UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Método baseado em processamento digital de imagens para diagnóstico precoce de micro-estruturas dentárias

Yuji de Oliveira Ohnishi

Orientador: **Prof. Dr. Paulo Estevão Cruvinel**

Co-orientador: **Prof. Dr. Walter Antônio de Almeida**

São Carlos

Maio/2005

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

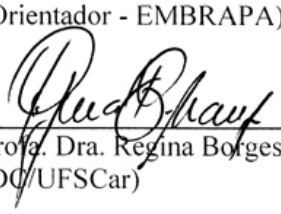
"Método baseado em processamento digital de Imagens para diagnóstico precoce de micro-estruturas dentárias"

YUJI DE OLIVEIRA OHNISHI

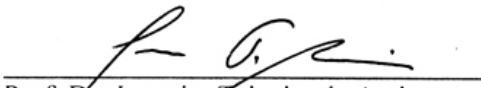
Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Membros da Banca:

Prof. Dr. Paulo Estevão Cruvinel
(Orientador - EMBRAPA)



Profa. Dra. Regina Borges de Araujo
(DC/UFSCar)



Prof. Dr. Joaquim Teixeira de Assis
(Instituto Politécnico/UERJ)

São Carlos
Maio/2005

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

O38mb

Ohnishi, Yuji de Oliveira.
Método baseado em processamento digital de imagens para diagnóstico precoce de micro-estruturas dentárias / Yuji de Oliveira Ohnishi. -- São Carlos : UFSCar, 2005.
127 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2005.

1. Reconhecimento de padrões. 2. Segmentação de imagens. 3. Classificação bayesiana. I. Título.

CDD: 006.4 (20ª)

DEDICATÓRIA

A Deus e a minha família que sempre
me ajudaram nos momentos mais difíceis.

AGRADECIMENTOS

A Deus pela oportunidade de realizar este sonho.

Aos meus pais e irmão pela força e incentivo a todo momento.

Ao professor Dr. Paulo Estevão Cruvinel, pelos ensinamentos e amizade, incentivando-me nos momentos mais difíceis.

Ao professor Dr. Walter Antônio de Almeida pela co-orientação.

Aos amigos que fiz em São Carlos: Diego Duarte, João Paulo Papa, Murilo Stelzer, Fabrício Breve, Gabriel Alves, Luciana Betetto e a todos do GAPIS.

À Comissão de Ética em Pesquisa da Fundação Educacional de Barretos, por permitir a utilização das radiografias para os estudos de casos.

À CAPES pela ajuda financeira durante todo o curso e a todos que direta ou indiretamente me ajudaram na conclusão deste mestrado.

RESUMO

Neste trabalho é apresentado um novo método com base em processamento de imagens que possibilita diagnóstico precoce da formação de microestruturas dentárias. No desenvolvimento foram utilizadas técnicas de segmentação de imagens, transformada de Hough para a detecção de padrões circulares de formação das microestruturas dentárias e técnicas de classificação Bayesiana para classificação de pacientes portadores de diferentes estruturas dentárias. Resultados ilustram estudos de caso com qualificação do método desenvolvido, bem como a adoção do padrão DICOM habilita o método para uso em ambientes clínicos especializados.

ABSTRACT

This work presents a new method based on digital image processing dedicated to establish diagnosis of dental microstructures. The system uses image techniques such as segmentation, Hough Transform for circular pattern recognition for dental microstructures, and Bayesian technique to classify patients that present different dental structures. Results show the qualification of the method. Also, the adoption of the DICOM standard has shown the suitability of the method to be used in clinical and specialized environments.

SUMÁRIO

ÍNDICE DE FIGURAS

INTRODUÇÃO

A cárie dentária é um processo patológico de destruição dos tecidos dentários pelos microrganismos. No homem antigo, esta estava usualmente localizada na junção amelocementária ou no cemento, enquanto que no homem moderno, os sítios mais comuns de cáries são os sulcos e fissuras (NEWBRUM, 1988).

Esse fato se explica pelo tipo de alimentação, onde antigamente ingeriam-se alimentos mais fibrosados e naturais, e hoje em dia estes alimentos são mais pastosos, mais cozidos, facilitando assim sua aderência e permanência nos sulcos e fissuras das superfícies oclusas dos dentes.

Se não tratada, a cárie dentária irá evoluir, atingir a polpa dentária, podendo provocar alterações pupares e/ou periapicais, aparecimento de microrganismos, que poderão ficar restritos a esta região ou irem para a circulação sangüínea determinando uma bacteremia transitória, podendo provocar uma endocardite, e até levar o paciente ao óbito, ou doenças como abscesso e derrame cerebral, meningite crônica e aguda, abscesso pulmonar, doença infecciosa ocular e de pele, tétano e alterações na gravidez (WEYNE, 1999). Na maioria dessas patologias, os pacientes apresentam vários dentes com extensas cavidades cariosas e com quadro de periodontite (problemas gengivais) avançada.

Para evitar esses males bucais, os profissionais têm o compromisso de realizar periodicamente um tratamento preventivo e curativo em várias escolas públicas e outros locais de uma comunidade.

Recentemente, numa pesquisa realizada entre 2.200 estudantes com idades de 7 a 20 anos, foi constatado que apenas 13% deles fazem tratamento dentário. O número de pessoas de várias idades com cáries continua elevado (PINTO, 1999).

