação com a empresa EONIC SYSTEMS (http://www.eonic.com). O ambiente denominado TEV (Teaching Environment for Virtuoso) está disponível para downloading no endereço: http://www.dc.ufscar.br/~tev.

A.4 Ambiente de Desenvolvimento de Programas de Visualização - IDL™

Interactive Data Language - IDL [6] é uma linguagem de programação orientada a *array* para desenvolvimento de aplicações de visualização e análise de dados. A linguagem interpretada IDL possui inúmeras funções matemáticas, além de funções para processamento digital de sinais, processamento de imagens e visualização de volume.

Aplicações sofisticadas podem ser facilmente desenvolvidas e executadas no ambiente de desenvolvimento IDL. O programador pode criar interfaces gráficas de usuário e escrever aplicações usando a linguagem IDL. Além disso, procedimentos ou funções especializadas podem ser escritos em C/C++ e adicionadas para a linguagem IDL [7].

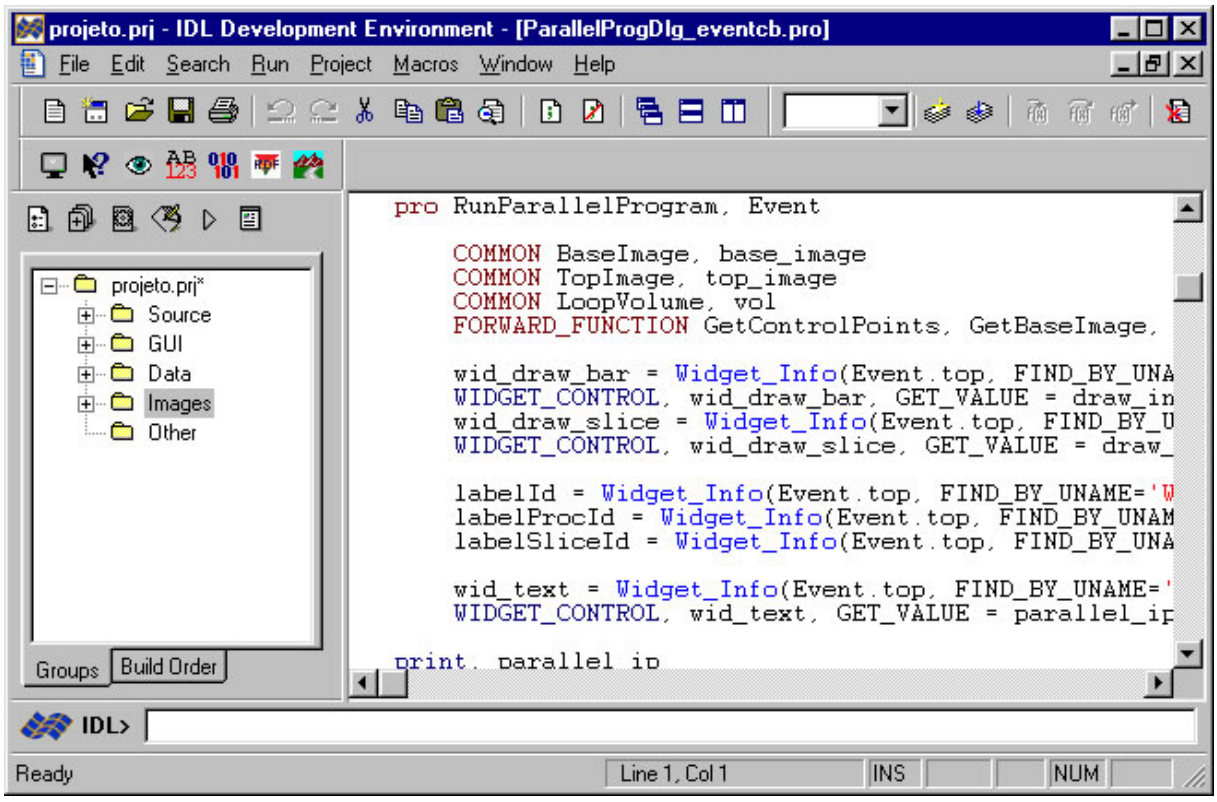
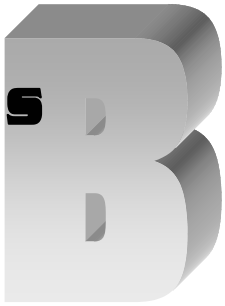


FIGURA A.3 – Ambiente de Desenvolvimento IDL.

Apêndices

Parte II

Vetores Aleatórios



Na prática, muitos problemas envolvem fenômenos aleatórios que são observações de natureza vetorial, como um sinal elétrico $X(t)$ medido nos instantes t_1, t_2,..., t_n. Assim, um vetor aleatório $\mathbf{X} = [X_1 \ldots X_n]^T$ é observado com n variáveis aleatórias X_i representando as medidas do sinal no instante t_i.

B.1 Vetor Esperado

O *vetor esperado* ou *vetor média* de um vetor aleatório $\mathbf{X} = [X_1 \ldots X_n]^T$ é definido por um vetor $\boldsymbol{\mu} = [\mu_1 \ldots \mu_n]^T$ cujos elementos são dados por

$$\mu_i = \int_{-\infty}^{+\infty} x_i p(x_i) dx_i , \qquad \text{com } i = 1,\ldots, n, \tag{B.1}$$

onde $p(x_i)$ é a densidade de probabilidade de X_i, definida por

$$p(x_i) = \int_{-\infty}^{+\infty} \ldots \int_{-\infty}^{+\infty} p(X) dx_1 \ldots d_{x_{i-1}} d_{x_{i+1}} \ldots dx_n \ . \tag{B.2}$$

B.2 Matriz de Covariância

A *matriz de covariância* **K** associada com um vetor aleatório **X** é definida por

$$\mathbf{K} = E[(\mathbf{X} - \boldsymbol{\mu}).(\mathbf{X} - \boldsymbol{\mu})^T]. \tag{B.3}$$

Os elementos c_{ij} da matriz são

$$c_{ij} = E[(X_i - \mu_i).(X_j - \mu_j)], \quad \text{com } i, j = 1, ..., n. \tag{B.4}$$

Os elementos diagonais c_{ii} da matriz de covariância são as *variâncias* σ_i^2 de X_i, e os elementos c_{ij} fora da diagonal são as *covariâncias* de X_i e X_j. Como $c_{ij} = c_{ji}$, as matrizes de covariância são simétricas.

$$\mathbf{K} = \begin{bmatrix} \sigma_1^2 & \cdots & c_{1n} \\ \vdots & \sigma_i^2 & \vdots \\ c_{n1} & \cdots & \sigma_n^2 \end{bmatrix}$$

