Universidade Federal de Santa Maria
Centro de Tecnologia
Programa de Pós-Graduação em Engenharia da Produção

MeSegHi: Um Método de Segmentação para o Processamento Linear e Não-Linear de Imagens

DISSERTAÇÃO DE MESTRADO

Luciana Vescia Lourega

Santa Maria, RS, Brasil

2006

MESEGHI: UM MÉTODO DE SEGMENTAÇÃO PARA O PROCESSAMENTO LINEAR E NÃO-LINEAR DE IMAGENS

por

Luciana Vescia Lourega

Dissertação apresentada ao Curso de Mestrado do Programa de Pós-Graduação em Engenharia da Produção, área de concentração em Tecnologia da Informação, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Mestre em Engenharia da Produção**

Orientador: Marcos Cordeiro d'Ornellas

Santa Maria, RS, Brasil

2006

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Vescia Lourega, Luciana

MeSegHi: Um Método de Segmentação para o Processamento Linear e Não-Linear de Imagens / Luciana Vescia Lourega. – Santa Maria: Programa de Pós-Graduação em Engenharia da Produção, 2006.

95 f.: il.

Dissertação (mestrado) – Universidade Federal de Santa Maria. Programa de Pós-Graduação em Engenharia da Produção, Santa Maria, BR–RS, 2006. Orientador: Marcos Cordeiro d'Ornellas.

1. Processamento de Imagens. 2. Segmentação de Imagens. 3. Padrões de Projeto. 4. Programação Orientada por Objetos. I. d'Ornellas, Marcos Cordeiro. II. Título.

✍ Essa dissertação foi escrita utilizando a ferramenta de edição LaTeX 2_ε

Universidade Federal de Santa Maria
Centro de Tecnologia
Programa de Pós-Graduação em Engenharia da Produção

A Comissão Examinadora, abaixo assinada,
aprova a Dissertação de Mestrado

MESEGHI: UM MÉTODO DE SEGMENTAÇÃO PARA O PROCESSAMENTO LINEAR E NÃO-LINEAR DE IMAGENS

elaborada por

Luciana Vescia Lourega

como requisito parcial para obtenção do grau de

Mestre em Engenharia da Produção

COMISSÃO EXAMINADORA:

Prof. Ph.D. Marcos Cordeiro d'Ornellas
(Presidente/Orientador - UFSM-Departamento de Eletrônica e Computação)

Prof[a]. Dr[a]. Ana Maria Marques da Silva
(UFSM-Departamento de Física)

Prof. Dr[o]. José Antônio Trindade Borges da Costa
(PUCRS-Faculdade de Informática)

Santa Maria, 24 de Novembro de 2006

Dedico esta dissertação aos meus pais
José M. Lourega e Maria Rita V. Lourega,
ao meus irmãos Rogerio e Marilu V. Lourega
e ao meu namorado Dirceu Luis Conrad.

AGRADECIMENTOS

Em primeiro lugar, agradeço a Deus e aos anjos protetores por ter me iluminado e por ter-me dado forças para concluir este trabalho.

Agradeço ao professor e orientador Marcos Cordeiro d'Ornellas por seus ensinamentos, incentivo e dedicação.

Quero agradecer também aos professores José Antônio Trindade Borges da Costa e Daniela Ushizima que sempre me conduziram em meus estudos com muito carinho e atenção.

Um agradecimento especial as minhas amigas Gabrielle Dias Freitas, Thais Frazzon, Tanise Dias Freitas, Ellen Dias Soares e Susana Mussoi, as quais estiveram sempre ao meu lado, me ajudando e me incentivando. Jamais esquecerei de vocês.

Não posso esquecer é claro dos meus amigos Daniel Welfer e Diego Schmaedech Martins, que sempre me auxiliaram e incentivaram nos momentos difíceis. E, finalmente, a todos os integrantes do grupo PIGS (Grupo de Processamento de Informação Multimídia) que indiretamente, contribuiram para o desenvolvimento desse trabalho.

“Há homens que lutam um dia e são bons.
Há outros que lutam um ano e são melhores.
Há os que lutam muitos anos e são muitos bons.
Porém, há os que lutam toda a vida.
Esses são os imprescindíveis.”
Bertolt Brecht

RESUMO

Universidade Federal de Santa Maria
Centro de Tecnologia
Pós-Graduação em Engenharia da Produção
Programa de Pós-Graduação em Engenharia de Produção
Universidade Federal de Santa Maria

MESEGHI: UM MÉTODO DE SEGMENTAÇÃO PARA O PROCESSAMENTO LINEAR E NÃO-LINEAR DE IMAGEM

AUTORA: LUCIANA VESCIA LOUREGA
ORIENTADOR: MARCOS CORDEIRO D'ORNELLAS
Data e Local da Defesa: Santa Maria, 24 de Novembro de 2006.

O presente trabalho tem como objetivo desenvolver o método de segmentação híbrido, o qual utiliza duas técnicas de segmentação: uma linear, *Mean Shift*, e outra não-linear, *Watersheds* por Image Foresting Transform (IFT). Para implementar esse método utilizou-se um conjunto reduzido de cores obtido pela aplicação do procedimento *Mean-Shift*, como marcadores para o algoritmo IFT. Com o desenvolvimento dessa técnica, o usuário não necessita selecionar o conjunto de marcadores corretos para realizar o processo de segmentação por *Watersheds-IFT*, pois o método híbrido fará isso automaticamente. Se as sementes geradas não forem suficientes ou forem inseridas em uma parte da amostra que não esteja sendo analisada, o usuário, por meio da interface, poderá inserir ou retirar tais sementes, a fim de obter um melhor resultado. Com o objetivo de desenvolver um sistema robusto e eficiente, com alto grau de reusabilidade, optou-se por utilizar o paradigma de programação orientada a objetos, juntamente com a aplicação de padrões de projeto. A linguagem de programação utilizada é a Java, que além de dar suporte à orientação a objetos, possui uma *Application Programming Interface* (API) denominada *Java Advanced Imaging* (JAI) que permite a implementação de operações para o processamento de imagens com maior facilidade. A linguagem UML é usada para projetar o sistema, ajudando o desenvolvedor a extrair melhor as informações necessárias a fim de construir o software. Após desenvolver o método de segmentação híbrida são realizadas três aplicações a fim de que sua eficiência seja comprovada. A primeira aplicação busca auxiliar os patologistas a realizarem a classificação do exame de Papanicolaou em seus respectivos níveis; a segunda, realiza uma comparação entre os métodos de segmentação *Watersheds-IFT* e híbrido; e a terceira visa auxiliar profissionais da área médica, a medir a profundidade do câncer de pele em tecidos histológicos.

Palavras-chave: Processamento de Imagens, Segmentação de Imagens, Padrões de Projeto, Programação Orientada por Objetos.

ABSTRACT

Master Thesis
Graduate Program in Production Engineering
Federal University of Santa Maria

MESEGHI: A HYBRID SEGMENTATION METHOD FOR LINEAR AND NON-LINEAR IMAGE ANALYSIS

AUTHORESS: LUCIANA VESCIA LOUREGA
ADVISOR: MARCOS CORDEIRO D'ORNELLAS
Date and Place: Santa Maria, November, 24th, 2006.

This research aims to develop the hybrid segmentation's method which uses two techniques: linear mean-shift and non-linear image foresting transform watersheds (IFT). To implement this method the reduced color set was used, it was obtained by mean-shift application, like markers to IFT algorithm. With this technique, the user doesn't need to concern about selecting the correct set marker to Watershed-IFT segmentation's process, since hybrid method will do so automatically. If the seeds are not good enough or are placed just outside the region of interest, the user can add or remove such seeds in order to improve watershed segmentation results. With the purpose of developing a robust and efficient system, with high degree of reusability, it was decided to make use of the object-oriented paradigm together design patterns' application. The Java programming language was used to implement the hybrid's method because it supports object oriented and it has an Application Programming Interface (API) Java Advanced Imaging (JAI) that allows an easier image processing operations' implementation. The Unified Modeling Language (UML) was used to design the system; it helped the developer to extract the best necessary information to construct the software. After it develops the hybrid segmentation method were realized three applications to prove hybrid method efficience. The first application helps the patologystis to classify Papanicolau examination in yours respectives levels; the second makes a comparison between Watersheds-IFT and hybrid segmentation methods; and the third aim to help profissionals of medicine area to measure the depth of the skin cancer in histologys tissues.

Keywords: Software Engineering, Design Patterns, Object Oriented Paradigm.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API *(Application Programming Interface)*. Interface de programação para construção de software.

AWT *(Abstract Window Toolkit)*. API que provê, entre outros, componentes gráficos de interface com o usuário.

CBSD *(Component-Based Software Development)*. Desenvolvimento de software baseado em componentes.

DBC *(Component Based Development)*. Desenvolvimento Baseado em Componentes.

GOF *(Gang of Four)* The Gang of Four (Gangue dos Quatro).Grupo de amigos que definiram 23 padrões de projeto, os quais foram divididos em três categorias: criacional, estrutural e comportamental.

GUI *(Graphics User Interface)*. Interface gráfica com o usuário.

HCI *(Human-Computer Interaction)*. Interação entre homem e máquina.

HTML *(HyperText Markup Language)*. Linguagem de Formatação de Hipertexto. Trata-se de uma linguagem de marcação utilizada para produzir páginas na Internet.

JAI *(Java Advanced Imaging)*. API Java para processamento de imagens.

JDK *(Java Development Kit)*. Kit de Desenvolvimento Java.

OMT *(Object Modeling Technique)*. Técnica de Modelagem de Objetos.

OOP *(Object-Oriented Programming)*. Programação orientada por objetos.

OOSE *(Object Oriented Software Engineering)*. Engenharia de Software Orientada a Objetos.

PIGS *(Multimedia Information Processing Group)*. Grupo de Processamento de Informação Multimídia

RTF *(Rich Text Format)*. É um tipo de arquivo que foi originalmente criado para transferir documentos de texto formatados entre aplicativos, mesmo que sejam executados em plataformas diferentes. É, portanto, um formato de arquivo que vários processadores de texto entendem.

SQL *(Structured Query Language)*. Linguagem de Consulta Estruturada.

UML *(Unified Modeling Language)*. Linguagem de Modelagem Unificada.

1 INTRODUÇÃO

Sistemas de processamento e análise de imagens vêm sendo empregados com sucesso na solução de problemas das diversas áreas do conhecimento. Na área médica, por exemplo, as imagens são utilizadas para diagnosticar patologias; no domínio geoespacial, elas são utilizadas para visualizar o estado climático de uma região ou até mesmo para registrar o relevo de outros planetas [KAS 1995].

Nestes sistemas de processamento de imagens, técnicas sofisticadas de segmentação e análise de imagens possibilitam uma inspeção automatizada dos artefatos analisados de forma rápida e precisa. Por este motivo, tais tecnologias vêm substituindo processos baseados na visão humana, lentos e sujeitos a falhas.

A segmentação, por sua característica, é uma tarefa de processamento de imagens desafiadora. O ser humano é capaz de, visualmente, saber o que extrair da imagem. Porém é freqüentemente observada a incapacidade do computador em executar a mesma tarefa.

A técnica de segmentação de imagem é uma etapa importante para a obtenção de um bom resultado. A segmentação consiste em dividir uma imagem em suas partes ou objetos constituintes, sendo uma das mais difíceis tarefas no processamento de imagens [GON 2000].

A grande dificuldade da segmentação reside no fato de não se conhecer, inicialmente, o número e o tipo de estrutura que se encontram numa imagem. Essas estruturas são identificadas a partir da geometria, da forma, da topologia, da textura, da cor ou do brilho. É com base nessas informações que são escolhidas, em princípio, estruturas que possibilitam a melhor aplicação.

A literatura contém diversos algoritmos de segmentação, entre eles destaca-se o método de segmentação por *Watersheds-IFT*. Nessa técnica a segmentação é realizada a partir de sementes selecionadas pelos usuários, onde cada semente corresponde a um pixel da imagem. O ato de gerar o conjunto certo de sementes ou marcadores na imagem é fundamental para que o usuário obtenha um resultado satisfatório em suas amostras. Mas essa tarefa não é tão simples, para selecionar as sementes corretamente na imagem é necessário conhecer a técnica *Watersheds-IFT*, dificultando assim seu uso para um usuário não familiarizado com a técnica.

A fim de minimizar essa dificuldade no processo de segmentação por *Watersheds-IFT*, o presente trabalho propõe a criação de um método híbrido pelo qual, a técnica de segmentação *Watersheds-IFT* será automatizada, auxiliando o usuário a realizar a difícil tarefa de encontrar o conjunto correto de marcadores, essenciais na obtenção de um bom resultado no processo de

segmentação.

Para realizar esse processo, o método de segmentação *Mean Shift* será utilizado, o qual tem por objetivo reduzir o conjunto de pontos presentes na imagem. Esse conjunto reduzido de pontos serão as sementes utilizadas no algoritmo IFT, facilitando assim a tarefa do usuário. Com base nesse contexto, o principal objetivo deste trabalho é automatizar o método de segmentação por *Watersheds-IFT*, possibilitando ao usuário uma análise mais rápida e precisa.

Após finalizar a implementação, foram realizados testes para avaliar a eficiência desta técnica, considerando os resultados obtidos e a complexidade dos algoritmos analisados.

1.1 Estrutura da dissertação

O capítulo 2 apresenta uma descrição do processo de análise de imagens digitais em microscopia, que inclui as particularidades de uma imagem digital, as etapas do processamento de imagens e a importância da análise quantitativa. No capítulo 3, são descritos o processo de componentização utilizado no método de segmentação híbrido, os padrões de projeto e as ferramentas utilizadas na construção do método híbrido, como por exemplo, a linguagem Java-JAI, a biblioteca Swing e a ferramenta de modelagem unificada (UML) juntamente com suas respectivas aplicações. O capítulo 4 apresenta os métodos de segmentação utilizados no desenvolvimento deste trabalho, *Watersheds-IFT* e *Mean Shift*, juntamente com o processo de implementação do método de segmentação híbrido. A seguir, no capítulo 5, são apresentados os testes realizados com o método de segmentação híbrido e seus respectivos resultados. Por fim, no capítulo 6, são apresentadas as considerações desse trabalho e sugestões para trabalhos futuros.