É preciso muito cuidado com a saúde bucal. Afinal, é através da boca que podem ser transmitidas muitas doenças. No trabalho preventivo, realizado nas escolas, são realizados exames dentários nos alunos, como bochecho com flúor e a escovação, além de reunião com os pais, onde é destacada a importância do trabalho de prevenção.

Segundo especialistas, a higiene bucal deve ser realizada desde a gestação até a fase adulta. Na gestação, a mãe deve se preocupar com sua própria alimentação, ingerindo substâncias ricas em cálcio como leite, queijo, legumes e verduras, que são fundamentais para que a dentição da criança seja resistente.

As pessoas que costumam não escovar os dentes regularmente desde a infância, já aos 20 anos podem sofrer perda do primeiro molar permanente; e aos 40 anos a perda de dente se acentua através das doenças periodontais, ou seja, inflamação na gengiva, afetando, na maioria das vezes, os dentes anteriores. Cerca de 30% da população de 60 anos já usam dentadura.

Vários estudos têm sido realizados para analisar os diferentes métodos de diagnóstico de lesão de cárie, embora com resultados bastante distintos. O diagnóstico clínico visual, associado ao exame radiográfico, tanto em superfícies oclusas como proximais, são os métodos mais regularmente utilizados e relatados como os mais sensíveis (ARAUJO & FUGUEIREDO, 1999).

O esmalte dentário normal é rígido e formado por cristais de hidroxiapatita, que são firmemente unidos, dando uma aparência semelhante ao vidro translúcido. Quando ocorre perda mineral, o esmalte começa a apresentar porosidade que conduz a uma mudança nas propriedades óticas do mesmo, tornando-o menos translúcido, onde clinicamente pode ser observado como alterações esbranquiçadas.

Até o presente momento, pelos métodos disponíveis, não é possível diagnosticar a doença cárie nos estágios iniciais, onde as mudanças nos tecidos mineralizados são muito pequenas para serem detectadas, pois o desenvolvimento das lesões cariosas envolvem uma interação complexa entre inúmeros fatores dentro do ambiente oral e os tecidos duros dos dentes.

O trabalho preventivo ganha importância e o desenvolvimento de técnicas avançadas como as que utilizam imagens para diagnóstico precoce, principalmente em crianças, se constituem atualmente em objeto de destaque e importância.

Com base em extensa revisão bibliográfica realizada, percebeu-se a ausência de métodos voltados à área odontológica que tenham como objetivo a detecção de micro-estruturas dentárias como forma de caracterização de cáries em estágios iniciais de

formação, possibilitando o especialista aplicar métodos preventivos (aplicações de flúor, métodos terapêuticos, entre outros) que inibam o desenvolvimento da mesma, o que é im indicador da originalidade do modelo apresentado neste trabalho.

Isto posto, o presente trabalho apresenta um método baseado em processamento digital de imagens para diagnóstico precoce de micro-estruturas dentárias, que para tal, são utilizadas técnicas de segmentação de imagens digitais, especificamente, segmentação orientada a regiões, técnicas de reconhecimento de padrões por meio do uso da Transformada de Hough, técnicas de classificação Bayesiana com uso da distância de Mahalanobis e como forma de armazenamento, adotou-se o padrão DICOM, que já vem sendo utilizado nas áreas biológicas.

Neste documento buscou-se organizar a sua apresentação considerando os seguintes capítulos:

Capítulo 1: *Segmentação e Classificação em Imagens Digitais*, onde são mostradas algumas técnicas para segmentação de imagem e classificação Bayesiana.

Capítulo 2: *O Padrão DICOM e sua utilização em imagens digitais para fins de diagnósticos*, onde é feito um estudo sobre as partes que compõem o padrão de armazenamento adotado.

Capítulo 3: Desenvolvimento do método baseado em processamento digital de imagens para diagnóstico precoce de micro-estruturas dentárias, com abordagem sobre os resultados obtidos e validação.

Capítulo 4: Conclusões e propostas para trabalhos futuros.

1 SEGMENTAÇÃO E CLASSIFICAÇÃO EM IMAGENS DIGITAIS

1.1 Imagens naturais e digitais

Quando se olha para cenas do mundo real ou imagens (pinturas, fotografias, etc) que representam uma cena, consciente ou inconscientemente, se é levado a observar suas estruturas que em muitas vezes são identificadas como objetos reais. Esses objetos podem ser concretos, como quando se vê árvores, carros, ruas em uma fotografia, ou abstratos, onde o que são percebidas são estruturas geométricas que os abstraem, como em pinturas abstratas. Independente do tipo de objeto retratado em uma imagem, o estímulo visual que chega à retina não passa de uma informação visual não-estruturada, o que uma imagem digital representa perfeitamente bem. Porém, no processo de percepção natural, uma vez recebido tal estimulo visual, naturalmente, "processa-se" esses estímulos em estruturas bem definidas (objetos) e relacionadas (posição de um objeto em relação a outros).

Em termos computacionais e matemáticos, imagens digitais são funções $f(x, y)$, onde x e y são coordenadas de um determinado pixel do domínio da imagem (o plano, um retângulo,...) e $f(x, y)$ é um número real que representa a intensidade de brilho ou nível de cinza da imagem no pixel de coordenadas (x, y). Esta representação digital do mundo real nada mais é do que uma informação visual não-estruturada da qual especialistas buscam extrair informações. Entretanto, neste caso, seria necessário passar de uma imagem digital não-estruturada para uma forma estruturada da mesma maneira como é feito no processo de percepção natural.