Desenvolvendo a equação (B.3),

$$\mathbf{K} = E[\mathbf{X}\mathbf{X}^T] - E[\mathbf{X}]\,\mu^T - \mu E[\mathbf{X}^T] + \mu\mu^T$$

$$= E[\mathbf{X}\mathbf{X}^T] - \mu\mu^T. \tag{B.5}$$

A matriz $\mathbf{S} = E[\mathbf{X}\mathbf{X}^T]$ é chamada de *matriz de auto-correlação* de **X**. A relação entre a matriz de covariância **K** e a matriz de auto-correlação **S** é dada por

$$\mathbf{K} = \mathbf{S} - \mu\mu^T \qquad \text{ou} \qquad \mathbf{S} = \mathbf{K} + \mu\mu^T. \tag{B.6}$$

B.3 Matriz de Correlação

A *matriz de correlação* **R** é uma matriz simétrica definida por

$$\mathbf{R} = \begin{bmatrix} 1 & \rho_{12} & \rho_{1n} \\ \rho_{12} & \ddots & \vdots \\ \rho_{1n} & \cdots & 1 \end{bmatrix}.$$

Os *coeficientes de correlação* são

$$\rho_{ij} = \frac{c_{ij}}{\sigma_i \sigma_j}, \quad \text{com } \rho_{ii} = 1,$$

onde ρ_{ij} é o coeficiente de correlação de X_i e X_j, c_{ij} é a covariância de X_i e X_j e σ_i é o desvio padrão de X_i.

Os elementos da matriz de covariância **K** podem ser expressos como

$$c_{ii} = \sigma_i^2 \quad \text{ou} \quad c_{ij} = \rho_{ij}\sigma_i\sigma_j ,$$

onde σ_i^2 é a variância de X_i. Assim, a matriz de covariância **K** pode ser escrita como uma combinação de duas matrizes

$$\mathbf{K} = \Gamma\mathbf{R}\Gamma ,$$

onde Γ é a matriz diagonal de desvios padrão,

$$\Gamma = \begin{bmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_n \end{bmatrix},$$

e **R** é a matriz de coeficientes de correlação.

B.4 Classificação de Vetores Aleatórios

Vetores aleatórios são freqüentemente classificados como sendo não-correlacionados, ortogonais ou independentes.

Definição B.1. Dois vetores aleatórios *n*-dimensionais **X** e **Y** são considerados não correlacionados se

$$E[\mathbf{X}\mathbf{Y}^T] = E[\mathbf{X}]E[\mathbf{Y}^T]. \tag{B.7}$$

Definição B.2. Dois vetores aleatórios **X** e **Y** são considerados ortogonais se

$$E[\mathbf{X}^T\mathbf{Y}] = \sum_{i=1}^{n} E[X_i Y_i] = 0. \tag{B.8}$$

Definição B.3. Dois vetores aleatórios **X** e **Y** são considerados independentes se

$$p(\mathbf{X},\mathbf{Y}) = p(\mathbf{X})p(\mathbf{Y}), \tag{B.9}$$

onde $p(\mathbf{X})$ e $p(\mathbf{Y})$ são as funções densidade de probabilidade de **X** e **Y**, respectivamente.

Curva Bezier

C.1 Representação de Curvas

C.1.1 Curvas não paramétricas

Uma representação não paramétrica pode ser explícita ou implícita. Para uma curva plana, uma forma não paramétrica explícita é dada por

$$y = f(x).$$

Nesta forma, para cada valor x, somente um valor y é obtido. Conseqüentemente, curvas fechadas (ou com múltiplos valores, como um círculo) não podem ser representadas explicitamente. Representações implícitas da forma

$$f(x, y) = 0$$

não tem esta limitação.

C.1.2 Curvas paramétricas

A forma paramétrica é apropriada para representar curvas fechadas e com múltiplos valores. Na forma paramétrica, cada coordenada de um ponto da curva é representada como uma função de um único parâmetro. Para uma curva

bidimensional, com parâmetro *t*, as coordenadas cartesianas de um ponto na curva são

$$x = x(t)$$

$$y = y(t)$$

O vetor que define a posição de um ponto na curva é dado por

$$\mathbf{P}(t) = [x(t) \;\; y(t)]$$

A derivada (vetor tangente à curva) no ponto *t* é dada por

$$\mathbf{P}'(t) = \frac{d}{dt}\mathbf{P}(t) = \left[\frac{d}{dt}x(t) \quad \frac{d}{dt}y(t)\right] = \left[x'(t) \quad y'(t)\right]$$

A inclinação da curva é dada por

$$\frac{dy}{dx} = \frac{dy/dt}{dx/dt} = \frac{y'(t)}{x'(t)}$$

A direção do vetor tangente é

$$\frac{\mathbf{P}'(t)}{\|\mathbf{P}'(t)\|}$$

onde $\|\mathbf{P}'(t)\|$ é a velocidade paramétrica.

A forma não paramétrica pode ser obtida da forma paramétrica eliminando-se o parâmetro *t* para obter uma única equação em função de *x* e *y*.

C.2 Curva Bezier

A curva Bézier foi desenvolvida pelo engenheiro Pierre Bézier durante seus trabalhos em projetos de automóveis para a Renault francesa no início da década de 1970. Uma curva Bezier é especificada por um conjunto de pontos de controle P_i, que indica a forma geral da curva. A curva é *controlada* pelos pontos, e não necessariamente deve passar por eles.

Matematicamente, uma curva Bézier paramétrica de ordem *n* é definida como

$$P(t) = \sum_{i=0}^{n} B_i(t) P_i \qquad 0 \le t \le 1$$

onde as funções *blending*, ou funções base da curva Bézier são

$$B_i(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

Para n pontos de controle (P_0, P_1, …, P_{n-1}), a curva Bézier é parametrizada por uma função polinomial de grau n–1.

C.2.1 Curva Bezier Cúbica

Uma curva Bezier cúbica é gerada com quatro pontos de controle. A Figura C.1 mostra alguns exemplos de curvas Bezier cúbicas.