2 A ANÁLISE DE IMAGENS DIGITAIS EM MICROSCOPIA

Desde sua invenção, há cerca de quatrocentos anos [CLA 2002], o microscópio tem desempenhado um papel fundamental na exploração de amostras orgânicas e inorgânicas, tornando-se uma ferramenta indispensável na pesquisa científica. Ao longo deste período, com a evolução do conhecimento científico básico em física ótica e a expansão dos campos de aplicação de microscopia nas áreas biológicas e de materiais, tanto no campo da microscopia ótica como na microscopia eletrônica foram desenvolvidos instrumentos cada vez mais poderosos. Uma gama de técnicas de contraste, aliadas ao surgimento de software e acessórios para processamento e análise de imagem, tornaram a microscopia uma ferramenta imprescindível na ciência moderna [CLA 2002].

O uso do microscópio ótico ou eletrônico possibilita a percepção de mais detalhes ou primitivas da imagem, obtendo subsídios para interpretar e analisar o conteúdo de forma confiável. No entanto, os microscópios óticos e eletrônicos possuem diferentes formas de obter as informações das imagens.

Na microscopia ótica, a amostra é iluminada com radiação eletromagnética de comprimentos de onda que variam do infravermelho ao ultra violeta próximos. Já na microscopia eletrônica, um feixe de elétrons varre a superfície da amostra. Em ambos os casos, o sinal detectado pode ser proveniente da superfície irradiada. No primeiro caso, a microscopia é denominada de reflexão ou de varredura e no segundo, de transmissão [SIL 2003].

A cada um desses sistemas são acoplados diferentes tipos de sensores ou detectores capazes de coletar um sinal que resultará na formação de uma imagem. No caso da microscopia ótica, o sensor pode ser uma câmera de vídeo ou de TV ou ainda uma câmera CCD (*Charge Coupled Device*).

Entretanto, mesmo utilizando bons equipamentos, deve-se ter cuidado no momento de adquirir imagens, pois vários problemas podem ser evitados como iluminação inadequada, ruído, entre outros. A estação de captura deve estar preparada para gerar imagens com a melhor qualidade possível, evitando que a imagem seja gerada com informações indesejáveis (ruídos) ou que informações importantes sejam perdidas. Quanto melhor a qualidade da imagem adquirida, mais fácil e rápido serão realizados os processamentos de filtragem, de segmentação, de correção de iluminação entre outros [MUR 2001].

Nesse trabalho, para adquirir tais imagens, utilizou-se um microscópio ótico de luz transmi-

tida ajustado com uma objetiva de 40x. Nesses microscópios, a iluminação é feita através da amostra e exige que a mesma seja transparente com espessura suficientemente fina para permitir a passagem da luz.

2.1 As Imagens Digitais e suas Particularidades

Uma imagem digital monocromática é uma função bidimensional de intensidade da luz *f(x,y)* discretizada tanto em coordenadas espaciais quanto em brilho. Nessa imagem, *x* e *y* denotam as coordenadas espaciais e o valor de *f* em qualquer ponto *(x,y)* é proporcional ao brilho da imagem, naquele ponto, conforme a Figura 2.1.

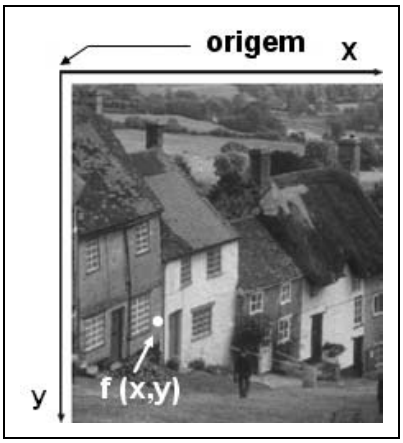

Figura 2.1: Convenção dos eixos para representação de imagens digitais.

Para o processamento computacional, de acordo com Gonzalez [GON 2000a], uma função *f(x,y)* precisa ser digitalizada tanto espacialmente quanto em amplitude. A digitalização das coordenadas espaciais *x,y* é denominada amostragem da imagem e a digitalização da amplitude é chamada quantização em níveis de cinza. Conforme Pavim [PAV 2005], o processo de amostragem e quantização de uma imagem está intimamente ligado à resolução da imagem produzida. Nota-se que quanto maior o número de pontos amostrados na imagem, maior será a resolução espacial desta. Da mesma forma, quanto maior o número de níveis de intensidade luminosa considerados na imagem para representar a tonalidade de um ponto amostrado, maior será a resolução de tonalidades da imagem.

O termo imagem *bitmap* é resultante do agrupamento de vários pontos, com diferentes tonalidades de cor e brilho, podendo chegar a uma imagem policromática. Estes pontos nas imagens digitais são chamados de *pixels*. Conforme Alexandre [LEãO 2005], o termo *pixel* é uma abreviação do inglês *picture element* que significa elemento de figura. É a menor unidade de uma imagem digital, geralmente de forma quadrada, aonde são descritos a cor e o brilho específico de uma célula da imagem, conforme a Figura 2.2. Cada *pixel* é criado quando a cor e brilho de

uma dada posição na matriz 2D são mensurados e armazenados como uma quantidade discreta.

Figura 2.2: Imagem digital e detalhe com *pixels* visíveis. Adaptada de [LEãO 2005]p.59.

A quantidade de *pixels* presentes em uma imagem nos fornece o tamanho da mesma. O tamanho do arquivo digital não se refere apenas a quantidade de detalhes visível - resolução espacial-, mas também ao número de cores presentes - profundidade de cor. Quanto maior a quantidade de *pixels*, maior será o tamanho da imagem e melhor a qualidade de detalhe visível da mesma, conforme Figura 2.3. O tamanho de uma imagem digital é descrito, por exemplo: 640 X 640 *pixels*, 1024 X 728 *pixels* ou outros valores.

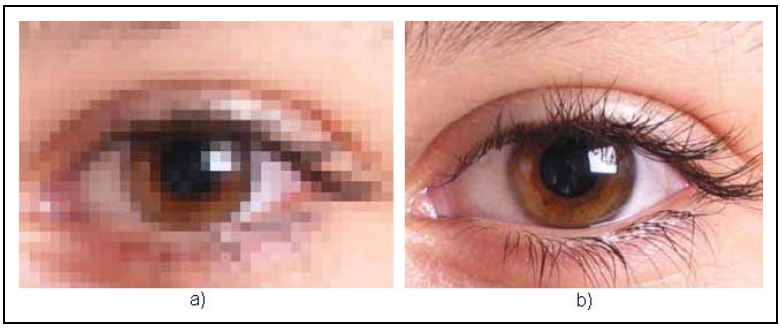

Figura 2.3: a)Imagem de tamanho 40x30 *pixels*. b) Imagem de tamanho 640x480 *pixels*. Adaptada de[LEãO 2005]p.60

Entretanto, uma imagem pode ser representada por um histograma que descreve a fração de área ocupada pelos *pixels* de cada tonalidade. Conforme Israel [ALB 2003], o histograma de uma imagem é muitas vezes denominado como um perfil de intensidade, distribuição de tons de cinza ou, quando o autor refere-se especificamente a um tipo de técnica de aquisição de imagem, pode ser denominado de perfil de elétrons retroespalhados [BOY 1990].

O histograma de uma imagem com *k* níveis de cinza é definido por uma função discreta $p(k) = \frac{n_k}{n}$ em que o argumento *k* representa os níveis de luminância discretos; n_k representa o número total de *pixels* na imagem com intensidade *k*; *n* é o número total de *pixels* da imagem: *n = M x N*, onde M e N representam as dimensões da imagem. De forma simplificada, afirma-se que o histograma de uma imagem representa a contagem dos níveis de cinza da imagem, podendo informar a distribuição dos *pixels* dentro dos *k* níveis possíveis [ALB 2003].

A forma do histograma de uma imagem fornece muitos dados sobre as suas características. Por exemplo, um histograma de uma imagem com pixels concentrados em uma faixa estreita de níveis de cinza indica que a imagem apresenta um baixo contraste. Um histograma de uma imagem com pixels distribuídos ao longo de toda a faixa de níveis de cinza indica que ela apresenta um bom contraste. Um histograma bimodal é aquele que apresenta duas concentrações de pixels, uma em torno de valores escuros e outra em torno de valores claros. Em histogramas bimodais, o objeto se distingue facilmente do fundo [NA 1994].

O histograma de uma imagem digital é uma ferramenta bastante útil uma vez que fornece uma visão estatística sobre a distribuição dos *pixels*, sobre o contraste da imagem e os níveis de iluminação. Além disso, o histograma é bastante utilizado na etapa de segmentação, principalmente em técnicas que utilizam a similaridade entre *pixels*.

A seguir serão descritas as fases de um sistema de processamento de imagens, juntamente com uma breve descrição de cada etapa.

2.2 Etapas do Processamento de Imagem

Conforme Portes [ALB 2003], um sistema de processamento de imagens é constituído de diversas etapas, tais como: aquisição e digitalização da imagem, pré-processamento, segmentação, representação e descrição, reconhecimento e interpretação, como ilustra a Figura 2.4.

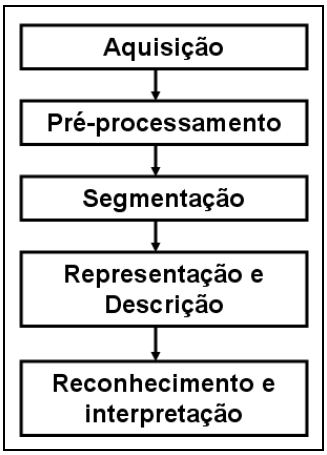

Figura 2.4: Etapas do processamento de imagens.

O primeiro passo do processo é a **aquisição** da imagem. Para fazer isso, necessita-se de dois dispositivos. O primeiro é um dispositivo físico que deve ser sensível ao espectro de energia eletromagnético, como por exemplo luz visível, raios-X ou radiação. O segundo, chamado digitalizador, é um dispositivo que converte o sinal elétrico analógico produzido na saída do sensor em um sinal digital. O sensor poderia ser uma câmera de TV monocromática ou colorida, ou ainda uma câmera de varredura por linha que produza uma única linha de imagem por vez.

Após a aquisição, o próximo passo trata de **pré-processar** a imagem. A função chave no pré-processamento é melhorar a imagem de forma a aumentar as chances para o sucesso dos processos seguintes. Tipicamente, essa fase envolve técnicas para correção de iluminação, remoção de ruído e isolamento de regiões.

O próximo estágio trata da **segmentação**, que divide uma imagem de entrada em partes ou objetos constituintes. A segmentação é considerada, dentre todas as etapas do processamento de imagens, a mais crítica do tratamento de informação. É na etapa de segmentação que são definidas as regiões de interesse para processamento e análise posteriores. Como conseqüência deste fato, quaisquer erros ou distorções presentes nesta fase se refletem nas demais etapas, de forma a produzir ao final do processo resultados não desejados que podem contribuir de forma negativa para a eficiência de todo o processamento.

O processo de **descrição**, também chamado **seleção de características**, procura extrair características que resultem em alguma informação quantitativa de interesse ou que permitam a discriminação entre classes de objetos.

O último estágio envolve reconhecimento e interpretação. **Reconhecimento** é o processo que atribui um rótulo a um objeto, baseado na informação fornecida pelos seus descritores. A **interpretação** envolve a atribuição de significado a um conjunto de objetos reconhecidos, facilitando a análise do usuário.

2.3 Vantagens e Desvantagens no Uso de Técnicas Digitais

No final do século XX, o registro de imagens de microscopia eletrônica em filme foi substituído pelo uso de dispositivos de digitalização dos sinais produzidos pela interação do feixe de elétrons com a amostra [COS 2004].

A substituição do filme fotográfico pelo arquivo digital mostrou-se muito vantajosa e abriu uma série de novas possibilidades. Como arquivo digital, a imagem pode ser visualizada, armazenada, copiada, compartilhada, transmitida e distribuída com rapidez e sem perda de informação. A esse arquivo podem ser associados metadados que descrevem o seu conteúdo como, por exemplo, a natureza da amostra e seu conteúdo de preparação, técnicas e condições de imageamento, ou características morfológicas, estruturais, texturais e físico-químicas dos objetos imageados. Dessa forma, a imagem pode ser incorporada a um banco de dados, de onde será eventualmente recuperada por mecanismos de busca ou, ainda, ter seus metadados cruzados com os de outras imagens em busca de correlações [COS 2004].

Além disso, o uso da imagem digitalizada permitiu aos usuários utilizarem técnicas de pro-

cessamento e análise, a fim de corrigir imperfeições ocorridas no momento da aquisição, auxiliando os usuários a realizar suas análises.

Atualmente, são utilizados equipamentos de aquisição cada vez mais modernos onde as imagens estão sendo adquiridas com ótimas condições de focagem, distingüindo as áreas escuras, que correspondem às regiões de alta densidade atômica, das áreas claras, de baixa densidade atômica. Por outro lado, quando as imagens não são de boa qualidade, seja por problemas de aquisição ou falta de equipamento adequados, muitas informações podem ser perdidas.

Essas imagens , bem como as consideradas “boas”, podem ser melhoradas usando um programa de processamento de imagens, como por exemplo, Adobe Photoshop, ImageTool, CorelDraw, entre outros. Em geral, esses programas apresentam muitas ferramentas que permitem uma melhor visualização das imagens, entretanto, é preciso cuidado ao utilizá-los a fim de não causar transformações incorretas, prejudicando assim a análise dos usuários.

Sendo assim, para garantir a obtenção de resultados que sejam exatos e reprodutíveis, é recomendável que o usuário conheça o significado das operações realizadas antes de aplicá-las à solução de um problema. Mais do que isso, ele precisa compreender que a qualidade da análise não depende apenas de métodos numéricos e computacionais adotados ou da sua implementação, mas também das etapas que precedem a sua aplicação, a saber, a aquisição e o processamento das imagens.