A forma como naturalmente as informações não-estruturadas são "processadas" passou a ser investigada no âmbito científico por psicólogos da escola *Gestalt* na década de vinte. Porém, foi no final da década de sessenta que o problema de transformação de uma imagem não-estruturada para estruturada foi estudado por engenheiros que desejavam

extrair automaticamente informações de imagens digitais. Foi a partir do interesse de construir robôs com uma capacidade de percepção apurada, o que seria necessário um melhor entendimento da visão humana e animal, que esses estudos foram desenvolvidos. Foi então que surgiu a necessidade de separar os vários objetos que compunham uma dada cena para estabelecer relações entre eles. Surgiu então, a necessidade de segmentar imagens.

1.2 Problemas em segmentação de imagens

Para se extrair informações de imagens digitais é necessário, inicialmente, dividi-la em partes ou objetos que a compõem. Este processo é denominado de Segmentação de Imagem e pode ser formalmente definido como sendo um processo de identificação (através de um algoritmo numérico) das regiões homogêneas de uma determinada imagem e das bordas ou limites destas regiões (MOREL & SOLIMINI, 1995). Essas regiões homogêneas são, na maioria dos casos, objetos e as bordas, seus contornos.

Inúmeros algoritmos foram propostos para segmentação de imagem e, apesar da diversidade de ferramentas que se dispõem a realizar tal tarefa, todas utilizam, essencialmente, o modelo de detecção de bordas, tendo então como principal diferença, os procedimentos utilizados para minimizar algumas dificuldades encontradas no processo de segmentação. Tais dificuldades são: a homogeneidade dos objetos que constituem a cena (imagem), ou seja, quão definido é o fim de um objeto e início de outro, o grau de fidelidade entre a imagem analisada e a imagem original, e a fidelidade entre as bordas obtidas e as descontinuidades da imagem.

Várias questões são abordadas pelas inúmeras técnicas de segmentação de imagem, contudo, seis destas são redundantes e de análise matemática não tão elegante. Porém, com as três principais, se obtém o modelo de Mumford-Shah. O modelo atualmente dado como modelo geral de segmentação é uma variação do modelo de Mumford-Shah, conhecido como *GNC Algorithm*, proposto por Blake e Zisserman (MOREL & SOLIMINI, 1995); os demais ou são variantes do GNC ou são algoritmos que tendem a minimizar tais variantes.

1.3 Modelo Mumford-Shah

O modelo de Mumford-Shah define o problema de segmentação como um problema de homogeneidade na união das regiões (objetos) e detecção de borda: dada uma imagem $f(x,y)$, o problema é caracterizado como sendo a busca simultânea pela imagem dividida em regiões suaves $u(x,y)$ e um conjunto *K* de descontinuidades bruscas, as bordas (contornos) de $f(x,y)$. Desta forma a melhor segmentação de uma dada imagem é obtida minimizando a função

$$Seg[u(x,y),K] = g\left\{ \iint_{\Omega/K} (|\nabla u(x,y)|^2 + [u(x,y) - f(x,y)]^2) dx\,dy, length(K) \right\} \quad (1.1)$$

onde o primeiro termo impõe que $u(x,y)$ seja suave no lado de fora das bordas, a segunda, que a imagem $u(x,y)$ de fato se aproxime da imagem original $f(x,y)$, e o terceiro termo, que o conjunto de descontinuidades (bordas) tenha um tamanho mínimo (e portanto, sejam tão suaves quanto possível).

Algoritmos que minimizam a função acima têm como resultado rascunhos da imagem original, ou seja, imagens que dão a impressão de terem sido desenhadas. Além disso, a análise feita sobre imagens obtidas a partir de tais algoritmos costuma dar bons resultados (MUMFORD & SHAH, 1985).

1.4 Segmentação de imagens

Geralmente o primeiro passo no processo de análise de imagem é a segmentação, que consiste em dividir uma imagem em regiões ou objetos que a constitui, segundo algum critério, ou seja, a segmentação deve parar no momento em que o objeto desejado da imagem for isolado.

Freqüentemente, o resultado obtido no processo de segmentação não é uma imagem, mas um conjunto de regiões/objetos que serão utilizados para extração de informações. Desta forma a precisão da fase de segmentação determina o sucesso ou insucesso dos procedimentos de análise de imagem por computador (GONZALEZ & WOODS, 2001).

1.4.1 *Detecção de descontinuidades*

A teoria de detecção de descontinuidade foi formulada por Hildreth and Marr (ROSENFELD & THURSTON, 1976), e teve início em uma das mais completas tentativas de implementação de um sistema de visão de propósito geral na década de 80. Marr concluiu que o propósito de um sistema de visão é a construção de uma descrição simbólica da mudança de intensidade em uma imagem. O sistema proposto iniciou com aplicação de máscaras para localização de estruturas como pontos e retas de orientações e tamanhos diversos. Depois de localizadas estas estruturas, um conjunto de operações foi realizado na tentativa de eliminar os ruídos obtidos e agrupar estas estruturas iniciais em linhas maiores e contornos.