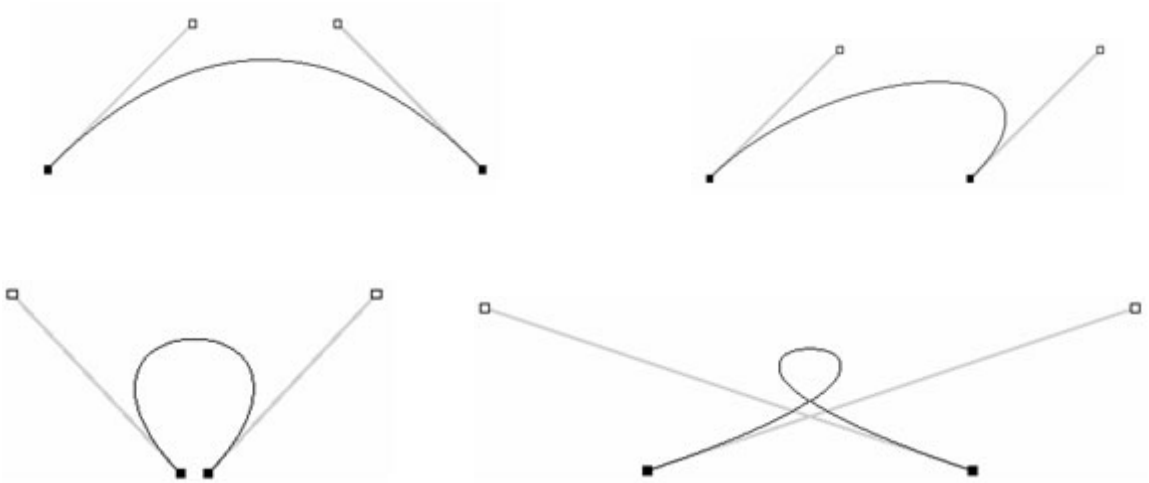


FIGURA C.1 – Curvas Bezier cúbicas e seus respectivos pontos de controle.

A curva está contida no **fecho convexo** do polígono de definição, isto é, no maior polígono convexo que pode ser obtido com os vértices do polígono de definição. O primeiro e o último ponto da curva coincidem com o primeiro e o último ponto do polígono de definição, respectivamente.

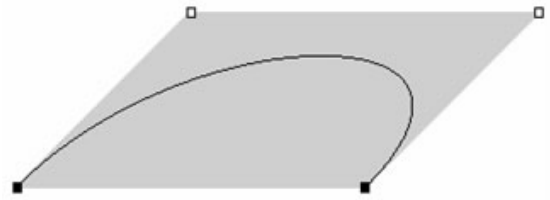


FIGURA C.2 – Polígono de definição da curva Bezier.

As quatro funções *blending* para curvas Bezier cúbicas são

$$B_0^3(t) = (1-t)^3$$

$$B_1^3(t) = 3t(1-t)^2$$

$$B_2^3(t) = 3t^2(1-t)$$

$$B_3^3(t) = t^3$$

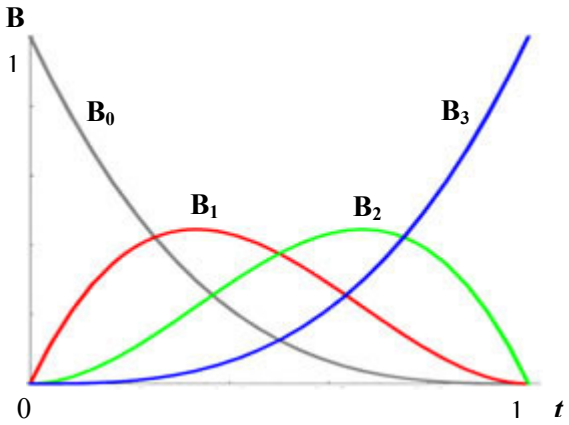


FIGURA C.3 – Funções blending para curvas Bezier cúbicas.

A equação da curva Bezier cúbica é

$$P(t) = (1-t)^3 P_0 + 3(1-t)^2 tP_1 + 3(1-t)t^2 P_2 + t^3 P_3$$

ou seja, as coordenadas da curva no ponto t, com t variando de 0 a 1, são

$$x(t) = (1-t)^3 x_0 + 3(1-t)^2 tx_1 + 3(1-t)t^2 x_2 + t^3 x_3$$
$$y(t) = (1-t)^3 y_0 + 3(1-t)^2 ty_1 + 3(1-t)t^2 y_2 + t^3 y_3$$
$$z(t) = (1-t)^3 z_0 + 3(1-t)^2 tz_1 + 3(1-t)t^2 z_2 + t^3 z_3$$

Podemos escrever a função da curva Bezier na forma matricial como

$$P(t) = \begin{bmatrix} (1-t)^3 & 3t(1-t)^2 & 3t^2(1-t) & t^3 \end{bmatrix} \cdot \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}$$

ou então, na forma,

$$P(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} M_B . \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

onde a matriz de Bezier é

$$M_B = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Da mesma forma, a derivada da curva Bezier pode ser escrita como

$$P'(t) = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} M_B . \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