Dessa forma, as técnicas de processamento e análise digitais possuem vantagens e desvantagens em seu uso; algumas delas serão listadas a seguir:

Vantagens
Realiza medidas constantemente e com maior velocidade.
Maior precisão na geração dos resultados.
Permite que o usuário realize medidas impossíveis de se obter manualmente.
Realiza medidas acuradas.
Corrige possíveis erros de aquisição como ruído, iluminação irregular, brilho, entre outros.
Realiza correções nas imagens, permitindo ao usuário uma melhor qualidade durante o processo de análise. Vários são os tipos de correções como filtro, contraste, brilho, entre outros.
Permite aumentar ou diminuir a resolução das imagens, com o intuito de analisar, mais detalhadamente, os objetos presentes na amostra.

Tabela 2.1: Vantagens encontradas em utilizar técnicas de processamento e análise de imagens.

Desvantagens
Sistemas não testados corretamente podem levar os patologistas a realizar análises incorretas, prejudicando assim, o tratamento de seus pacientes.
Tecnologia de alto valor financeiro e de difícil utilização.
Usuário não sabe utilizar corretamente as técnicas de processamento, causando, muitas vezes, perda de informação.

Tabela 2.2: Desvantagens encontradas em utilizar técnicas de processamento e análise de imagens.

Com isso, é preciso ter cuidado ao se utilizar ferramentas de processamento e análise de imagens a fim de não cometer erros que possam prejudicar o resultado. A falta de cautela no uso de técnicas digitais tráz certa insegurança aos analistas no momento de gerarem seus relatórios, os quais contêm uma análise e/ou implicam, até mesmo, numa tomada de decisão. Mas apesar das dificuldades encontradas pelos usuários em utilizar técnicas de processamento de imagens, os métodos computacionais representam um ganho significativo de tempo e precisão em relação aos métodos visuais.

2.4 A importância da Análise Quantitativa

O termo análise está relacionado à parte do tratamento onde é feita uma descrição da informação presente na imagem. Essa parte é chamada de parametrização. É nela que várias medidas quantitativas (parâmetros) são utilizadas para descrever diferentes informações dentro de uma imagem [ALB 2001]. Tratar uma imagem, com o objetivo de extrair informações quantitativas consiste em transformá-la sucessivamente a fim de deixar mais acessível o seu conteúdo informacional.

Algumas aplicações típicas onde é necessário realizar o processo de análise quantitiativa são: a determinação do número de células presentes em um tecido biológico, o cálculo das formas dos contornos de uma célula ou ainda a determinação da distribuição de uma população específica de um conjunto de células. As técnicas dedicadas à análise de imagens podem variar, significativamente, segundo a sua complexidade e a necessidade em tempo de processamento [PRA 2001].

É nesta área que se encontra um nível elevado de complexidade no tratamento da informação. Um exemplo prático é a classificação automática de células anormais dentro de um conjunto de células observadas em microscopia. Esta análise específica demanda soluções dadas pelas técnicas de classificação e pelo reconhecimento de formas. Neste caso, deve-se medir vários parâmetros pertinentes ao problema, como por exemplo, a superfície, a forma de cada célula, sua quantidade, o número de células vizinhas a uma dada célula e a densidade de células em uma certa região.

Em seguida, tem-se a comparação destas medidas com várias classes de células organizadas em uma base de dados, catalogadas anteriormente. Obtém-se então uma classificação das células

com uma dada probabilidade de serem células anormais ou normais. Dessa forma, a análise quantitativa irá relacionar a cada problema uma grandeza (área, volume, entre outras), extraindo informações específicas, fornecendo dados (números), os quais servirão de base para que o analista possa realizar, com maior precisão, seus pareceres finais.

Sendo assim, a análise quantitativa auxilia os usuários a analisarem, de forma rápida e precisa, suas amostras. Além disso, possibilita que um número maior de amostras sejam examinadas em um menor tempo, e que um número maior de pontos sejam amostrados em um mesmo campo de observação.

3 DESENVOLVIMENTO DE SOFTWARE BASEADO EM COMPONENTES PARA A SEGMENTAÇÃO

Nesse capítulo serão apresentados alguns princípios de estruturação de software para a construção de sistemas flexíveis, reusáveis e robustos.

As seções 1.1, 1.2 e 1.3 apresentam, respectivamente, os seguintes conceitos: Desenvolvimento de software baseado em componentes, reusabilidade de componentes por meio de padrões de projeto e ferramentas utilizadas no desenvolvimento do método híbrido (Java, Jai, Swing e UML) que serviram para desenvolver o método de segmentação híbrido.

3.1 O Desenvolvimento Baseado em Componentes

Antes de abordar o desenvolvimento de software baseado em componentes, primeiramente será explicado o que é um componente.

Componente podem ser descritos como partes pré-definidas de software, com interfaces e comportamento delineados, que podem ser usados e reutilizados em diferentes aplicações e tem a finalidade de prover conjuntos de serviços que são acessíveis através de uma interface bem definida.

Deste modo, o desenvolvimento baseado em componentes (DBC) permite que aplicações sejam estruturadas a partir de componentes pré-construídos ao invés de desenvolvê-los desde o início [CRN 2003]. O DBC é um amadurecimento natural da indústria de software baseando-se na força da orientação a objeto (OO). A indústria de software procura com o DBC, maximizar a reutilização e a produtividade do desenvolvimento de aplicações. Segundo Sun Microsystems [Sun 2004], existem várias vantagens em construir um componente, as principais são:

- Reusabilidade: a reutilização da funcionalidade do componente por toda a aplicação;
- Produtividade: com o estabelecimento de uma interface bem definida e a redução da complexidade através do encapsulamento, desenvolver aplicações torna-se mais rápido e simples;
- Facilidade de uso e Aplicação: Através do modelo fornecedor/consumidor (no modelo fornecedor/consumidor os chamados fornecedores devem ser capazes de publicar componentes com suas respectivas especificações em um formato compreensível e em um

lugar onde outras pessoas tenham acesso. Já os consumidores têm que estar aptos a localizar componentes que satisfaçam suas necessidades), desenvolvedores podem rapidamente tornar-se produtivos no DBC, sem um extenso treinamento;

- Mudanças executadas com facilidade e rapidez: O aumento da modularidade e ausência de dependências permitem aos desenvolvedores modificar, adicionar ou substituir componentes tão rapidamente quanto as necessidades de negócio mudam;

- Melhor foco de negócios: níveis mais altos de abstração permitem a desenvolvedores e a gerentes de negócios trabalharem juntos para planejar, projetar e construir a aplicação em termos de negócio em alto nível;

- Proteção dos investimentos: Com a criação de novos padrões de inter-operacionalidade entre os componentes (Java Beans, COM/DCOM e CORBA), desenvolvedores poderão ficar certos de que os componentes baseados nesses padrões além de funcionarem imediatamente, também funcionarão no futuro.

É importante, para esta perspectiva, que já existam componentes de software implementados e que estejam disponíveis para serem selecionados e reutilizados na composição dos sistemas. As atividades essenciais para o desenvolvimento com componentes, segundo [ALA 1997], são: seleção, qualificação, adaptação, composição e atualização.

A Figura 3.1 adaptada de [ALA 1997], ilustra as cinco atividades propostas, onde cada uma delas é brevemente caracterizada a seguir:

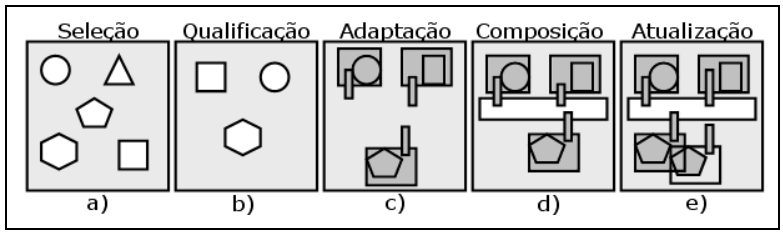

Figura 3.1: Atividades realizadas a fim de efetuar o desenvolvimento de software através de componentes. a) Seleciona componentes, os quais podem ser utilizados em um sistema. b) Ajusta o componente escolhido ao sistema que está em desenvolvimento. c) Adapta os componentes selecionados e qualificados a fim de corrigir possíveis conflitos existentes em um sistema. d) Une componentes através de uma infra-estrutura comum. e) Atualiza versões antigas de componentes ou troca os módulos por um novo.

Seleção: Esta primeira atividade consiste em pesquisar componentes que apresentem potencial para serem empregados no desenvolvimento do sistema. É necessário realizar um processo de investigação das propriedades e da qualidade do componente, seus parâmetros e uma breve descrição do ambiente de execução, gerando, assim, uma relação de componentes candidatos, os

quais serão avaliados na próxima etapa a fim de determinar quais serão utilizados. Caso nenhum candidato a componente tenha sido selecionado é necessário envolver-se no desenvolvimento de componentes.

Qualificação: esta etapa consiste em garantir que o componente candidato execute as funcionalidades necessárias, ajuste-se ao estilo arquitetural definido para o sistema e apresente as qualidades requeridas para a aplicação (desempenho, confiabilidade, usabilidade, etc.) [PRE 2001]. A qualificação geralmente envolve uma análise detalhada de toda a documentação ou especificação disponível, da discussão com os desenvolvedores dos componentes e usuários do sistema e do teste de execução do componente em diferentes cenários [ALA 1997].

Adaptação: Atividade que busca corrigir potenciais fontes de conflito entre os componentes selecionados e qualificados para compor o sistema.

Composição: Nesta etapa é realizada a união dos componentes através de uma infra-estrutura comum. Para que esta união seja consistente, a infra-estrutura deve prover uma ligação dos módulos e um conjunto de convenções conceituais compartilhadas pelos componentes.

Atualização: Como última atividade tem-se a atualização dos componentes, onde versões antigas serão substituídas ou novos componentes, com comportamento e interface similares, serão incluídos.

As cinco atividades apresentadas descrevem possíveis etapas no desenvolvimento de software com componentes. Entretanto, não existe uma ordem para sua realização ou mesmo obrigação de sua realização. As etapas necessárias, bem como a importância e esforços empregados em cada uma, correspondem a uma decisão relacionada ao processo de desenvolvimento com componente a uma aplicação.

Com isso, o DBC mudou o processo da arquitetura de software na medida que algumas partes do sistema puderam ser adquiridas dos fornecedores de componentes. Tem-se agora, dois tipos de processos: o convencional e o baseado em componentes, os quais são ilustrados na Figura 3.2.

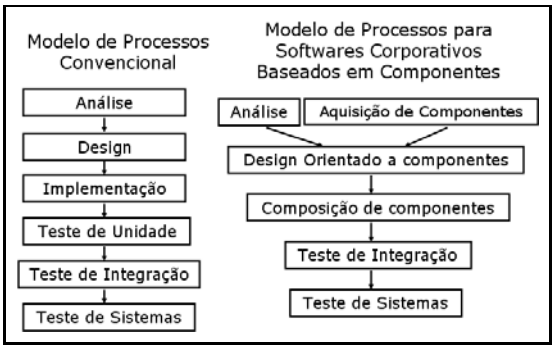

Figura 3.2: Modelos utilizados no processo de construção do software. a) representa o modelo convencional. b) Representa o modelo baseado em componentes que, ao contrário do anterior, realiza a montagem do software mais rápido e com menor custo. Adaptada de [MIC 2001]

A diferença entre esses dois sistemas está em sua montagem. Enquanto o modelo convencional faz uma análise do sistema, levantando os principais requisitos para após iniciar a implementação, o modelo baseado em componentes realiza uma análise dos componentes existentes a fim de encontrar o que melhor se adapta ao seu sistema, tendo um desenvolvimento mais rápido e com menor custo, pois não terá que construir uma aplicação desde o início como o modelo convencional faz.

Também é importante salientar que o atual sucesso da abordagem de desenvolvimento de software baseado em componentes deve-se principalmente ao paradigma da orientação a objetos [VIT 2003]. As linguagens de programação OO reconhecem a necessidade de considerar o comportamento de um sistema como sendo um conjunto de ações inter-relacionadas, assim como a abordagem do DBC, centrando foco nas similaridades e diferenças de comportamento dos objetos no sistema e na captura dos relacionamentos entre esses objetos. Assim, no contexto do paradigma OO, quando um novo comportamento deve ser definido, este é geralmente criado através da reutilização ou da herança de um comportamento já existente no sistema, que é então especializado.

Em muitas situações, essa abordagem é benéfica, no entanto, observa-se que a mesma tem algumas limitações. Por exemplo, quando devemos levar em consideração o aumento da produtividade, a necessidade de que todos os detalhes do comportamento de dado componente do sistema sejam entendidos por um desenvolvedor de aplicações não pode ser considerada muito produtiva [SHA 2004]. Bem como, para sistemas complexos, as hierarquias de comportamento herdado podem se tornar estruturas pesadas, difíceis de serem entendidas e com várias interdependências que as tornam difíceis de ser modificadas. Se considerarmos que as partes reutilizadas são desenvolvidas por terceiros, tem-se então, um ambiente com grandes problemas de produtividade. A razão é simples: dificilmente entenderemos, de maneira detalhada, o comportamento de cada uma das partes.

Para superar essas dificuldades, é preciso que o comportamento e as interfaces desses componentes sejam claramente especificadas. Além disso, é preciso que sejam fornecidos mecanismos que permitam a conexão dos componentes de forma flexível e que cada um destes seja substituído por outro similar a qualquer momento, sem a necessidade de modificações internas na estrutura dos componentes relacionados.

Sendo assim, nota-se que o desenvolvimento de software baseado em componentes representa um dos mais importantes fenômenos da década na área de desenvolvimento de software, uma vez que tem aumentado a necessidade de construir softwares eficientes em um curto intervalo de tempo e com um baixo custo.

3.2 Reutilizando Componentes por meio de Padrões de Projeto

O desenvolvimento de novos software muitas vezes é complexo e lento. Para auxiliar e solucionar estas questões existem muitas pesquisas e metodologias voltadas para o reuso de software.