Com isso, existem, basicamente, três tipos de descontinuidades em imagem digital: pixels, linhas e contornos. Na prática, a forma mais comum para encontrar descontinuidades em uma imagem é aplicar uma máscara pixel-a-pixel. Utilizando, por exemplo, uma máscara 3 × 3, como apresentado na Figura 1.1, o procedimento se resume em computar o somatório dos produtos dos coeficientes da máscara com o nível de cinza de cada pixel onde a máscara está sendo aplicada. De forma geral, este procedimento pode ser representado pela expressão a seguir, onde z_i é o nível de cinza do pixel associado ao coeficiente w_i.

$$\begin{aligned} R &= w_1 z_1 + w_2 z_2 + \ldots + w_9 z_9 \\ &= \sum_{i=1}^{9} w_i z_i \end{aligned} \tag{1.2}$$

onde R é a resposta obtida pela máscara utilizada.

Como é de costume, o cálculo é realizado em relação à localização central da máscara. Para os casos onde a máscara está centrada nos pixels da borda da imagem, o cálculo é feito utilizando um valor apropriado do vizinho mais próximo.

w1	*w2*	*w3*
w4	*w5*	*w6*
w7	*w8*	*w9*

Figura 1.1- Exemplo de máscara 3 × 3, onde são os coeficientes da máscara.

1.4.1.1 Detecção de pixel

Para a detecção de pixels isolados em uma imagem pode-se utilizar a máscara mostrada na Figura 1.2. Um pixel é identificado na posição onde a máscara estiver centralizada se $|R| > T$, onde T é um limite não-negativo e R é dada pela equação (1.2).

-1	*-1*	*-1*
-1	*8*	*-1*
-1	*-1*	*-1*

Figura 1.2 - Máscara utilizada para detecção de pixels isolados.

Basicamente, é feita uma comparação entre um pixel (suposto pixel isolado) e os pixels vizinhos. Um pixel é encontrado quando houver uma diferença de nível de cinza entre o pixel central da máscara e os demais pixels vizinhos a este.

1.4.1.2 Detecção de linha

Uma formulação ligeiramente mais complexa é dada pela detecção de linhas em uma imagem, que consiste na aplicação das máscaras mostradas na Figura 1.3. A primeira máscara é utilizada para a detecção de linhas (de um pixel de espessura) na horizontal, ou seja, *R* [equação (1.2)] terá um resultado maior quando uma linha horizontal estiver situada na linha central da primeira máscara.

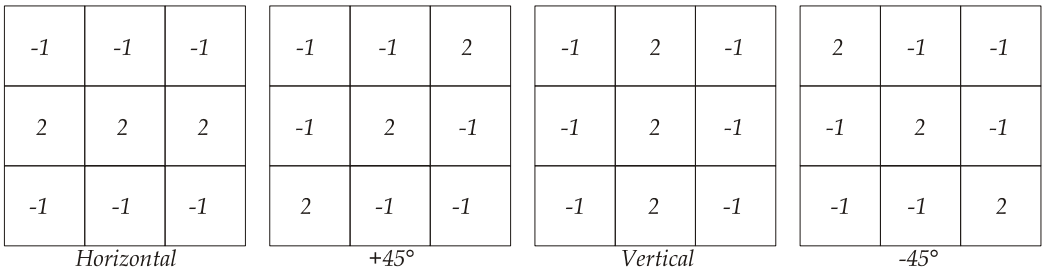


Figura 1.3- Máscara para detecção de linhas horizontais, diagonais e verticais

A mesma idéia é aplicada às demais máscaras da Figura 1.3, onde a segunda apresentará um *R* maior quando a linha estiver na diagonal, tombada para a direita (45°), a terceira, quando a linha estiver na vertical, e a quarta para linhas na diagonal, tombada para a esquerda (-45°).

Considerando R_1, R_2, R_3 e R_4 como resultados obtidos pelas respectivas máscaras apresentadas na Figura 1.3, onde R_i é dado pela equação (3.1-1), ao aplicar cada uma das máscaras em uma imagem; em um dado pixel da imagem, se $|R_i|>|R_j|$, para todo $j \neq i$, então este pixel é dito como uma possível localização de uma linha com a orientação (direção) da máscara *i*. Por exemplo, se em um pixel da imagem, $|R_1|>|R_j|$, para $j=2, 3, 4$, tal pixel é dado como uma possível localização de uma linha horizontal.

1.4.1.3 Detecção de contornos

Resumidamente, um contorno ou borda é o limite entre duas regiões com uma certa diferença de tons de cinza.

Esta abordagem é a mais utilizada quanto à detecção de descontinuidades, isto porque pontos e linhas de um pixel de espessura dificilmente ocorrem nas maiorias das aplicações reais. Porém, deve ser observada a existência de imperfeições no processo de aquisição de imagem onde bordas se comportam como "rampas", como mostrado na Figura 1.4 (direita).

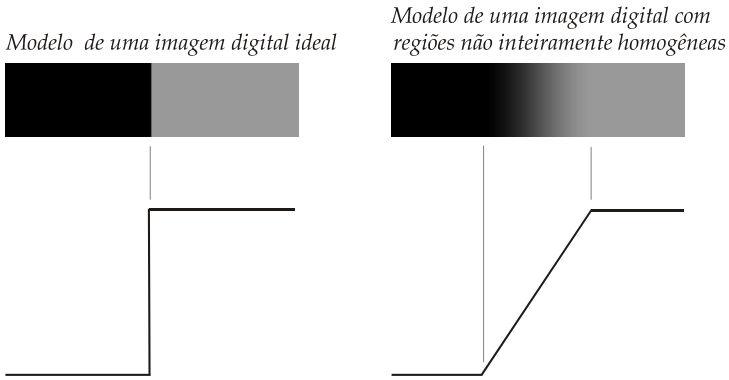


Figura 1.4- Modelo do limite entre duas regiões e seus respectivos diagramas de tons de cinza.