Código Fonte IDL

D.1 Interpolação 3D de Imagens usando a Teoria de Estimação Bayesiana

O programa completo do método de "Interpolação 3D de Imagens usando a Teoria de Estimação Bayesiana" implementado na linguagem de programação IDL está listado abaixo. A numeração das linhas não faz parte do código do programa.

```
;-------------------------------------------------------------------------------------------------------
;Cálculo da média amostral e covariância amostral das imagens observadas
;-------------------------------------------------------------------------------------------------------

1       mA = MEAN(A)            ; Média de A.
2       mB = MEAN(B)            ; Média de B.
3       mC = (mA + mB) / 2      ; Média da imagem interpolada (média aritmética).

4       vA = VARIANCE(A)        ; Variância da imagem A.
5       vB = VARIANCE(B)        ; Variância da imagem B.
6       vC = SQRT(vA * vB)      ; Variância da imagem interpolada (média
                                ; geométrica).

7       A = A – mA
8       B = B – mB
```

```
;-------------------------------------------------------------------------------------------------------------
;Cálculo do vetor de covariâncias entre pixel estimado e pixels observados
;-------------------------------------------------------------------------------------------------------------
9       corrAB = CORRELATE(A, B)        ; Coeficiente de correlação espectral entre
                                        ; imagens.
10      corr3D = SQRT(corrAB)           ; Coeficiente de correlação espectral entre
                                        ; a imagem observada e a interpolada.
11      covAC = corr3D * SQRT(vA * vC)  ; Covariância entre a imagem A e a imagem
                                        ; interpolada.
12      covBC = corr3D * SQRT(vB * vC)  ; Covariância entre a imagem B e a imagem
                                        ; interpolada.
13      roh = .97                       ; Coeficiente de correlação horizontal.
14      rov = .97                       ; Coeficiente de correlação vertical.

15      cA = covAC*roh
16      cAv = covAC*rov
17      cAhv = covAC*roh*rov
18      cBh = covBC*roh
19      cBv = covBC*rov
20      cBhv = covBC*roh*rov

      ;Vetor de covariâncias entre uma imagem observada e o ponto a ser estimado.
21      kyx= [ cAhv, cAv, cAhv, cA, covAC, cA, cAhv, cAv, cAhv,
               cBhv, cBv, cBhv, cB, covBC, cB, cBhv, cBv, cBhv  ]


22      s = SIZE(A)
23      nx = s[1]                       ; Número de colunas da imagem.
24      ny = s[2]                       ; Número de linhas da imagem.


;-------------------------------------------------------------------------------------------------------------
;Cálculo da matriz de covariância amostral entre imagens observadas
;-------------------------------------------------------------------------------------------------------------
25      Kxx = FLTARR(18,18)                 ; Alocação da matriz de covariância.
26      X = FLTARR(18)                      ; Alocação do vetor de observações.

27      N = 0.0
28      for    i = 1,  nx – 4    do begin
29             for    j = 1,  ny – 4    do begin
30                    A1 = A[i : i+2, j : j+2]      ; Janela 3x3 da imagem A.
31                    B1 = B[i : i+2, j : j+2]      ; Janela 3x3 da imagem B.
32                    A9 = REFORM(A1, 9)            ; Vetor linha com 9 pixels de A.
33                    B9 = REFORM(B1, 9)            ; Vetor linha com 9 pixels de B.
34                    X[0: 8]  = A9                 ; Vetor linha com 18 pixels.
35                    X[9:17] = B9
36                    Xt = TRANSPOSE(X)             ; Vetor coluna com 18 pixels.
37                     t  = Xt ## X                 ; Multiplicação de matrizes.
38                     Kxx = Kxx + t                ; Somatório da matriz.
39                     N = N + 1.0                  ; Número de vetores observados.
40             end
41      end
42      Kxx = Kxx / N                       ; Matriz de covariância amostral.
```

```
;-------------------------------------------------------------------------------------------------------------
;Cálculo dos coeficientes para estimação
;-------------------------------------------------------------------------------------------------------------
43      invKxx = INVERT(Kxx)        ; Matriz inversa de K.

44      ax = kyx ## invKxx          ; Coeficientes procurados.

45      a1 = REFORM(ax, 9, 2)       ; Transforma vetor linha 1x18 para matriz 2x9.
46      a2 = a1[*,0]                ; Vetor linha com primeiros 9 elementos.
47      a3 = a1[*,1]                ; Vetor linha com últimos 9 elementos.
48      a4 = REFORM(a2,3,3)         ; Transforma vetor linha 1x9 para matriz 3x3.
49      a5 = REFORM(a3,3,3)


;-------------------------------------------------------------------------------------------------------------
;Interpolação 3D da imagem
;-------------------------------------------------------------------------------------------------------------
        ; Multiplicação pelos coeficientes em blocos 3x3 da janela A1.
        ; Multiplicação pelos coeficientes em blocos 3x3 da janela B1.
50      ITP = CONVOL(A, a4) + CONVOL(B, a5)
51      ITP = ITP + mC                                  ; Imagem interpolada final.
52      imagem = ITP[1:nX–2, 1:nY–2]

53      erro = vC - (kyx ## invKxx ## TRANSPOSE(kyx))  ; Erro de interpolação.
```

Apêndices

Parte III

Imagens de Reconstrução e Visualização 3D

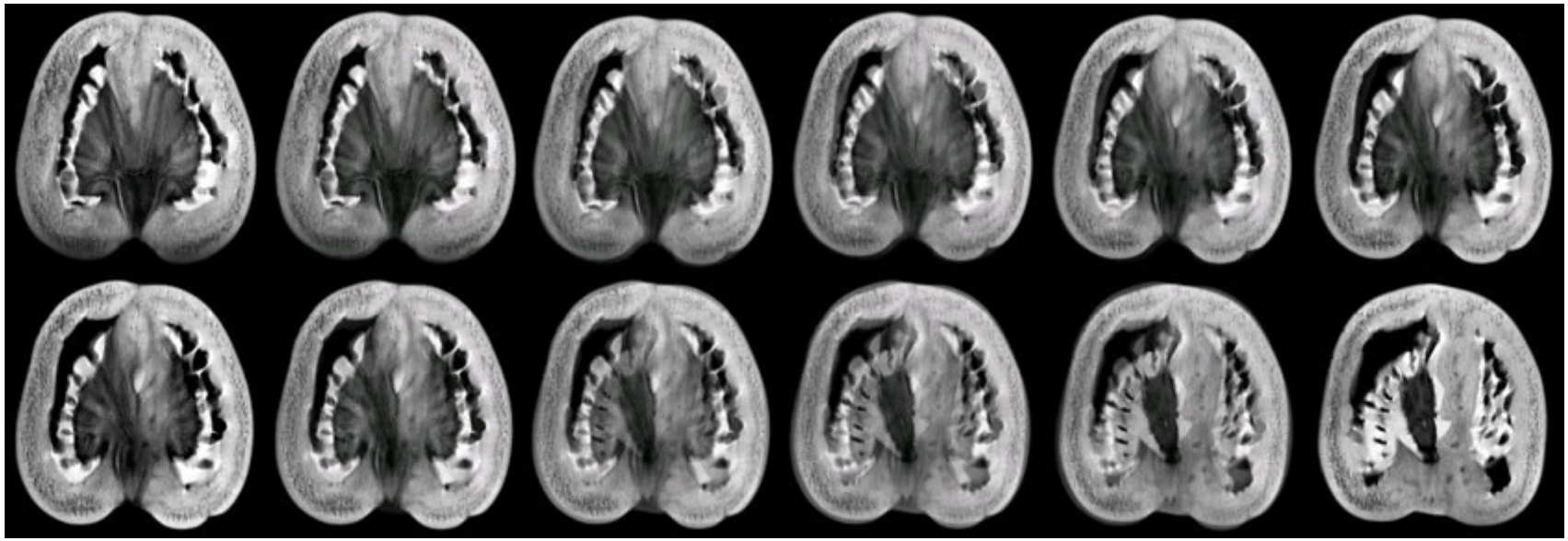


FIGURA E.1 – Conjunto de 12 imagens reconstruídas a partir de 3 imagens tomográficas de ressonância magnética do tomate.