Segundo Buschman *et al.* [BUS 1996], quando especialistas trabalham na solução de um determinado problema, é comum comparar o projeto atual com projetos desenvolvidos anteriormente de modo a recorrer a uma solução já usada em algum projeto anterior. Procurar e investir em uma solução completamente nova não é uma prática comum, pois a experiência os leva a comparar problemas na tentativa de se utilizar a essência de soluções já aplicadas. Assim, ao confrontar-se com novos problemas, é comum recuperar e lembrar soluções antigas pensando em pares "problema-solução" [BUS 1996].

Esses pares problema-solução podem ser categorizados em famílias de problemas e soluções similares, sendo que cada uma dessas famílias exibe um padrão de ocorrência tanto nos problemas como nas soluções. Assim, quando uma mesma solução pode ser utilizada para uma classe comum de problemas, pode-se afirmar que está usando-se um *padrão*.

Christopher Alexander introduziu a idéia do uso de padrões na área de arquitetura na década de 1970, mas o conceito foi estendido a outras áreas, como na computação. De acordo com Alexander [ALE 1977], "um padrão descreve um problema que ocorre inúmeras vezes em determinado contexto, e descreve ainda a solução para esse problema, de modo que essa solução possa ser utilizada sistematicamente em distintas situações". Em outras palavras, um padrão de projeto pode ser descrito como uma solução que já foi testada para um problema. Desse modo, padrões geralmente descrevem uma solução ou uma instância da solução que foi utilizada para resolver um problema específico.

A análise, o projeto e a codificação de programas também apresentam características que se repetem com certa freqüência no desenvolvimento de *software*. Assim, as soluções encontradas para problemas similares são chamadas de *padrões*, visto que são bases de comparação ou servem de modelo para outros sistemas [MAR 2004].

A principal razão para se utilizar padrões de projeto no desenvolvimento de *software* baseia-se no fato de que a fase de projeto é mais difícil e a que consome mais tempo de todas as fases do desenvolvimento de um programa [SAN 1997]. Os padrões permitem que essa fase seja otimizada, pois pode-se aproveitar as experiências que são comprovadamente eficientes, reduzindo o tempo gasto na busca de uma solução.

Um dos aspectos considerados cruciais para a produção de programas de alta qualidade baseia-se no grau de reuso dos componentes [PRE 1995]. Os componentes reusados tem um impacto positivo no momento de corrigir o programa porque as partes que já foram usadas e testadas contêm menos erros do que as novas funções desenvolvidas. Assim, ao se criar componentes baseados em padrões de projeto comprovadamente eficientes, aumenta-se a reusabilidade dos programas e componentes produzidos.

Outro benefício do uso de padrões de projeto é que eles facilitam o desenvolvimento de

classes altamente coesas [GAM 2000], ou seja, nas quais todos os métodos estão fortemente relacionados uns com os outros. Isso ocorre porque todos os métodos de uma classe contribuem para a solução do problema, garantindo também o mínimo de acoplamento possível entre duas classes.

De acordo com as características apresentadas, pode-se sumarizar, pelo menos, quatro vantagens do uso de padrões de projeto para o desenvolvimento de sistemas:

1. Formam um vocabulário comum que permite uma melhor comunicação entre os desenvolvedores, uma documentação mais completa e uma melhor exploração das alternativas de projeto. Reduz a complexidade do sistema através da definição de abstrações que estão acima das classes e instâncias;

2. Constituem uma base de experiências reutilizáveis para a construção de software. Funcionam como peças na construção de projetos de software mais complexos; podem ser considerados como micro-arquiteturas que contribuem para arquitetura geral do sistema;

3. Reduzem o tempo de aprendizado de uma determinada biblioteca de classes. Isto é fundamental para o aprendizado dos desenvolvedores novatos;

4. Quanto mais cedo são usados, menor será o retrabalho em etapas mais avançadas do projeto.

Os programas para processamento de imagens são conhecidos por serem difíceis de implementar. Dificuldades surgem tanto no projeto quanto na codificação dos programas. Essas dificuldades surgem em razão da diferença de conhecimento que existe entre os desenvolvedores e os usuários, que pode causar falhas na modelagem e fazer com que os algoritmos sejam usados de forma inadequada devido à falta de especificação [D'OR 2001]. Assim, considerando as vantagens do uso de padrões de projeto enumeradas anteriormente, a utilização dos padrões para o desenvolvimento de programas para o processamento de imagens pode suprir o problema da difícil codificação desse tipo de *software*, pois poderá haver o reuso de soluções comprovadamente bem sucedidas.

De acordo com Gamma *et al* [GAM 2000], um padrão tem geralmente quatro elementos essenciais:

1. **Nome:** Define, em uma ou duas palavras, uma referência que pode ser usada para descrever o padrão. Cada padrão deve ter um nome bastante claro e intuitivo, a fim de facilitar a sua identificação.

2. **Problema:** Descreve em quais situações o padrão deve ser aplicado, explicando seu problema e contexto. Gama *et al* [GAM 2000] salientam ainda que algumas vezes o problema pode incluir uma lista de condições que devem ser satisfeitas para que o uso do padrão seja justificado.

3. **Solução:** "Descreve os elementos que compõem o projeto, seus relacionamentos, suas responsabilidades e colaborações". Convém salientar que a solução não descreve um projeto concreto para uma determinada situação, e sim uma descrição abstrata de um problema que pode ser aplicada em situações diferentes.

4. **Conseqüências:** Representam os resultados e as análises das vantagens e desvantagens da aplicação do padrão. Podem ser avaliados os impactos do uso do padrão sobre a flexibilidade, a extensibilidade ou a portabilidade de um sistema.

De acordo com Gamma e *http://hillside.net*[1] [GAM 2000], os quais são considerados a principal referência na área de padrões de projeto, os padrões são categorizados em três grupos:

- **Criacionais:** estão relacionados à instanciação de objetos;
- **Estruturais:** referem-se à composição de classes ou objetos;
- **Comportamentais:** caracterizam formas de interação entre classes e objetos.

Existem outras maneiras de se categorizar os padrões, como, por exemplo, a forma explorada por Buschmann *et al.* [BUS 1996]. Porém, os padrões sumarizados pela GOF (*Gang Of Four*) são os mais populares, pois representam soluções para problemas comuns no desenvolvimento de programas.

Sendo assim, os padrões de projeto, buscam isolar no processo de desenvolvimento de *software* as partes que são estáveis daquelas que são alteradas com freqüência. Dessa forma, o resultado esperado são aplicações mais fáceis de manter, compreender e reutilizar.

3.3 Ferramentas Utilizadas no Desenvolvimento do Método Híbrido

O método híbrido foi implementado utilizando a linguagem de programação Java juntamente com sua API Java Advanced Imaging (JAI). Para construir a interface e os diagramas foi utilizada, respectivamente, a biblioteca `Swing` e a linguagem UML.

3.3.1 A Linguagem de Programação Java

A tecnologia Java foi criada como uma ferramenta de programação em computação, parte de um pequeno trabalho anônimo e secreto chamado "the Green Project" da Sun Microsystems em 1991. Nos primeiros 18 meses de trabalho a equipe "Green Team" chegou à sua primeira conclusão afirmando que uma tendência significativa seria a convergência de computadores e de dispositivos controlados digitalmente. Essa conclusão resultou na linguagem de programação não atrelada a dispositivos, apelidada de "Oak" [SUN 2005].

Para demonstrar como essa nova linguagem poderia impulsionar o futuro dos dispositivos digitais, a equipe Green Team desenvolveu um controlador portátil para sistemas de entretenimento doméstico, voltado ao setor de televisão digital. A idéia estava muito à frente de seu

[1] Repositório de Padrões, os quais estão disponíveis na página http://hillside.net

tempo e o setor de TV digital não estava preparado para o incrível avanço oferecido pela tecnologia Java.

Mas o "boom" da Internet aconteceu, e, rapidamente, uma grande rede interativa estava se estabelecendo. Era esse tipo de rede interativa que a equipe estava tentando vender para as empresas de TV a cabo. E, da noite para o dia, não era mais necessário construir a infra-estrutura para a rede, em um golpe de sorte, ela simplesmente está lá. Gosling foi incumbido de adaptar o Oak para a Internet e, em janeiro 1995, foi lançada uma nova versão do Oak que foi rebatizada para Java [SUN 2005a].

A tecnologia Java tinha sido projetada para se mover através de redes de dispositivos heterogêneos, redes como a Internet. Agora aplicações poderiam ser executadas dentro dos Browsers nos Applets Java e tudo seria disponibilizado pela Internet instantaneamente.

Segundo descrito por Almeida [ALM 1998], a linguagem Java é simples, orientada a objetos, interpretada, robusta, segura, portável, além de ser dinâmica. A combinação dessas características torna Java uma poderosa linguagem de programação, sendo de extrema utilidade para os desenvolvedores de software. A seguir, será dada uma breve descrição sobre algumas das características citadas por [ALM 1998].

- Simplicidade: Java é simples para o uso devido a sua similaridade com a linguagem C e por eliminar componentes causadores de erros e problemas de memória;

- Robustez: Java pode ser considerada robusta pelo fato que, o uso de ponteiros e a administração de memória deixam de ser de responsabilidade do programador;

- Linguagem Interpretada: por ser uma linguagem interpretada, Java permite ser compilada para uma determinada máquina virtual e mesmo assim, o código resultante poder ser utilizado em qualquer plataforma de *hardware* que possua um interpretador da linguagem;

- Arquitetura Neutra: por possuir uma arquitetura neutra, as aplicações podem ser portadas para várias plataformas. Essa característica é alcançada, pois as aplicações são escritas e compiladas em *byte code* por uma máquina virtual Java, a qual emula um processador;

- Segurança: Java é segura por não permitir acesso direto à memória, por disponibilizar o tratamento de exceções e por ser fortemente tipada.

Além dessas vantagens, Java é uma linguagem de múltiplos processos (*multithreaded*). Ela provê para múltiplas *threads* de execução, também chamadas de processos leves (*lightweight processes*) que podem tratar diferentes tarefas. O suporte nativo para implementação de múltiplos processos evita a necessidade de se usarem bibliotecas, que, por não estarem disponíveis em todos os ambientes, dificultam a portabilidade dos programas [ALM 1998].

Atualmente, a tecnologia Java encontra-se em redes e dispositivos que vão desde a Internet e supercomputadores científicos a laptops e telefones celulares, de simuladores de mercado da Wall Street a dispositivos para jogos e cartões de crédito simplesmente em todo lugar.

Sendo assim, desde seu lançamento, em maio de 1995, a plataforma Java foi adotada mais rapidamente do que qualquer outra linguagem de programação [HOR 2001]. Ela continua crescendo até hoje e é um padrão para o mercado, oferecendo qualidade, performance e segurança aos seus usuários.

3.3.2 Java Advanced Imaging (JAI)

A API (Application Programming Interface) JAI, como é comumente chamada, é formada por um conjunto de classes especialmente desenvolvidas para o desenvolvimento de aplicações em processamento de imagens [Sun 2004a].

Ela oferece suporte e formatos complexos de imagens, incluindo imagens com mais de três dimensões, como, por exemplo, operadores pontuais, de área, de escala e estatísticos.

Apesar de a JAI dar suporte a um grande número de operações para o processamento de imagens, ela foi projetada para permitir que os programadores estendam as classes existentes de maneira a fornecer as operações a uma determinada aplicação [Sun 2004a].

Uma das grandes vantagens ao se usar a JAI está no fato que ela permite processar de maneira idêntica tanto um arquivo de uma imagem quanto um objeto de imagem transmitido pela rede, ou ainda um fluxo de dados em tempo real. Isso é possível devido ao encapsulamento dos formatos dos dados da imagem, juntamente com as invocações dos métodos remotos em um objeto de dados de imagem reusável [Sun 2002]. A JAI permite ainda acesso paginado a imagens grandes. Por exemplo, se uma imagem é maior que a memória principal do computador, ela pode ser dividida em partes menores para fins de manipulação.

Como vantagem adicional, aplicações escritas em Java podem ser executadas em praticamente qualquer plataforma, devido ao fato de Java ser executada através de uma máquina virtual [Sun 2004b]. Conseqüentemente, aplicações que utilizam JAI também podem ser executadas em qualquer plataforma.

Sendo assim, esses fatores tornam a combinação da linguagem Java com a biblioteca JAI uma opção atraente quando comparada a soluções comerciais de ambientes de desenvolvimento para processamento de imagens.

3.3.3 A Biblioteca *Swing*

Nas primeiras versões do Java, a única forma de fazer programas gráficos era através da AWT (Abstract Window Toolkit- ferramentas de janelas abstratas), uma biblioteca de “baixo-nível” que dependia do código nativo da plataforma onde rodava. Ela trazia alguns problemas de compatibilidade entre as plataformas, fazendo com que o programa nem sempre tivesse a aparência desejada em todos os sistemas operacionais, sendo, também mais difícil de ser usada [Wik 2006].

Para suprir as necessidades cada vez mais freqüentes em obter uma API estável e fácil de usar, o *Swing* foi criado como uma extensão do Java a partir da versão 1.2. A API *Swing* fornece componentes de mais alto nível, possibilitando assim uma melhor compatibilidade entre os vários sistemas onde Java é executado.

Ao contrário da AWT, a biblioteca *Swing* não contém uma única linha de código nativo, e permite que as aplicações tenham diferentes tipos de visuais (skins), os chamados “Look and Feel”. Já com AWT isso não é possível, tendo todos os programas a aparência da plataforma onde estão instalados [GUJ 2005].

Além disso, a biblioteca *Swing* possui outras vantagens. Há várias opções extras disponíveis, tais como áreas de texto que nativamente podem mostrar conteúdo RTF ou HTML, botões com suporte a imagens, sliders[2], selecionadores de cores etc. É também possível alterar o tipo de borda para a maior parte dos componentes, todos podem ter imagens associadas e é possível até ter controle de como são desenhados os mínimos detalhes de apresentação [GUJ 2005].

Sendo assim, *Swing* é uma API Java para interfaces gráficas. Ela é compatível com a API AWT, mas trabalha de uma maneira totalmente diferente. A API *Swing* procura renderizar ou desenhar por contra própria todos os componentes, ao invés de delegar essa tarefa ao sistema operacional, como a maioria das outras APIs de interface gráfica trabalham, resolvendo o problema de falta de portabilidade existente na antiga AWT.