Geralmente, a detecção de bordas é realizada através do cálculo da primeira derivada (Gradiente $G(x,y)$) e segunda derivada (Laplaciano $L(x,y)$) da imagem, como é ilustrado na Figura 1.5.

Sendo $G(x,y)$ dado por:

$$\nabla f(x,y) = mag\left[\nabla f(x,y)\right] = \left[G_x^2 + G_y^2\right]^{1/2}, \tag{1.3}$$

onde $\nabla \mathbf{f}$ é o vetor gradiente dado por:

$$\nabla f(x,y) = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f(x,y)}{\partial x} \\ \dfrac{\partial f(x,y)}{\partial y} \end{bmatrix} \tag{1.4}$$

e $L(x,y)$:

$$\nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2} \tag{1.5}$$

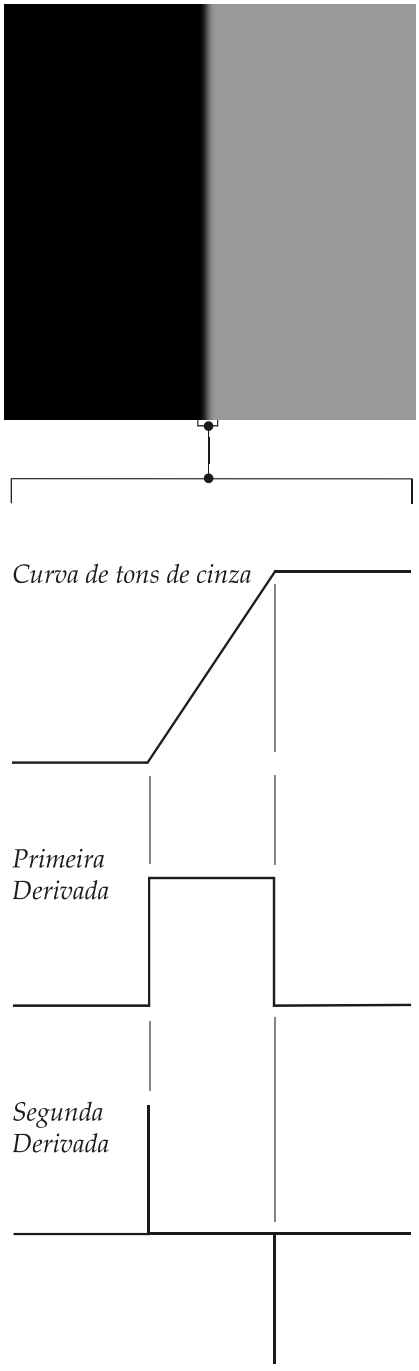


Figura 1.5- Duas regiões separadas por uma borda vertical com a primeira e segunda derivada da imagem

É possível observar que, pela primeira derivada, a curva é positiva quando se passa da região mais escura para a região mais clara e zero na região onde o tom de cinza se torna constante.

Com a segunda derivada, é possível observar que a curva é positiva para a parte do contorno onde é feita uma transição de uma região mais escura para uma região mais clara e negativa caso contrário.

Porém o interessante da segunda derivada é o fato da curva passar por zero durante a transição entre duas regiões, o que pode ser utilizado como um recurso para localização de borda em uma imagem.

Sendo assim, a primeira derivada é comumente utilizada que determinar a presença de bordas na imagem e a segunda, para determinar a localização dos pixels que constituem o contorno em questão.

Um problema que pode ser observado na utilização de filtros passa-alta (utilização da primeira e segunda derivada) na detecção de bordas é que tais filtros são extremamente sensíveis ao ruído, como é mostrado na Figura 1.6.

Uma solução possível é a aplicação de um filtro passa-baixa antes de se efetuar a detecção de borda.

A imagem gradiente não fornece diretamente os contornos. Na prática, ela fornece traços grossos e irregulares.

Para obter traços mais precisos e mais nítidos, usa-se o processo de limiarização (thresholding) da imagem do módulo do gradiente.

A precisão dos traços depende do valor da limiarização e da exatidão dos cálculos das derivadas: um valor alto usado na limiarização fornece traços finos, mas muitas vezes interrompidos, enquanto um valor baixo proporciona traços mais completos, porém mais grossos e podem produzir linhas que não existem na imagem de origem.