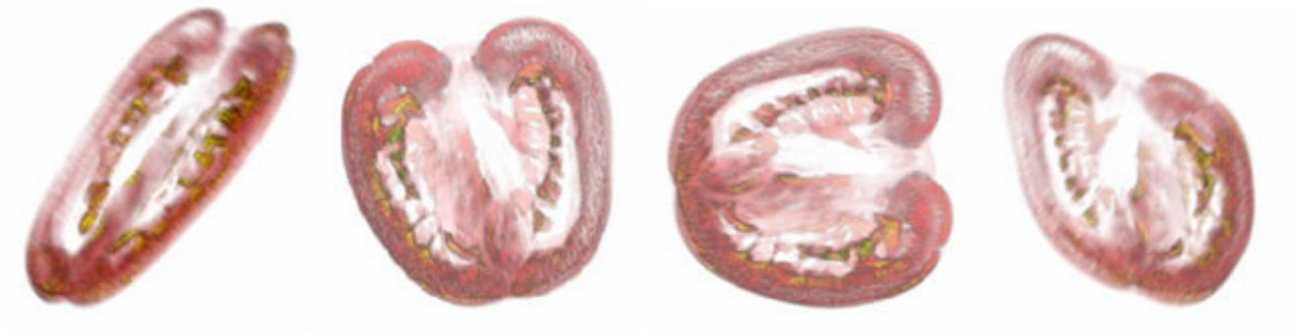


FIGURA E.2 – Visualização 3D do tomate após a reconstrução do volume usando cross-dissolve.

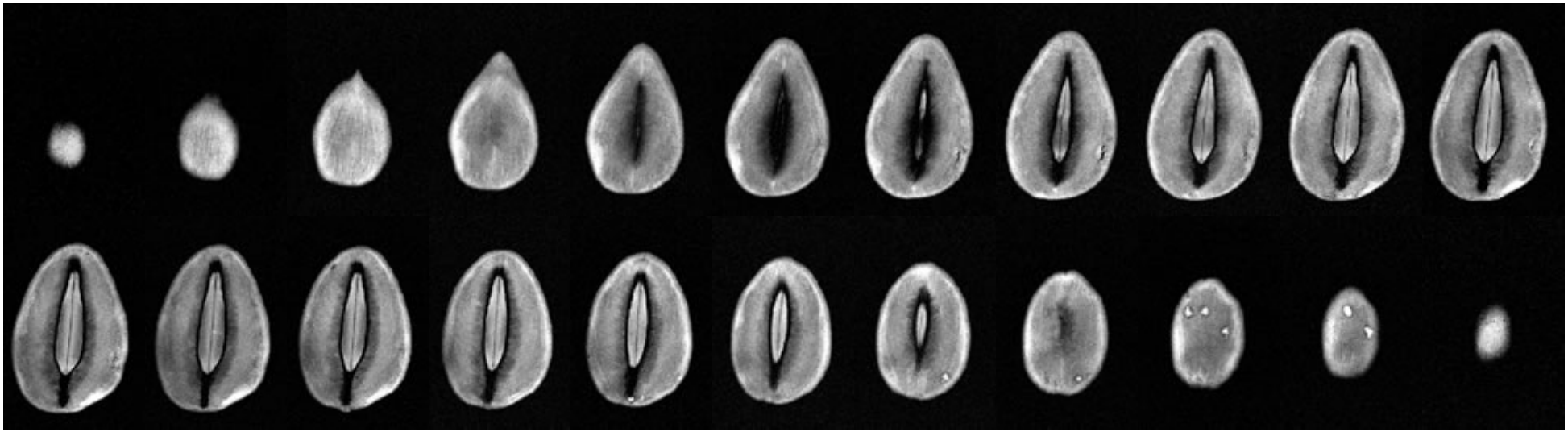


FIGURA E.3 – Conjunto de 22 imagens originais de tomografia de ressonância magnética nuclear da manga.

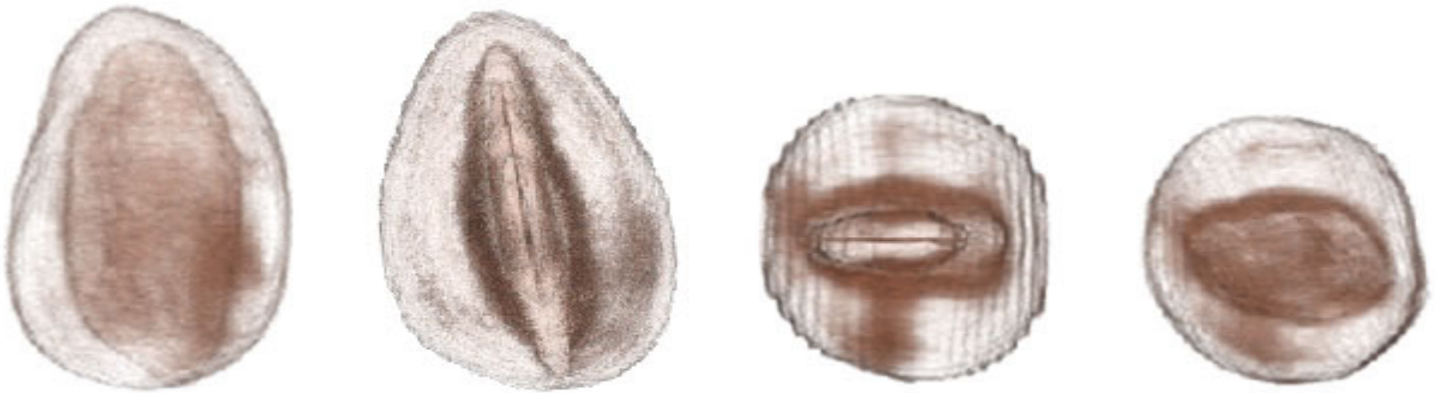


FIGURA E.4 – Visualização 3D da manga após a reconstrução de volume usando cross-dissolve. Total: 189 imagens.