3.3.4 A Unified Modeling Language (UML)

UML significa Unified Modeling Language (linguagem de modelagem unificada). Ela deriva principalmente da unificação das linguagens de modelização de três métodos: o “OMT” de Rumbauch, o “método de Booch” e os “casos de uso” de Jacobson. Pode-se citar outros métodos importantes como Fusion[3], Shlaer-Mellor[3] e Coad-Yourdon[3]. Todos eram métodos completos, destacavam-se em certos pontos, porém tinham suas limitações. O método Booch destacava-se durante as fases de projeto e construção de sistemas, o OOSE[4] fornecia excelente suporte para captura de requisitos, a análise e o projeto em alto nível; o OMT-2[4] era mais útil com a análise e sistemas de informações com uso de dados [BOO 2000]. Na Figura a seguir é possível visualizar a união das linguagens, as quais compõe a UML.

[2] A interação do usuário é feita por meio do deslocamento de uma “seta” que aponta para o valor escolhido e que esse valor é exibido em uma espécie de régua. Mais informações no site www.apl.jhu.edu/ hall/java/ Swing-Tutorial/Swing-Tutorial-JSlider.html

[3] Técnicas de modelagem criadas no final da década de 80 e início da década de 90. Conforme David [DAV 2001] a linguagem Fusion utiliza uma abordagem dirigida a dado. Nessa linguagem, o desenvolvimento de uma aplicação passa pelas etapas de análise, projeto e implementação. Já a técnica Shlaer-Mellor utiliza a modelagem de dados com um enfoque estruturado bottom-up. O método Coad/Yourdon divide a análise orientada a objetos em classes e objetos, relacionando-os por meio de conceitos de agregação, generalização/especialização, associações entre outros.

[4] Conforme Admilson [NOG 2005], OOSE e OMT-2 significam respectivamente: Object Oriented Software Engineering e Object Modeling Technique

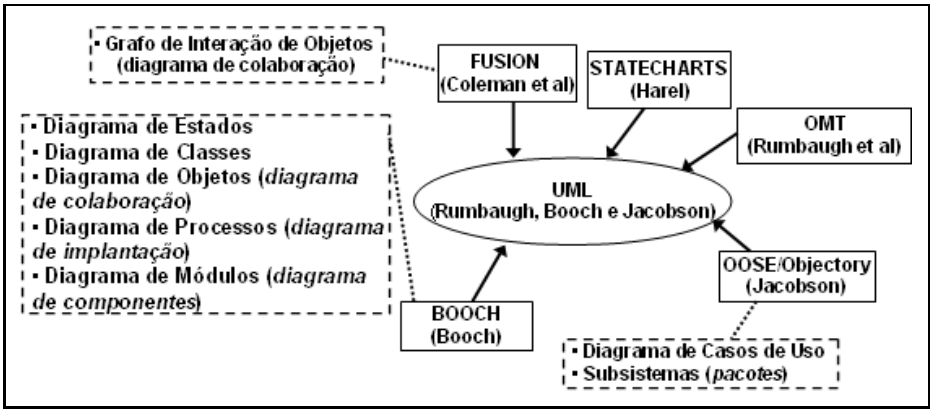

Figura 3.3: A figura ilustra a união das linguagens OO, formando uma única linguagem padrão: a UML. Adaptada de [BOA 2006]

Com isso, os criadores da UML procuraram desenvolver uma linguagem unificada padrão que pudesse ser de fácil entendimento a todos. Preocuparam-se em deixá-la aberta aos desenvolvedores, onde os mesmos pudessem criar seu próprio método de trabalho.

Como todas as linguagens, a linguagem UML é usada para transmitir e receber de outros algumas informações. Nesse caso, as informações são definições detalhadas de como um sistema deve ser implementado, a fim de realizar o que o usuário deseja, gerando uma documentação do sistema.

Portanto, é fundamental conhecer a linguagem para entender os modelos que outras pessoas desenvolveram com muito esforço ou para explicar a outras pessoas (por exemplo programadores) como um sistema deve ser implementado. Cada conceito a ser modelado (subsistema, classe, relações, etc.) tem uma representação gráfica específica na linguagem. É muito importante sempre respeitar as convenções definidas na UML para que outras pessoas possam entender os diagramas gerados.

Sendo assim, a UML pode ser usada para modelar todas às fases de um sistema, desde os primeiros contatos até a geração do código. É aplicada em qualquer tipo de sistemas em termos de diagramas de orientação a objetos, mas também pode ser utilizada em sistemas mecânicos, de engenharia em geral, podendo também ajudar na organização dos processos de uma empresa.

3.4 Aplicação das Ferramentas no Sistema

Nessa seção será apresentada a aplicação de cada ferramenta no processo de desenvolvimento do método híbrido.

3.4.1 A Linguagem Java

Atualmente, temos um grande número de linguagens de programação disponíveis no mercado. Escolher uma linguagem a fim de desenvolver um projeto depende muito das características (segurança, portabilidade, velocidade de execução, etc.) que esse projeto deve conter, a fim de satisfazer as necessidades de seus clientes.

A linguagem de programação escolhida para desenvolver esse trabalho foi a linguagem Java. Essa escolha ocorreu devido às três características presentes na linguagem, as quais são importantes no processo de desenvolvimento do método híbrido. A seguir será apresentada cada uma dessas características.

Primeiramente, a linguagem Java foi utilizada para poder realizar o processo de reuso uma vez que, os componentes *Watersheds-IFT, Mean Shift* e *Save* estavam implementados nessa linguagem. A reutilização de componentes aumentou a modularidade do sistema, permitindo que os módulos utilizados fossem agregados rapidamente ao software. Se essa característica não fosse utilizada, o método híbrido levaria muito tempo para ser concluído, tornando a pesquisa lenta e cara.

A portabilidade é a segunda característica importante presente na linguagem Java. Ela permite que o componente híbrido seja melhor utilizado, pois, através dela, o módulo híbrido pode ser acoplado em qualquer sistema, independente da plataforma que o usuário estiver utilizando.

E finalmente, a terceira característica importante a ser citada é que a linguagem Java possui junto a si APIs (Interface de Programação de Aplicações). Nessas bibliotecas está um conjunto grande de classes organizadas em pacotes, onde cada um desses pacotes traz classes com funcionalidades básicas e vital para um determinado ramo da programação java. Nesse trabalho utilizaram-se várias APIs: `java.util` (oferece classes com funcionalidades diversas, como por exemplo, data corrente, números aleatórios etc), `java.awt` (classes para a criação de objetos de interface gráfica e para manipulação de eventos), `java.io` (mantém classes usadas para controle de entrada e saída em arquivos e em dispositivos), entre outras.

3.4.2 A API JAI

A biblioteca JAI é um pacote Java que possui classes e métodos para a construção dos principais algoritmos de processamento digital de imagens. A sua utilização permite ao desenvolvedor utilizar o mínimo de programação, necessitando apenas que o programador faça a sua chamada através de uma linha de comando.

Essa biblioteca permite a criação de aplicações mais complexas e com performance melhor. Além disso, ajuda os desenvolvedores a competirem nos ambientes de rede heterogêneos e altamente distribuídos, simplificando a criação de aplicações de imagens com suporte a vários sistemas, de clientes magros até poderosas estações de trabalho.

Contudo, a API JAI foi utilizada nesse trabalho apenas para realizar operações de entrada e saída, visto que, na implementação do método híbrido não foi necessário chamar nenhuma de suas operações. É importante frisar também que os métodos *Watersheds-IFT* e *Mean Shift* não

fazem parte dessa API.

3.4.3 A Biblioteca *Swing*

A biblioteca *Swing* foi utilizada nesse trabalho com o propósito de construir uma interface gráfica para o sistema. No processo de construção da interface não foi aplicada nenhuma metodologia a fim de facilitar a sua utilização por parte do usuário, pois esse não é o foco do trabalho.

Após o processo de desenvolvimento do método híbrido, foi constatado que sua utilização automática, isto é, sem a influência do usuário, não gerava resultados satisfatórios para que os usuários pudessem realizar suas análises. Sendo assim, a construção da interface tornou-se necessária para que o usuário pudesse auxiliar no processo de segmentação híbrida. Esse auxílio acontece quando é permitido ao usuário, por meio da interface gráfica, adicionar ou remover sementes na imagem, a fim de obter um melhor resultado ao final do processo. A interface implementada apresenta requisitos mínimos porque o nosso foco é o método e não a interface humano computador (IHC).

Dessa forma, a interface apresenta componentes gráficos básicos, que permitem uma mínima interação com o usuário. Nesses componentes estão incluídos os botões e as janelas, as quais permitem ao usuário acompanhar o procedimento híbrido. Isso pode ser melhor visualizado na Figura 3.4.

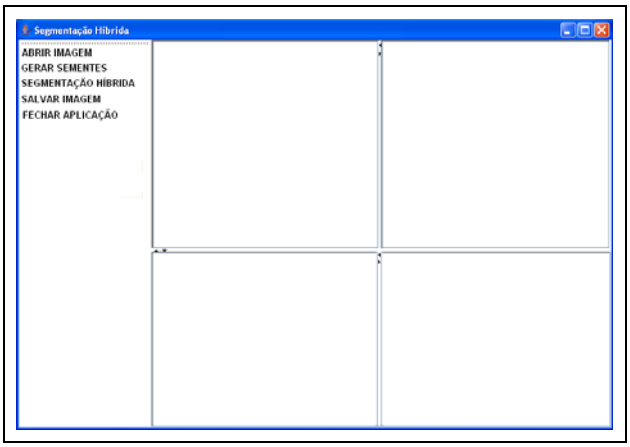

Figura 3.4: Interface construída no sistema para realizar o processo de segmentação híbrida.

A interface está dividida em cinco partes: uma pequena parte é destinada para alojar os botões (Abrir imagem, Gerar Sementes, Segmentação híbrida, Salvar Imagem e Fechar Aplicação), as outras partes são destinadas a visualização das imagens: imagem original, imagem resultante da segmentação *Mean Shift*, imagem marcador e imagem resultado.

Para implementar os botões contidos na interface foi utilizado o comando “JButton” o qual está inserido na API *Swing*. Dentro da classe híbridoGUI (classe na qual a interface está im-

plementada) há o método construtor, que mostra os botões na interface, bem como o comando "Listen", o qual executa a ação implementada em cada botão. Por exemplo, o botão "Gerar Sementes" possui, em sua classe, o método construtor "ListenSegment" e o método "Execute-Action", o qual implementa a ação a ser executada quando o usuário clica no botão "Gerar Sementes". As ações executadas por esse botão são:

- Executa o método de segmentação *Mean Shift*, o qual está representado;
- Atualiza a imagem de marcadores na tela; e
- Habilita o botão Segmentação híbrida da aplicação.

Essas três ações podem ser visualizadas nas linhas 10,11 e 12 da listagem abaixo.

Código 1: Classe responsável por instanciar o botão Segmentar Imagem e implementar a ação que deve ser executada quando o botão for pressionado

```
01.  /*Algoritmo que implementa o método construtor
02.  public ListenSegment(String text){
03.    super(text);
04.    this.setBorderPainted(true);
05.  }
06.  /*Algoritmo que implementa a ação que o botão irá realizar
07.  /*quando pressionado
08.  public void ExecuteAction(){
09.  marcadores = JAI.create("fileload", path);
10.  segmentMS(path);
11.  showImageMarcadores(marcadores);
12  button5.setEnabled(true);
13.  }
```

Sendo assim, observa-se que a biblioteca *Swing* é mais avançada que a AWT, pois com ela é possível implementar todos os controles visuais em Java, utilizando apenas as primitivas gráficas da plataforma. Também permite ao programador especificar uma aparência e um comportamento diferente para cada plataforma, ou uma aparência e um comportamento uniforme entre todas as plataformas, oferecendo um maior nível de portabilidade e flexibilidade aos seus usuários.

3.4.4 A Modelagem do Método de Segmentação Híbrido

Realizar a modelagem de um sistema é tão essencial quanto indispensável quando se deseja construir um software. Através dela, é possível projetar cada fase do sistema, desde os primeiros contatos até a geração do código, facilitando a visualização do projeto e posteriormente sua implementação.

No corrente trabalho, a técnica de modelagem foi a primeira atividade a ser realizada no desenvolvimento do método híbrido. Com ela, visualizou-se os relacionamentos das classes e toda a estrutura desse método, ficando muito mais fácil implementar e compreender o software.

A modelagem do sistema foi realizada utilizando a ferramenta Enterprise Arquitech (EA), a qual utiliza a linguagem UML. Com essa ferramenta construíram-se dois diagramas: o de classe e o de atividade. Os diagramas de classe são importantes para a visualização, a especificação e a documentação de modelos estruturais [BOO 2000]. Por exemplo, ao construir uma casa, é necessário analisar a construção de blocos básicos como, paredes, pisos, janelas, vigas entre outros. Esses itens não podem ser considerados independentes, pois é necessário que eles estejam interagindo, a fim de determinar a arquitetura final da casa. Já a planta apresentada no final é a apresentação gráfica desses itens e seus relacionamentos. A construção de um software tem muitas dessas características e através do diagrama de classe será possível visualizar os relacionamentos entre as classes e especificar os detalhes da construção, conforme mostra a Figura 3.5.