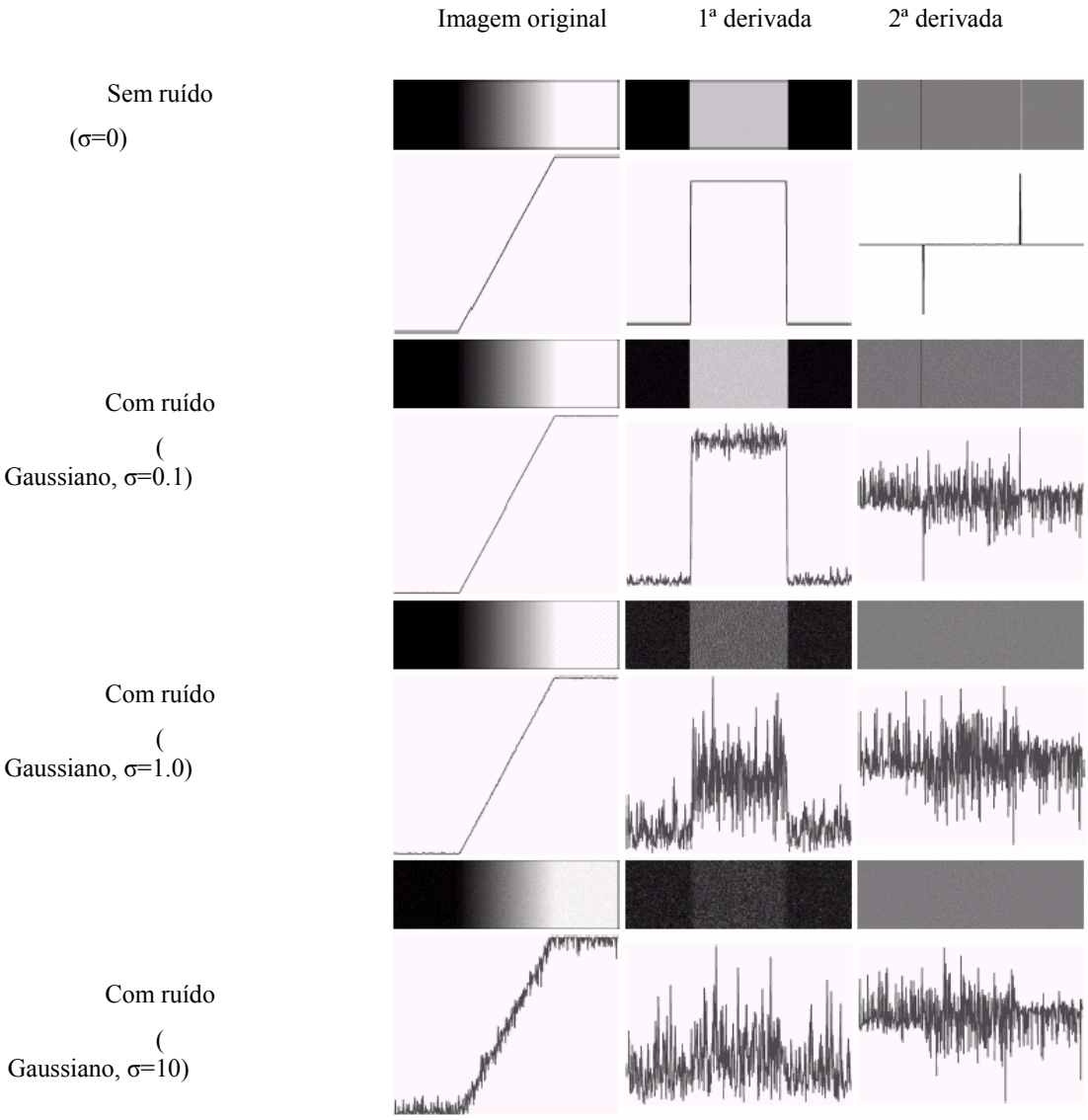


Figura 1.6- Demonstração da sensibilidade da primeira e segunda derivada ao ruído.

1.4.2 *Junção de contornos*

Como em alguns casos as bordas podem apresentar imperfeições devido ao ruído, ou seja, os contornos obtidos num processo de detecção de bordas podem ser na verdade um conjunto de segmentos não unidos. Para tornar a borda um único filamento ou

segmento separando duas regiões torna-se necessário a utilização de um pré-processamento para ligar estes vários fragmentos, obtendo assim, bordas uniformes e inteiras.

Vários métodos estão à disposição para a execução desta tarefa. Os principais são: Processamento Local, Processamento Global via transformada de Hough e Processamento Global via Técnicas de Grafos (GONZALEZ & WOODS, 2001).

1.4.2.1 Processamento local

Uma das técnicas mais simples para corrigir os erros de bordas não-contínuas que baseia-se em aplicar uma máscara 3×3, por exemplo, e detectar os pixels similares, segundo algumas propriedades.

As duas principais propriedades utilizadas são: a potência de resposta do operador de gradiente utilizado para produzir o pixel da borda e o ângulo do vetor gradiente.

A potência de resposta do operado de gradiente para uma imagem $f(x,y)$ é obtida utilizando a equação (1.3).

Sendo assim, um pixel da borda com coordenada (x_0, y_0), vizinho ao pixel (x,y), será similar ao pixel (x,y) se

$$|\nabla f(x,y) - \nabla f(x_0,y_0)| \leq E \tag{1.6}$$

onde E é um limiar não-negativo.

A segunda propriedade que define a similaridade entre os pixels de borda é o ângulo do vetor gradiente, definido como:

$$\alpha(x,y) = \tan^{-1}\left(\frac{G_y}{G_x}\right) \tag{1.7}$$

Da mesma forma que a similaridade por magnitude (1.6), um pixel da borda com coordenada (x_0, y_0), vizinho ao pixel (x,y), terá um ângulo similar ao do pixel (x,y) se

$$|\alpha(x,y)-\alpha(x_0,y_0)|<A \tag{1.8}$$

onde *A é* um limiar angular não-negativo.

Um pixel vizinho ao pixel (x,y) é similar a este se os critérios de magnitude e ângulo forem satisfeitos.