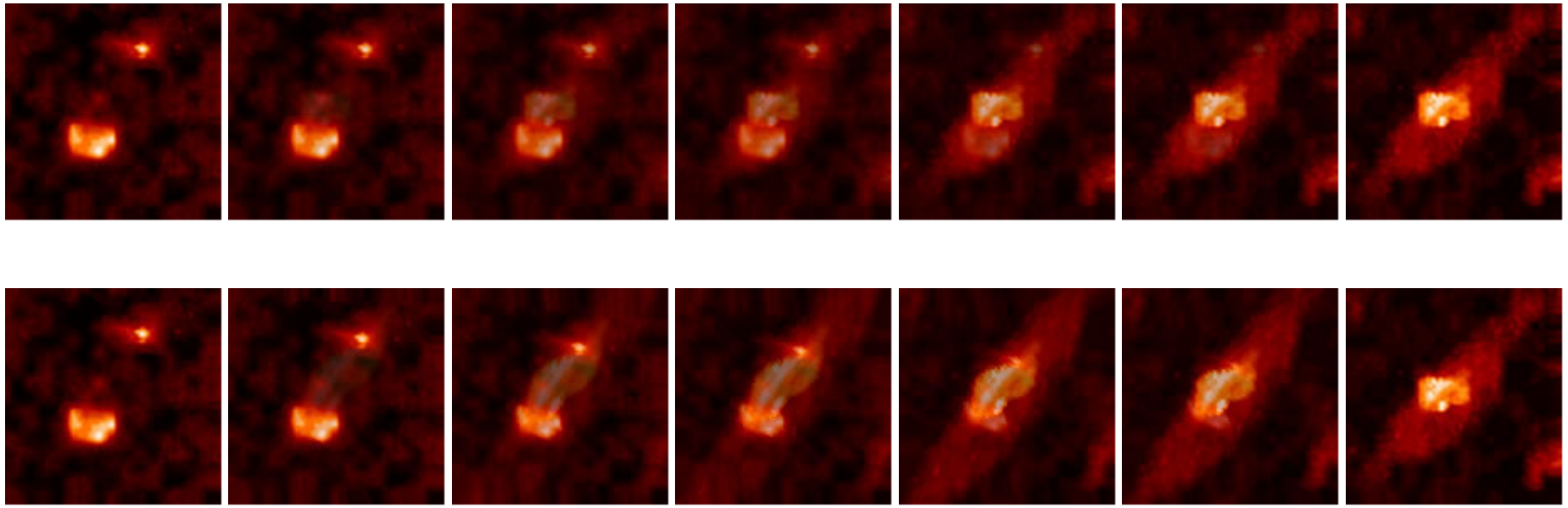


FIGURA E.5 – Algumas imagens intermediárias do loop geradas com: (a) cross-dissolve e (b) metamorfose de imagens.

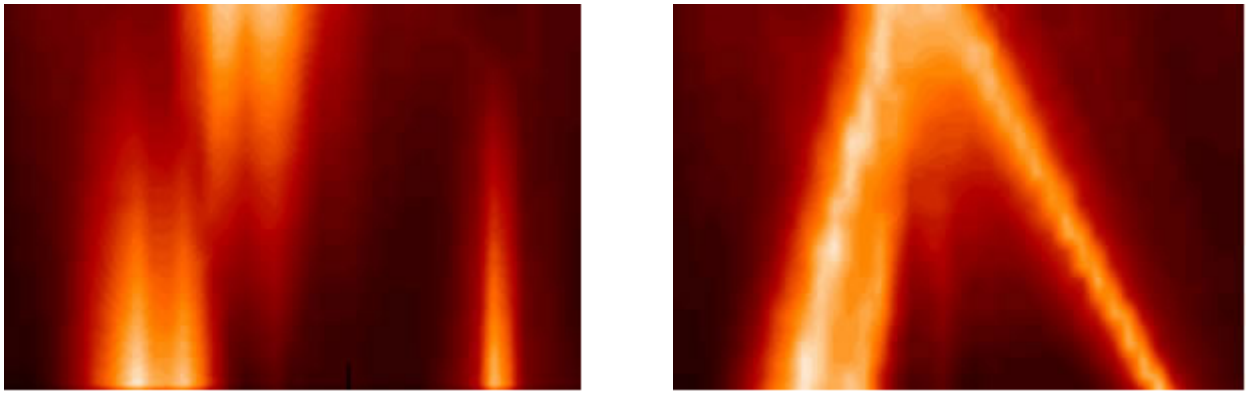


FIGURA E.6 – Loop 3D reconstruído com: (a) cross-dissolve e (b) metamorfose de imagens.