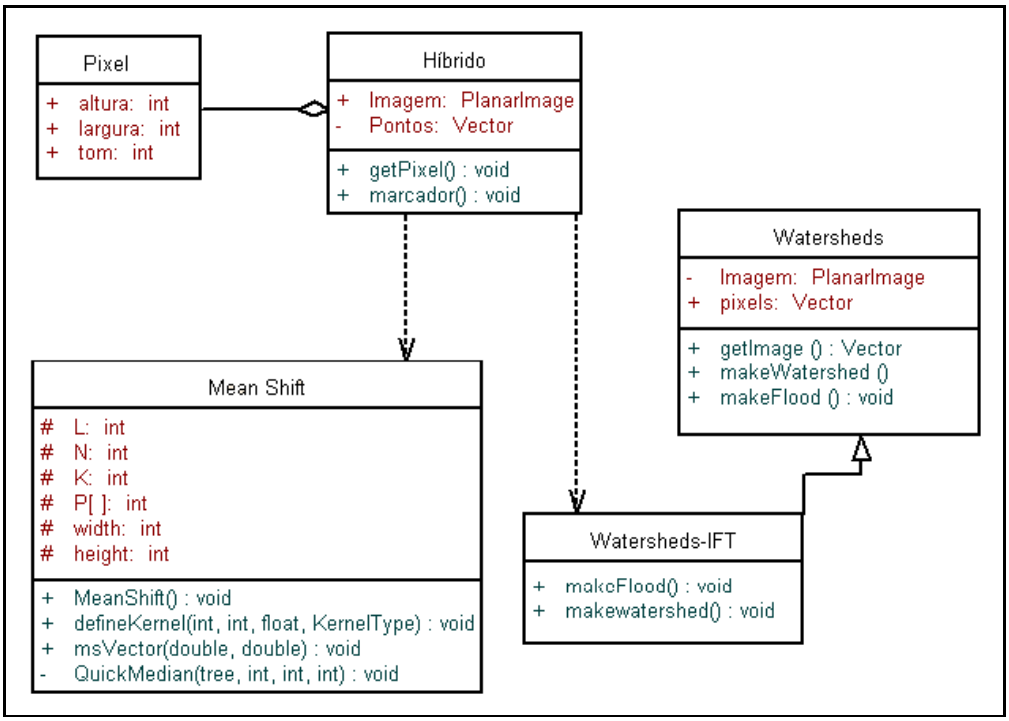

Figura 3.5: Diagrama de classe do sistema. Nele é possível visualizar os relacionamentos existentes entre classes (Pixel, Híbrido, *Mean Shift*, *Watersheds-IFT* e *Watersheds*), métodos(getImage(), makeFlood(), getPixel(), etc.) e atributos (altura, largura, tom entre outros).

Através da Figura 3.5, pode-se extrair diversas informações, as quais são importantes saber antes de o sistema começar a ser implementado, a fim de que se possa corrigir possíveis erros. A seguir são listadas as informações que foram extraídas do diagrama de classe mostrado acima.

- Classe Pixel: esta classe fornece os métodos e os atributos necessários para o estabelecimento dos valores e das relações de vizinhança existentes entre os pixels presentes nas imagens em níveis de cinza.

- Classe Híbrido: esta classe implementa os métodos necessários a fim de realizar o processo de segmentação híbrida.

- Classe *Mean Shift*: esta classe implementa os métodos necessários a fim de executar o algoritmo *Mean Shift*, sendo também utilizada pela classe híbrido, a qual necessita de seus métodos para poder executar suas funções.

- Classe *Watershed-IFT*: esta classe implementa os métodos necessários para executar a transformada *watershed* por operadores IFT (Image Foresting Transform). Também é utilizada pela classe híbrida, a qual necessita de seus métodos para ser funcional.

- Classe *Watershed*: é uma classe abstrata que fornece meios para que as diferentes abordagens de segmentação de imagens por watersheds sejam implementadas, como watershed por imersão, por distância topográfica e por operadores IFT. Esse trabalho utiliza apenas a abordagem *watersheds-IFT*;

Sendo assim, o diagrama de classes permite encontrar facilmente informações sobre métodos, atributos e nomes das funções auxiliando o processo de implementação do método híbrido.

Já os diagramas de atividades são essencialmente gráficos de fluxo, os quais mostram o fluxo de controle de uma atividade para outra em um único processo. Uma atividade é essencialmente parte de um fluxo, que interage com uma outra atividade, podendo ser paralela a ela ou não. A execução paralela é representada pelos ícones Forquilha/Esperar, e para as atividades executadas em paralelo, não é importante a ordem na qual elas se executam (elas podem ser executadas ao mesmo tempo ou uma após a outra). A seguir será exposto o diagrama de atividade do sistema.

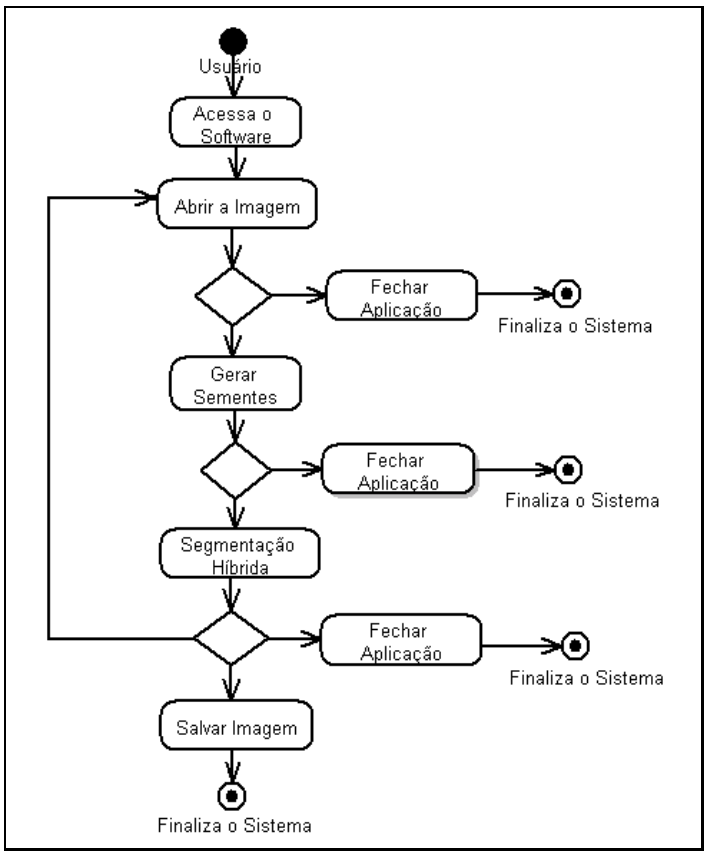

Figura 3.6: O diagrama de atividade mostra o caminho que o usuário deve percorrer a fim de executar o processo de segmentação híbrida.

Através do diagrama de atividade é possível visualizar os passos que o usuário deve seguir, a fim de desencadear uma determinada ação. Por exemplo, para poder realizar a ação de “Gerar Sementes”, tem-se que, primeiro, executar a ação “Abrir a Imagem”, uma vez que o botão gerar sementes não é habilitado, não permitindo que o usuário pratique a ação “Gerar Sementes”. Com isso, o diagrama de atividade nos mostra todas as ações que o usuário deve seguir para poder utilizar adequadamente o sistema.

3.5 Componentes Utilizados na Implementação do Método de Segmentação Hibrido

Os módulos que fazem parte da construção do método de segmentação híbrida são os componentes de segmentação *Watersheds-IFT* e *Mean Shift* juntamente com o componente *Save*. Com essa abordagem, a aplicação (método híbrido) constituiu-se a partir de um conjunto de módulos (componentes) interligados.

A existência desses componentes permite que o processo de construção do método híbrido não começasse a ser construído desde o início, proporcionando uma melhor produtividade e

uma melhor qualidade, haja visto que, o software reutilizável demanda mais testes e garantia de qualidade, simplesmente porque as conseqüências de um erro são mais sérias e o uso contínuo ocasiona uma maior probabilidade de detecção de erros.

Os componentes *Watersheds-IFT* e *Mean Shift* estão sendo reutilizados porque estão implementados na mesma linguagem de programação em que o componente híbrido está sendo desenvolvido, a linguagem Java. Essa linguagem dá suporte a fatores de qualidade (confiabilidade, independência de plataforma e portabilidade) como facilidade de reuso e manutenção, incluindo adaptabilidade.

Sendo assim, a utilização desses componentes, diminui significativamente o tempo de implementação da nova aplicação - a Segmentação Híbrida, bem como a quantidade de código desenvolvido, na medida em que parte das responsabilidades ficou atribuída aos artefatos de software reutilizados.

3.6 A Reusabilidade do Componente Híbrido

O desenvolvimento de software orientado a componentes é uma abordagem que promove o reuso de artefatos de alta granularidade, com a capacidade de promover reuso de projeto e código. A adoção destas abordagens no desenvolvimento de software - em conjunto ou isoladamente - pode diminuir significativamente o tempo para o desenvolvimento de novas aplicações, bem como a quantidade de código a desenvolver, pois parte das responsabilidades das novas aplicações são atribuídas aos artefatos de software reutilizados.

Durante o desenvolvimento do método híbrido, à medida que a modelagem foi sendo desenvolvida, o aspecto de reuso foi sendo juntamente pensado e elaborado. Para que o componente híbrido se tornasse reutilizável, foi necessário desenvolver uma classe (reuso), a qual contém apenas os caminhos necessários para que outros desenvolvedores possam agregar esse componente a diferentes sistemas, sem ter que conhecer a parte interna do código.

Dentro da classe “reuso” foram implementados três passos necessários para o processo de reutilização do componente. No primeiro passo, a imagem de entrada é recebida (linha 06), no segundo, foi chamado o comando `instanciacomponente` (linha 11) que realiza o processo de segmentação híbrida e, por último, passou-se a imagem resultado para o usuário (linha 13). Para realizar o primeiro e o último passo foi utilizada a JAI. A seguir é exposto o código que realiza esse procedimento.

Código 2: Algoritmo que possibilita o método híbrido tornar-se um componente reutilizável

```
01.  /*Algoritmo implementado dentro do método híbrido o qual permite
02.  /*o processo de reuso
03.  public class reuso {
04.  public static void main(String args[]){
05.  /*recebe a imagem de entrada
06.  String diretorio = "./ALGRAINS.BMP";
07.  PlanarImage resultado;
08.  /*Instancia a classe híbrido, a qual contém todo o processo
09.  /*de segmentação do sistema
10.  MH.hibrido reusoComponente = new hibrido();
11.  resultado = reusoComponente.instanciaComponente(diretorio);
12./*mostra a imagem resultado após o processo de segmentação híbrida
13.  JAI.create("filestore",resultado,"./resultado.tiff");
14.    }
15.  }
```

Desta forma, os passos utilizados na classe reuso foram implementados conforme o código acima, pois tem-se o objetivo de incorporar o método híbrido ao sistema Arthemis, o qual está sendo desenvolvido pelo grupo PIGS[5], uma vez que esses comandos são exatamente aqueles que os desenvolvedores necessitam saber para agregar um componente novo a seu sistema.

Com isso, o Arthemis poderá, ao final desse trabalho, contar com mais uma ferramenta de segmentação, auxiliando seus usuários no processo de análise de imagens. É importante também frisar que o componente híbrido poderá ser reutilizado em outros sistemas, desde que possuam os mesmos propósitos (segmentação por contorno, utilização de sementes, entre outros), os quais foram desenvolvidos nesse método.

[5]O Grupo de Processamento de Informação Multimídia (PIGS) dedica-se particularmente ao desenvolvimento de pesquisas na área de automação e informática industrial, integrando conceitos e técnicas de processamento e análise de imagens, reconhecimento de padrões e visão computacional. O PIGS também atua na área de engenharia de software, desenvolvendo sistemas de imageamento, construídos através de componentes de software e padrões de projeto. Recentemente, o PIGS ampliou as suas linhas de pesquisa para incluir o desenvolvimento de jogos para computador.

4 SEGMENTAÇÃO DE IMAGENS

Para extrair informações de imagens digitais é necessário separar do restante da imagem, os objetos de interesse que a compõem. Este processo é denominado segmentação. Por exemplo: numa imagem de células, pode-se segmentar as bordas das células do resto da imagem. O mesmo processo ocorre numa imagem do coração, quando se separa o ventrículo esquerdo do restante, bem como numa imagem de texto, quando se segmenta os parágrafos pela forma com que eles são alinhados: centralizado, à direita ou à esquerda.

Estes exemplos são suficientes para dar a noção do quão importante é conseguir um resultado preciso na segmentação dos objetos de interesse numa imagem. É a partir dessa segmentação que se extraem informações da imagem, na maioria dos casos, medidas dos objetos segmentados. Geralmente, percebe-se uma certa facilidade e rapidez na identificação visual de objetos ou informações de uma cena, mesmo quando ela não é estática. Isso pode levar a pensar que a automatização da mesma seja igualmente simples, o que não é verdade, uma vez que, computacionalmente, segmentar uma imagem é uma tarefa difícil e cujos resultados nem sempre são satisfatórios.

Há diversos problemas que impedem o sucesso de uma segmentação como, por exemplo, o sombreamento de objetos tridimensionais, presença de ruído na imagem digitalizada, manchas, distorções geométricas, falta de informação suficiente para distinguir as partes da imagem, excesso de objetos na imagem, sobreposição de objetos além da dificuldade para quantificar características complexas como textura.

As abordagens existentes na literatura para solucionar esses problemas são múltiplas e basicamente levam à noção de região, de contorno, de textura, de contornos ativos, ou ainda a uma abordagem mista. Na solução desses problemas é preferível a abordagem mista, mesmo que seu uso seja mais complicado [AYA 2004].

A detecção de regiões em uma imagem pode ser feita com dois objetivos: extrair uma determinada região ou dividir a imagem num conjunto de áreas distintas. Uma região de uma imagem é definida como um conjunto de pontos ligados onde, de qualquer um de seus pontos pode-se chegar a qualquer outro ponto dessa mesma região ou por um caminho completamente contido em seu interior. Estas regiões normalmente apresentam alguma característica homogênea que, comumente, é a continuidade dos níveis de cinza de seus pixels [ANT 1999]. A Figura 4.1 mostra uma imagem sendo dividida em regiões.