1.4.2.2 Transformada de Hough

Proposta como um método para detecção de padrões complexos em imagens binárias, a Transformada de Hough foi introduzida em 1962 pelo autor, o qual a transformada recebeu o nome, e publicada na forma de uma patente. Um dos objetivos do invento de Hough é a previsão de um método para reconhecimento de padrões complexos em uma fotografia; outro objetivo é propiciar um método e um meio aperfeiçoado para o reconhecimento de trilhas de partículas em fotografias obtidas em uma câmara de bolha (PEREIRA, 1995). No caso da transformada de Hough para linha reta, toma-se uma linha contida em um plano cartesiano de eixos X e Y e tem-se que uma reta pode ser representada pela equação:

$$y = mx + c \tag{1.9}$$

onde, *c* e *m* são os coeficientes linear e angular, respectivamente, da reta. Enquanto que, *x* e *y* são os valores da projeção do ponto sobre o eixo X e Y, respectivamente. A transformada de Hough para uma reta consiste em transladar os pontos do plano cartesiano *x*, *y* (imagem) para outro plano, o plano de parâmetros (ou espaço de parâmetros), isto é, mudá-los do plano "*x,y*" para o plano "*m,c*", através da equação (LOW, 1991):

$$c = -mx + y \tag{1.10}$$

Quando muda-se de um ponto do plano cartesiano para o plano de parâmetros, aquele ponto define um segmento de reta. Então é possível extrair as seguintes propriedades:

- Propriedade 1- Um ponto no plano cartesiano representa uma reta no plano de parâmetro;
- Propriedade 2- Um ponto no plano de parâmetros representa uma reta no plano cartesiano;
- Propriedade 3- Todos os pontos colineares no plano cartesiano apresentam um ponto comum de intersecção de retas no plano de parâmetros.

Na Figura 1.7 tem-se o plano cartesiano com uma reta determinada por três pontos, (1,3); (2,2); (3,1). Na Figura 1.8 é apresentado o plano de parâmetros onde surgem três retas, conforme Propriedade 1 acima citada, e percebe-se que os três segmentos de retas se interceptam no ponto $c=4$ e $m=-1$. As coordenadas deste ponto no plano de parâmetros (c,m) constituem os coeficientes linear e angular da equação de uma reta no plano cartesiano que é:

$$y = -1x + 4 \tag{1.11}$$

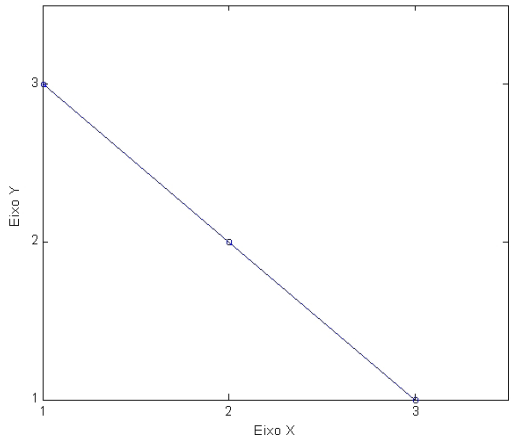


Figura 1.7 - Reta formada pelos pontos (1,3), (2,2) e (3,1) no plano cartesiano

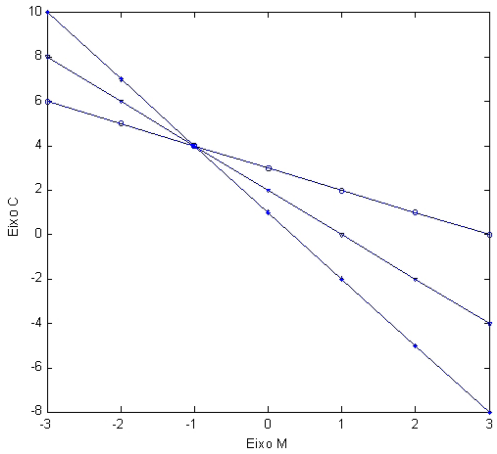


Figura 1.8 - Plano de parâmetros com a intersecção das três retas resultantes dos definidos na Figura 1.7

A proposta apresentada que mapeia uma linha usando os parâmetros Inclinação-Intersecção no espaço de parâmetros, tem uma desvantagem. Tanto a inclinação como a intersecção, são ilimitados e essa parametrização é sensível à escolha do eixo de coordenadas, no plano cartesiano, o que complica a aplicação da técnica.

Em 1972, Duda e Hart, propuseram uma solução para o problema da aplicação da técnica (DUDA & HART, 1973). Os autores utilizaram coordenadas polares para definir o segmento de reta, trabalhando com os parâmetros Ângulo-Raio ao invés de Inclinação-Intersecção.

Assim, uma reta pode ser representada por uma equação dada por duas variáveis denominadas ρ e θ, onde ρ é a menor distância algébrica entre a origem do plano cartesiano e a reta, e θ o ângulo entre o eixo x e o segmento "r". Conforme Figura 1.9.

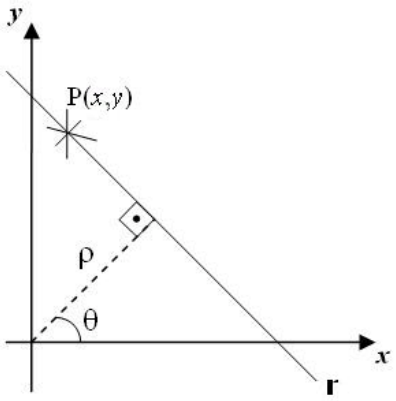


Figura 1.9 - Representação de ponto na forma paramétrica

Na Figura 1.9, r é uma das possíveis retas que passam através do ponto P(*x,y*); ρ é a menor distância da reta a origem e θ é o ângulo entre a distância e o eixo *x*. A equação de uma linha correspondente a esta geometria é:

$$\rho = x.\cos\theta + y.sen\theta \tag{1.12}$$

Porém, DUDA e HART (1973) lembram que esta equação não representa uma única reta. Como duas retas paralelas simétricas à origem possuem coeficientes angular es iguais e distâncias da reta à origem iguais, a Equação 1.12 pode representar duas retas, mas, sempre apresentam uma defasagem entre seus ângulos θ de 180 graus. Assim, se for restringido esse intervalo para a faixa angular de $0 \le \theta < 180$ força esta equação a representar uma única reta.