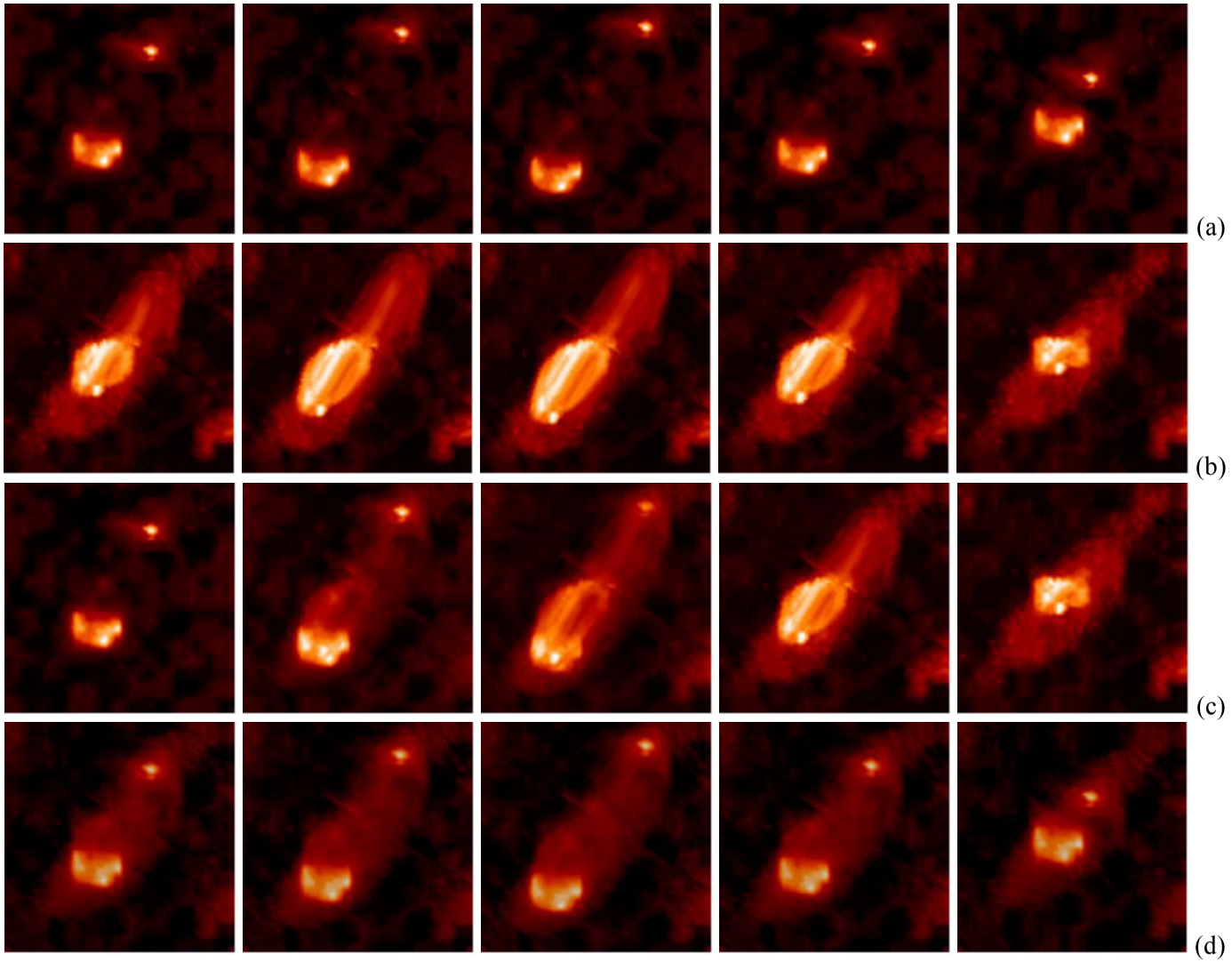


FIGURA E.7 – Imagens geradas com o método de reconstrução 3D controlado por curva Bezier: (a) Imagens deformadas da base; (b) Imagens deformadas do topo; (c) Imagens intermediárias entre (a) e (b) usando cross-dissolve; e (d) Imagens intermediárias usando interpolação bayesiana.

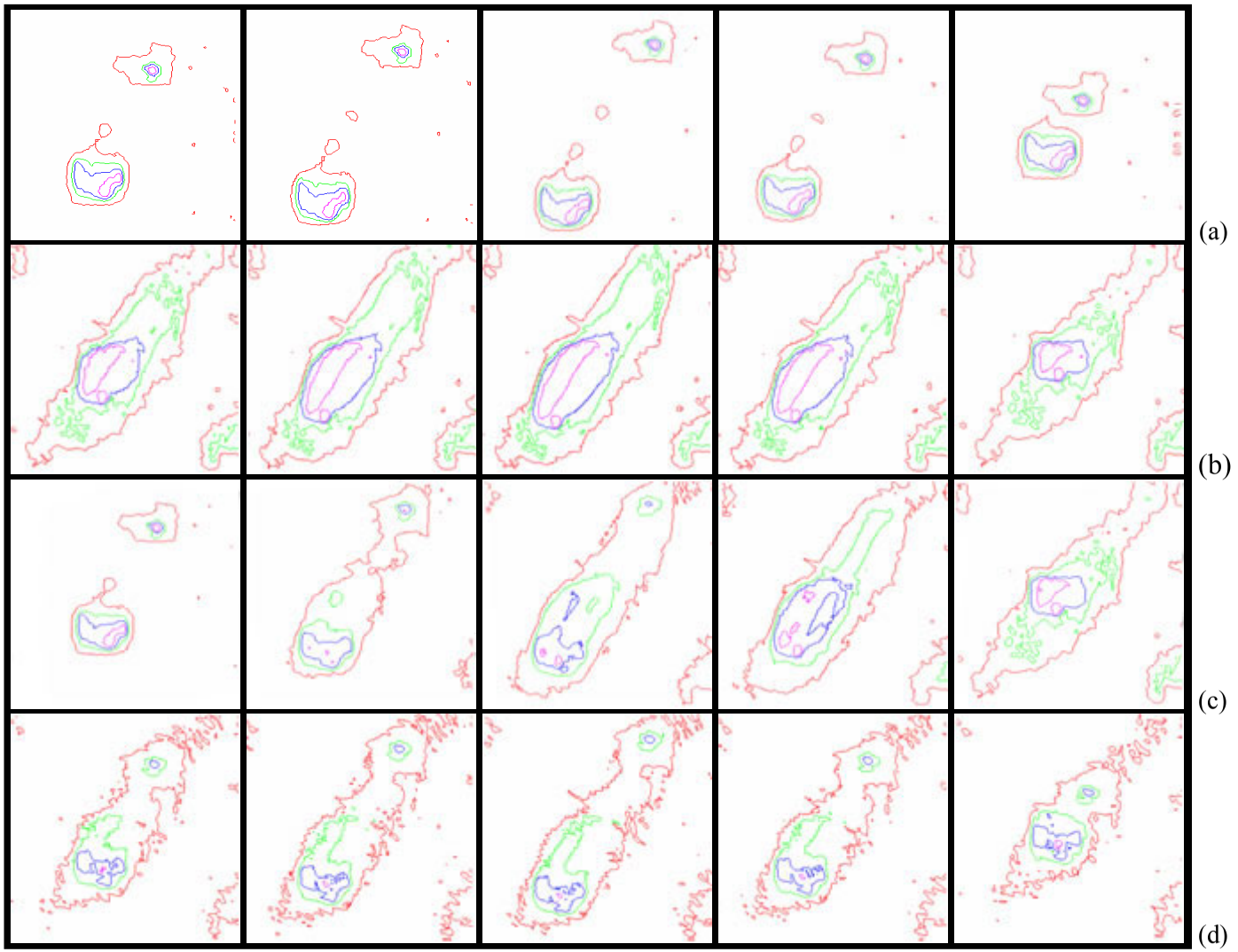


FIGURA E.8 – Contorno das imagens geradas com o método de reconstrução 3D controlado por curva Bezier: (a) Bases deformadas; (b) Topos deformados; (c) Intermediárias usando cross-dissolve; e (d) Intermediárias usando interpolação bayesiana