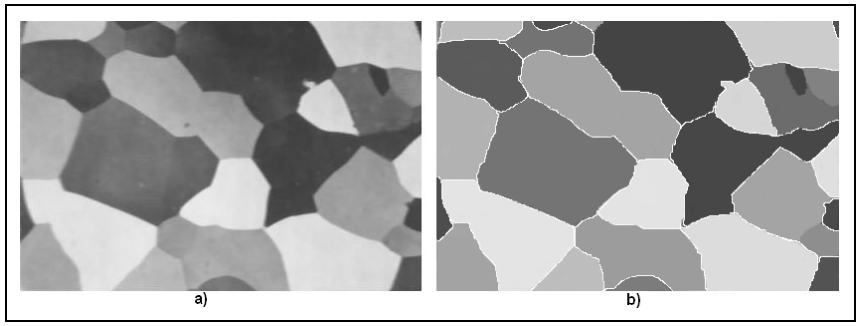

Figura 4.1: Resultado do processo de segmentação por região. a) Imagem Original. b) Imagem segmentada.

A segmentação por contorno é definida como uma mudança brusca de níveis de cinza entre duas regiões relativamente homogêneas. Ela pode aparecer como uma seqüência de pontos, uma linha, um segmento, uma curva ou uma forte variação do nível de cinza médio [ANT 1999]. Na Figura 4.2 é possível visualizar a variação dos níveis de cinza onde, o gato, a mesa, o livro e o monitor são regiões diferentes presentes na imagem.

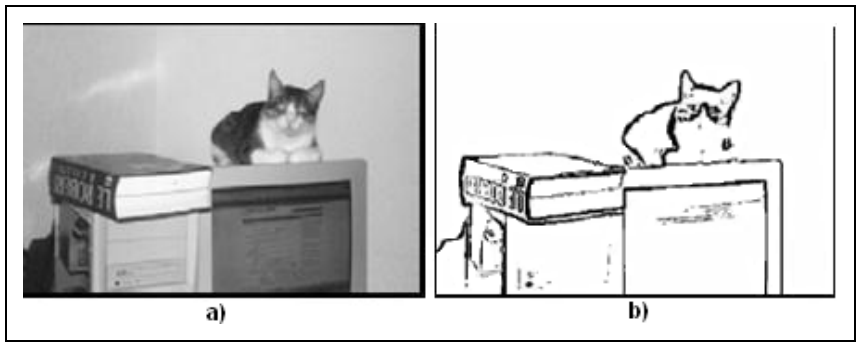

Figura 4.2: Resultado do processo de segmentação por contorno. a) Imagem Original. b) Imagem segmentada.

A segmentação por textura permite colocar em evidência os motivos da imagem, considerando-se as propriedades de regularidade e de repetição [STE 1999], (Figura 4.3). A noção de textura supõe que:

1. O motivo seja um agrupamento aleatório ou não aleatório de subconjuntos da imagem;
2. As entidades detectadas sejam uniformes e tenham aproximadamente as mesmas dimensões em qualquer lugar da imagem;
3. O motivo local seja repetido ao menos estaticamente numa região comparativamente grande em relação ao tamanho desse motivo.

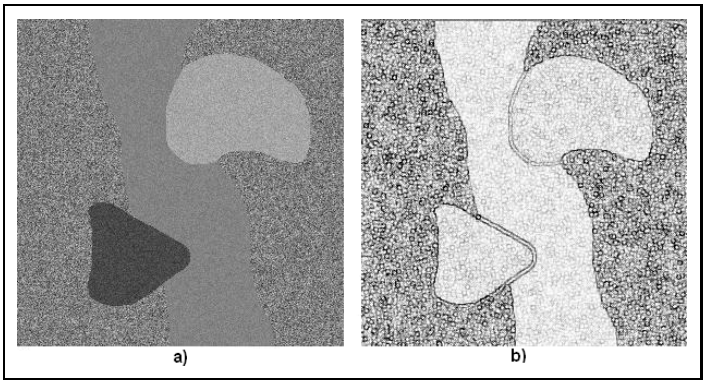

Figura 4.3: Resultado do processo de segmentação por textura. a): Imagem Original. b): Imagem segmentada.

Como não é fácil encontrar uma noção própria de textura, alguns pesquisadores consideram que ela pode revestir-se de um aspecto anárquico e homogêneo (exemplos da erva, da casca de árvore, entre outros) enquanto outros consideram que ela se reveste mais de um aspecto determinístico e estruturado (exemplos do tecido, do muro, etc.).

Finalmente vamos considerar o método de contornos ativos, o qual pode ser definido como um modelo de uma curva deformável. O modelo de contorno ativo é uma técnica de ajuste de uma curva deformável em relação às bordas de regiões homogêneas da imagem, baseada na minimização [AYA 2004].

Existem duas funções associadas a energia atuando no processo de segmentação por contorno ativo: a energia interna e a energia externa. A energia interna está relacionada ao comprimento do contorno e manter o mesmo espaçamento entre os pontos, conforme mostra a Figura 4.4(a). Já a energia externa está relacionada às informações dos níveis de cinza. Essa energia vai gerar um campo de força externo à curva que deverá "puxar" a mesma em direção às bordas, Figura 4.4(b).

Desta forma, o modelo está sempre procurando minimizar a sua função energia. Assim, pode-se dizer que é um método dinâmico. Durante o processo de minimização, o contorno ativo tem um comportamento parecido ao de uma cobra que rasteja, daí originou-se a denominação de "*snakes*". Na Figura 4.5, é possível visualizar um exemplo da segmentação por contorno ativo sendo aplicada em uma imagem.

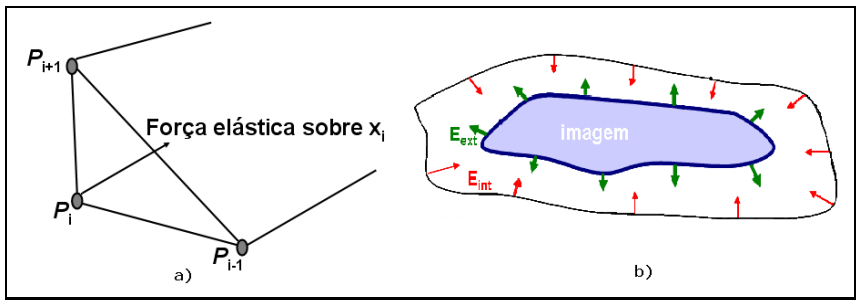

Figura 4.4: Processo de segmentação por contorno ativo. a) Mede-se o comprimento da imagem, seguindo o mesmo padrão de distância em toda a extensão da imagem. b)A curva interna é puxada para fora a fim de atingir o equilíbrio, formando assim, um contorno na imagem.

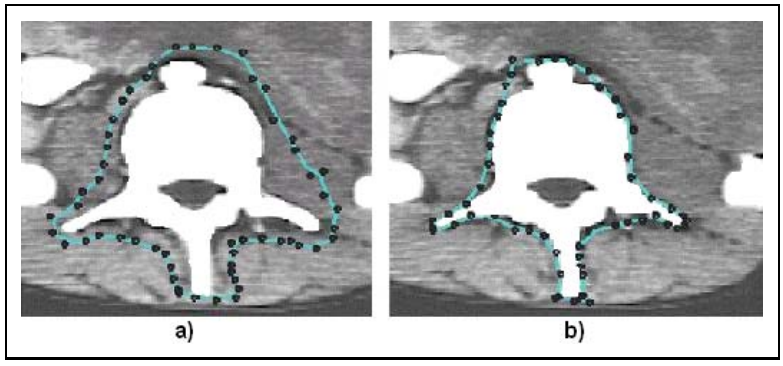

Figura 4.5: Segmentação por contorno ativo. a) Curva procurando o mínimo da função de energia. b) Imagem que mostra o resultado da função de energia mínima, encontrando o objeto desejado. Adaptada de [D'OR 2003].

4.1 Métodos de Segmentação

Nesta seção, dois métodos serão apresentados: *Mean Shift* e *Watersheds - Image Foresting Transformation (IFT)*. Ambos foram utilizados no desenvolvimento do método de segmentação híbrida e serão analisados a seguir.

4.1.1 Método Linear - *Mean Shift*

O algoritmo *Mean Shift* é uma técnica de classificação não supervisionada por estimação do gradiente de densidade de pontos no espaço de características, a qual foi proposta em 1975 por Fukunaga e Hostetler [K.FU 1990], e reapresentada mais recentemente por Cheng [CHE 1995].

4.1.1.1 O procedimento Mean Shift

O método de segmentação por *Mean Shift* é um algoritmo eficiente, o qual detecta agrupamentos de amostras por similaridade de características, isto é, detecta "clusters" de pontos no

espaço de características que constituem-se as classes adotadas na classificação.

Quando esta técnica é aplicada a um espaço de cores, produz uma diminuição do número de cores usados para descrever uma imagem. Isso é o mesmo que fazemos quando atribuímos o mesmo nome a todo um conjunto de matizes e tonalidades. Assim, por exemplo, reduzimos a azul o azul escuro, o azul claro, o marinho, o celeste o turquesa e mais uma infinidade de outros azuis. Para aproximar a classificação produzida pelo mean shift à sensação de cor produzida pelo sistema visual humano, utiliza-se o espaço de cores L*u*v*. A seguir serão apresentadas as etapas utilizadas no desenvolvimento desse método.

O algoritmo *Mean Shift* consiste em cinco etapas, as quais serão explanadas a seguir[1]

1. Escolhe-se o tamanho de uma janela, a qual será utilizada como um campo e onde os pontos serão verificados a fim de analisar se pertencem ou não a esse campo de trabalho.

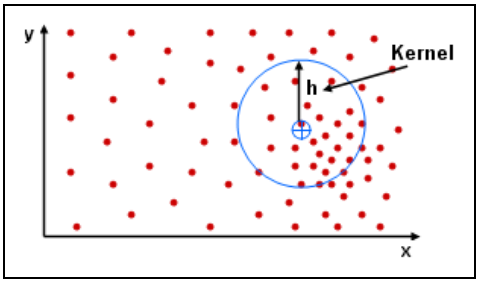

Figura 4.6: Nessa figura é possível visualizar a escolha do tamanho da janela através do raio h.

2. Seleciona-se um ponto onde estará localizada a janela, o qual será o ponto central da mesma.

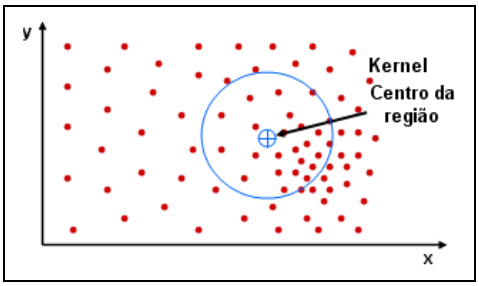

Figura 4.7: Ponto selecionado a fim de localizar a janela.

[1]As figuras utilizadas para exemplificar o método de segmentação por *Mean Shift* foram retiradas do artigo Feature Sensitive Mesh Segmentation with Mean Shift de Hitoshi Yamauchi e Seungyong Lee [YAM 2005], conforme as páginas 13,14,15 e 16.

3. Computa-se a média local dos pontos dentro da janela. Na seqüência, descobre-se o centro de massa.

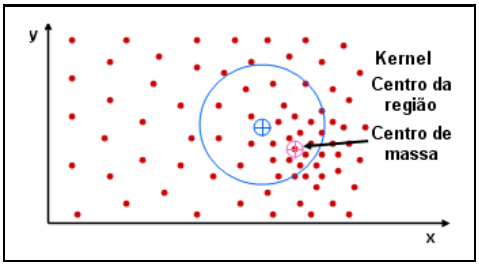

Figura 4.8: Descobre-se o centro de massa a fim de realizar o deslocamento da janela.

4. Desloca-se a janela para o ponto médio calculado na etapa anterior. A diferença entre a média local e o centro dão origem ao vetor *Mean Shift* em cada janela.

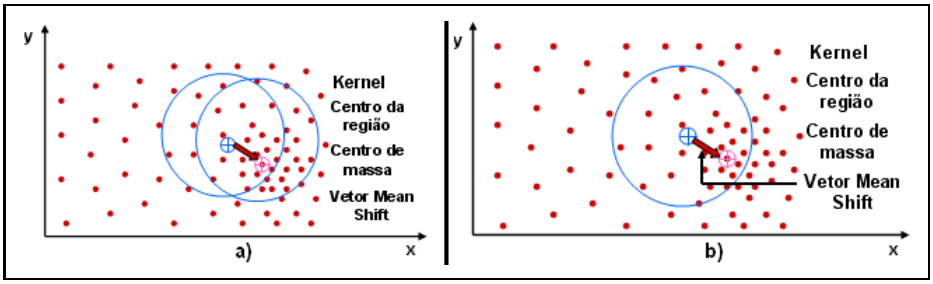

Figura 4.9: Desloca a janela para um outro ponto médio e a diferença entre o ponto médio e o centro *x* gera o vetor *Mean Shift*. [H.YA 2005],p.14. a): Deslocamento da janela até o ponto médio calculado. b): Vetor *Mean Shift*.

5. Executa-se o passo três e quatro até que todos os pontos tenham convergido para o mesmo valor(cor ou textura) do ponto médio. A convergência está alcançada quando o vetor médio do deslocamento é aproximadamente igual a zero.

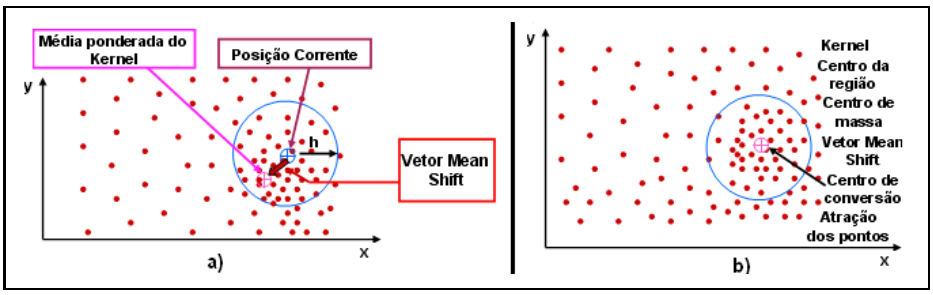

Figura 4.10: Executa os passos três e quatro até que o vetor *Mean Shift* alcance um valor próximo a zero. (a): Mostra os passos 1, 2, 3 e 4 sendo executados várias vezes, a fim de descobrir o ponto médio da imagem. (b): Realiza o processo de convergência dos pontos.