Ambientes de Desenvolvimento do Programa

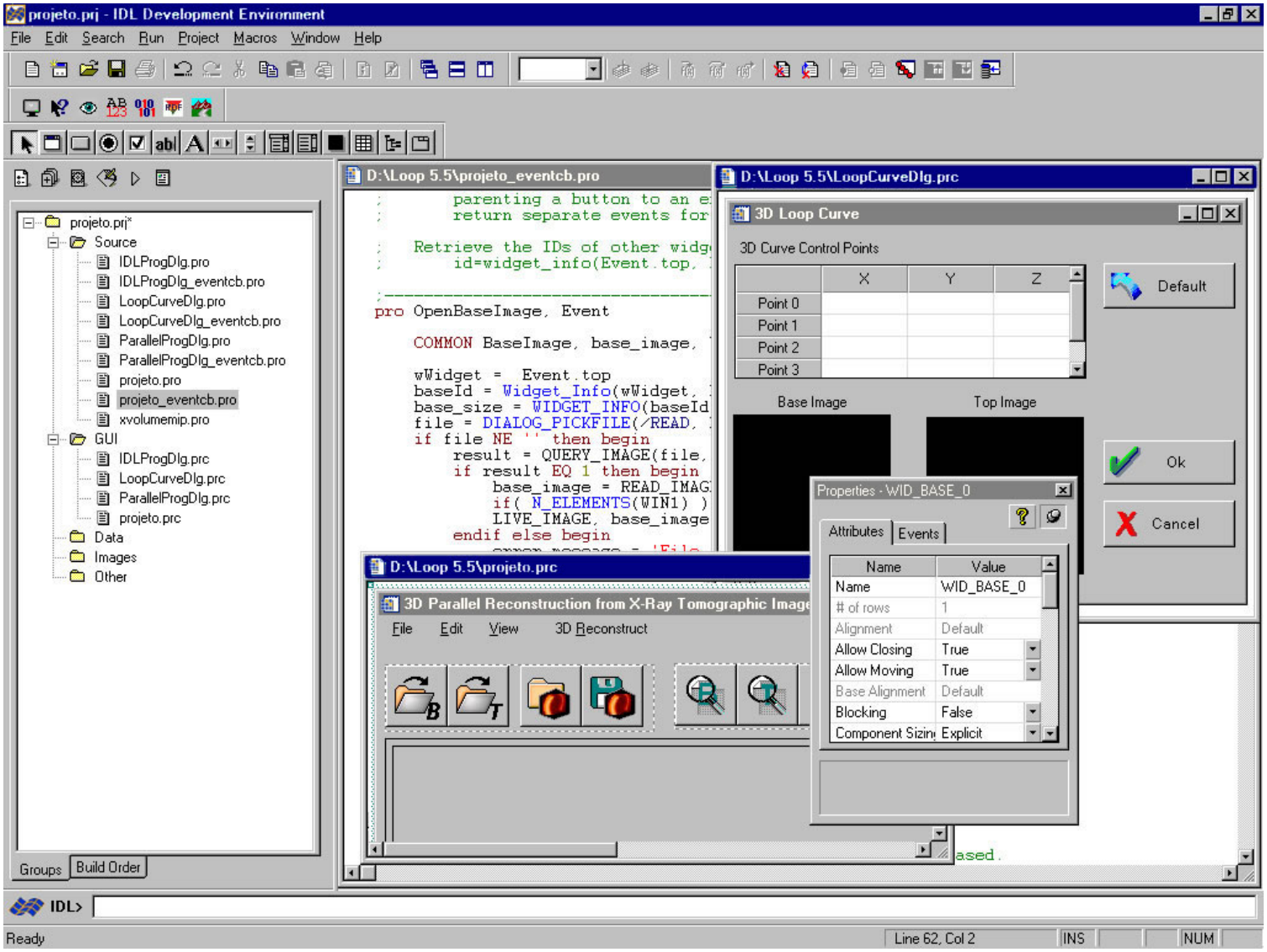


FIGURA F.1 – Interface gráfica do ambiente IDL para desenvolvimento do programa de visualização 3D

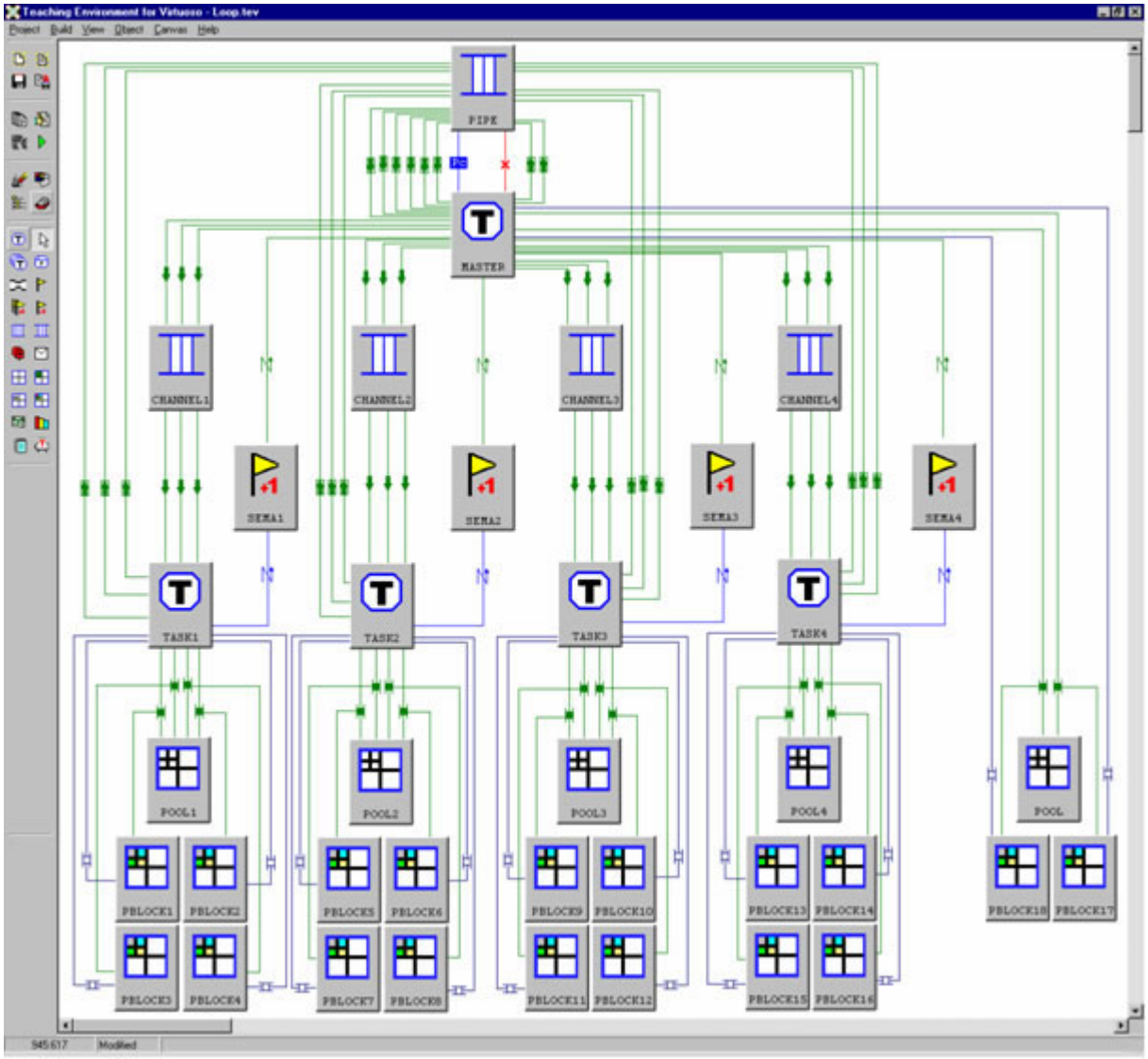


FIGURA F.2 – **Representação gráfica do programa paralelo de deformação de imagens tomográficas do loop no Ambiente Visual de Desenvolvimento de Programas Paralelos de Tempo Real.**