Após ter realizado todos os passos do processo de segmentação por *Mean Shift*, o conjunto de pontos da imagem original será reduzido. Isso porque, foi usado como parâmetro, uma característica do ponto central, a cor, levando todos os outros pontos, pertencentes a essa janela a ficarem com as mesmas características desse ponto central, reduzindo, assim, a quantidade de pontos existentes na imagem. Abaixo é possível visualizar um exemplo da técnica de segmentação por *Mean Shift*.

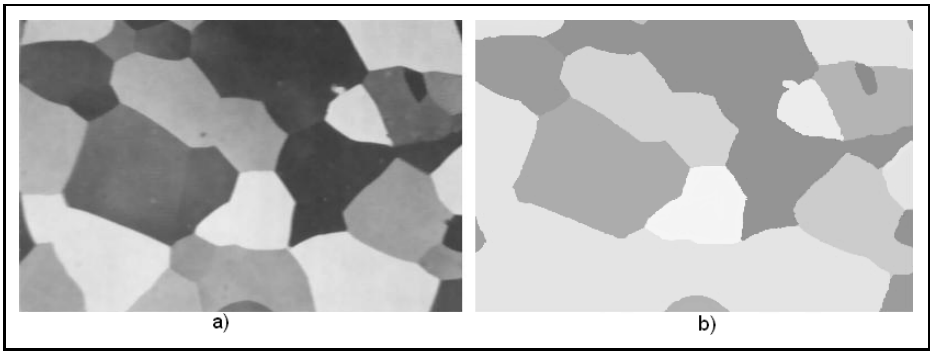

Figura 4.11: Nessa figura é possível visualizar a união de certas regiões, diminuindo os tons de cinza presentes na imagem. a): Imagem Original. b): Imagem segmentada através do método *Mean Shift*.

4.1.1.2 Exemplos da Segmentação

Três parâmetros controlam a segmentação: a busca do raio h, o *threshold* T1 (que impõe o limite da densidade), e o T2, o qual determina o tamanho mínimo do componente conectado.

Os resultados apresentados aqui são obtidos com h=4, T1=50 e T2=1000. Uma imagem típica do leucócito é mostrada na Figura 4.12-a. A imagem segmentada é apresentada em pseudocores na Figura 4.12-b. Pseudocores idênticas são usados para mostrar a delineação do *cluster*

na Figura 4.13-e, onde esses clusters são deslocados para obter uma melhor visualização. As Figuras 4.13-a, b, c e d, respectivamente, correspondem a *n=6005* vetor de cor, parte da amostra com *m* = 73, candidatos à centro do cluster, e *p = 4* centros de *clusters* [COM 2001].

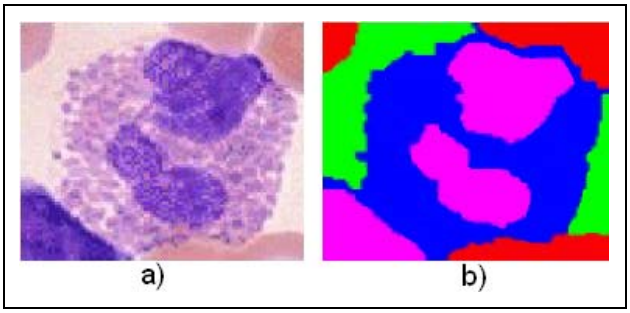

Figura 4.12: Método *Mean Shift* sendo aplicado a uma imagem de leucócitos, passando como parâmetro *h=4* T1=50 e T2=1000. a): Imagem Original. b): Imagem resultante do procedimento *Mean Shift*, sendo representada em pseudocores. Adaptada de [COM 2001], p.7

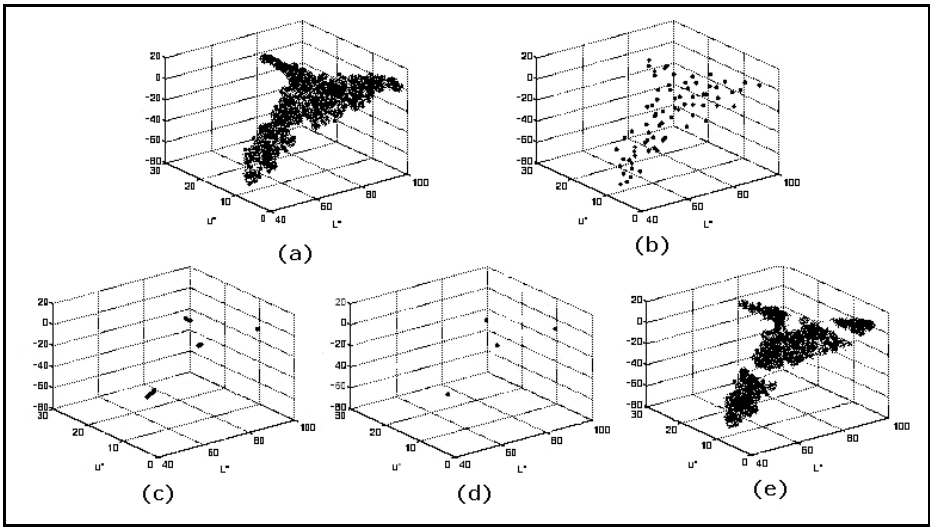

Figura 4.13: Procedimento *Mean Shift* sendo representado em gráficos: (a): Vetor de cor. (b): Conjunto de amostras. (c): *Clusters* candidato ao centro. (d): Agrupamentos centrais. (e) *Clusters* delineados. Adaptada de [COM 2001], p.7

A decomposição resultante é significativa no domínio espacial e no espaço de cor; a textura do citoplasma, por exemplo, é classificados em um *cluster*.

A qualidade da segmentação pode também ser avaliada na figura 4.14 que mostra a imagem original, as imagens com contorno, e as imagens segmentadas de uma espécime de linfoma da zona do manto (superior) e de duas espécimes de leucemia linfocítica crônica (abaixo).

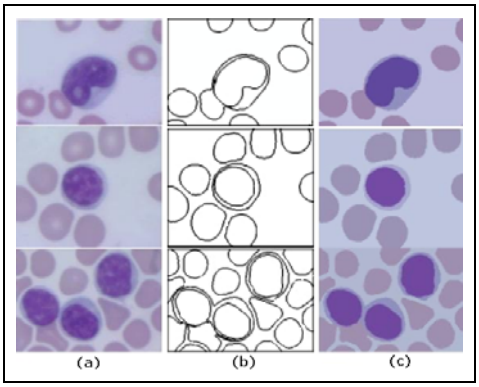

Figura 4.14: Resultado da segmentação por *Mean Shift* em uma imagem médica. a): Imagem Original. b): Contornos. c): Imagem Segmentada. Adaptada de [COM 2001], p.8

Os resultados adicionais são apresentados na Figura 4.15, onde o núcleo de cada célula foi delineado usando o algoritmo *Mean Shift* e nenhum pós-processamento adicional foi necessário. O conjunto de dados contém imagens de diferentes cores, contrastes, nível de ruído e tamanho (para a conveniência, são indicadas no mesmo tamanho).

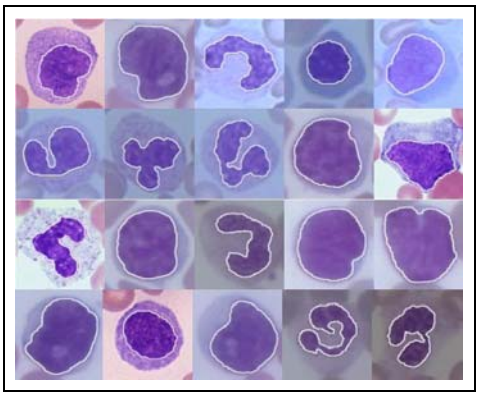

Figura 4.15: A segmentação do núcleo é realizado em várias categorias de células. A borda do núcleo é marcada com um contorno branco. Adaptada de [COM 2001], p.9

4.1.2 **Método Não-Linear** - *Watersheds*

A segmentação de imagens por *watersheds* pode ser classificada como uma técnica de segmentação baseada no crescimento de regiões ou como uma técnica de morfologia matemática. Esse método baseia-se no princípio de “inundação de relevos topográficos”, onde a imagem é inundada com água, permitindo a formação de bacias de captação e de linhas *watersheds*, ou linhas divisoras de água, possibilitando o particionamento da imagem em regiões.

É possível visualizar a “inundação” de duas maneiras distintas: conforme a Figura 4.16, a

água vinda de cima, como se fosse chuva; ou água vinda de baixo, como se o relevo estivesse perfurado nos pontos de altitude mínima e fosse imerso em um lago [ROE 2000],[WAN 2004]. Conforme as bacias vão sendo inundadas, águas provenientes de diferentes bacias se encontram, formando, nos pontos de encontro, represas ou linhas divisoras de águas, as chamadas *watersheds*.

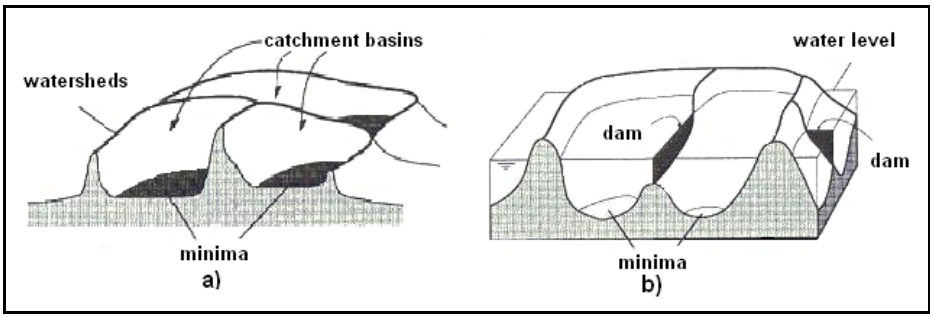

Figura 4.16: a) Linhas divisórias, mínimos e bacias de captação em um relevo topográfico em imagens em nível de cinza. b) Construção de represas que irá dividir duas bacias de captação. [JäHN 2000], p.512

De acordo com a Figura 4.16, as regiões mais baixas seriam correspondentes as de menor gradiente[2] e as regiões mais altas as de maior gradiente. O crescimento de regiões seria equivalente a uma inundação feita a partir da abertura de um pequeno furo nas regiões que, partindo de um mínimo local formem uma bacia hidrográfica. Eis o porquê do nome divisor de águas. O agrupamento dos pixels é feito por mecanismos de busca de valores próximos a partir de cada mínimo local [WAN 2004].

Segundo Vicent e Soille [VIC 1991], temos vários algoritmos para a definição de transformada da *Watersheds*, entre eles pode-se citar algoritmos por imersão, por componentes gráficos, por distância topográfica e por *Image Foresting Transform* (IFT). O presente trabalho utilizará o algoritmo IFT, o qual será analisado a seguir.

4.1.2.1 Watersheds - Image Foresting Transformation (IFT)

Muitos problemas de processamento de imagem podem ser interpretados como um problema de divisão da imagem baseado, em parte, nos pixels raízes, onde cada raiz define uma zona de influência que consiste em agrupar os pixels mais próximos a elas [LOT 2002].

O image foresting transform (IFT) reduz tais problemas, pois transforma o trajeto de uma floresta de custo-mínimo em um grafo dirigido cujos nós são os pixels da imagem e cujos arcos são definidos por uma relação de adjacência qualquer entre esses pixels. Um tipo de relação de adjacência razoável é a adjacência Euclideana, a qual define arestas saindo de um vértice para

[2]O gradiente morfológico é comumente utilizado para detectar contornos, sendo um passo intermediário para subseqüentes aplicações, como no caso da segmentação de imagens [FAC 1996].

todos os outros vértices distantes até um dado raio (de acordo com a métrica euclideana), veja um exemplo na Figura 4.17.

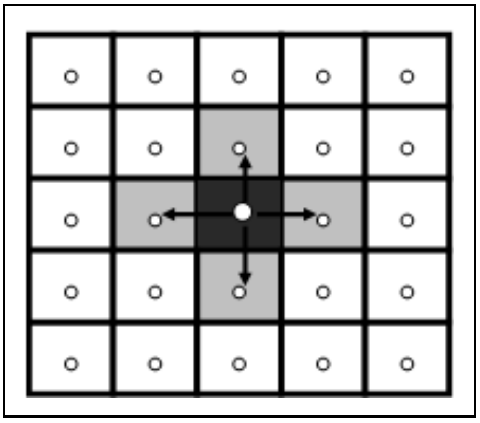

Figura 4.17: Adjacência euclideana de uma figura 2D de raio 1, o qual analisa apenas os pixels que estão ao seu redor. [ALE 2000], p.18

O custo de um caminho é calculado por uma função dependente da aplicação e com base em propriedades locais da imagem - tais como brilho, gradiente e posição de pixel ao longo do caminho.

Para uma função de custo adequada, a IFT associa a cada pixel da imagem um caminho de custo mínimo. Dessa forma, a imagem será particionada em uma floresta de caminhos ótimos. Nesta floresta, cada árvore tem como raiz um pixel semente e como nós-filhos os pixels da imagem mais “conexos” com a raiz que com qualquer outra semente, em algum sentido apropriado [FAL 2003], (Figura 4.18).

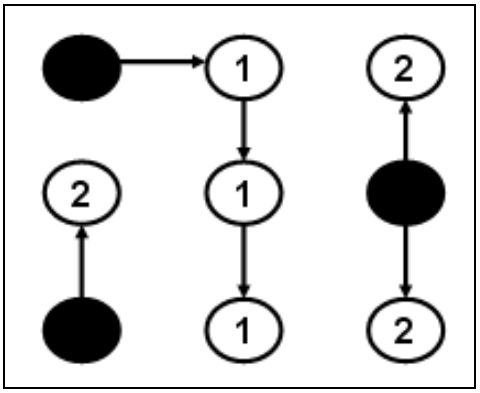

Figura 4.18: Grafo resultante da adjacência da Figura 4.17. Os pixels pintados de pretos são as sementes e os pixels brancos são seus nós-filhos [ALE 2000], p.18