1.4.3 *Segmentação por crescimento de região:*

Outra técnica utilizada para segmentar uma imagem digital em regiões consiste em, a partir de um conjunto de pixels sementes, agregar a esta semente pixels vizinhos com propriedades similares.

Seja R a região total da imagem e R_i as regiões em que a imagem é dividida, as seguintes propriedades dever ser verdadeiras:

- $R_1 \cup R_2 \cup ... \cup R_i = R$
- R_i é uma região conexa, $i = 1, 2, ..., n$
- $R_i \cap R_j = \phi$ para todo *i* e *j*, $i \neq j$
- $P(R_i)$ = verdadeiro para todo $i = 1, 2, ..., n$
- $P(R_i \cup R_j)$ = falso para todo $i \neq j$

onde $P(R_i)$ é um predicado lógico definido sobre todos os pixels da região R_i, como por exemplo, pixels com tons de cinza entre 100 e 200.

O objetivo é gerar regiões conectadas e uniformes a partir de cada pixel semente.

Um pixel é adicionado a uma região se:

- Se ele não estiver designado para outra região;
- Se ele for vizinho desta região;
- Se a região criada com a adição do novo pixel continuar uniforme;

1.5 Classificação Bayesiana

Classificadores probabilísticos, em particular classificadores Bayesianos são tipos de classificadores comumente utilizados pela sua praticidade e relativa eficiência em suas aplicações.

A classificação é feita buscando minimizar o risco de Bayes para teste de hipóteses. O risco de Bayes é definido, segundo Scharf (SCHARF, 1991; MASCARENHAS *et al*, 2000), como uma média da função de perda, λ_{ij}, que corresponde ao custo de classificar uma região como pertencente à classe *i*, sendo esta, na realidade, pertencente à classe *j*.

Sendo assim, a estratégia de decisão para a classificação consiste em minimizar a perda condicional média, dada por:

$$L_x(w_j) = \sum_{j=1}^{m} \lambda_{ij} p(w_j \mid X) \tag{1.13}$$

onde, $\{ w_j, j = 1,\ldots, \text{m} \}$ é o conjunto de m classes; X é a observação e $p(w_j \mid X)$ é a probabilidade condicional da classe ser w_j, dada a observação X.

A observação X é uma variável aleatória que representa os possíveis resultados do experimento. Podendo esta ser discreta ou contínua e será detalhada mais à frente.

Definindo o custo λ_{ij} como uma função de perda simétrica (zero-um):

$$\lambda_{ij} = \begin{cases} 0, i = j \\ 1, i \neq j \end{cases} \tag{1.14}$$

então (1.9) torna-se:

$$L_x(w_j) = 1 - p(w_j \mid X) \tag{1.15}$$

e $L_x(w_j)$ será mínimo quando a probabilidade a posteriori, $p(w_j \mid X)$, for máxima.

Desse modo, a regra de decisão pode ser expressa por:

$$X \in w_i \Leftrightarrow p(w_i \mid X) = \max p(w_j \mid X) \tag{1.16}$$

ou seja, para os custos dados em (1.14), a maior probabilidade a posteriori indica a classe a ser escolhida.

Partindo de (1.16), Lee e colaboradores (LEE *et al*, 1987) definem a função de adesão para um conjunto de n observações, ou seja, $X = \{x_1, ..., x_n\}$. Deste modo tem-se que:

$$p(w_j \mid X) = p(w_j \mid x_1, ..., x_n) \tag{1.17}$$

Mas $p(w_j \mid X)$ pode ser escrito como

$$p(w_j \mid X) = \frac{p(X \mid w_j) p(w_j)}{p(X)} = \frac{p(x_1, ..., x_n \mid w_j) p(w_j)}{p(x_1, ..., x_n)} \tag{1.18}$$

onde $p(X \mid w_j)$ é a densidade condicional, $p(w_j)$ é a probabilidade a priori e $p(X)$ é dado por:

$$p(X) = \sum_{j=1}^{n} p(X \mid w_j)\, p(w_j) \tag{1.19}$$

O custo de classificação é baseado na regra de Bayes, que é utilizada para estimar a probabilidade condicional de uma determinada classe, a partir da qual é adotado um conjunto de restrições, como independência entre as observações (DUDA & HART, 1973), que decompõe esta probabilidade em produtos de probabilidades condicionais, tendo então:

$$p(w_j \mid X) = p(w_j) \prod_{k=1}^{n} \frac{p(x_k \mid w_j)}{p(x_k)} \tag{1.20}$$

Como $p(X \mid w) = \dfrac{p(w \mid X)\, p(X)}{p(w)}$, então a Função de Adesão é definida como:

$$p(w_j \mid X) = p(w_j)^{1-n} \prod_{k=1}^{n} p(w_j \mid X) \tag{1.21}$$

Dentre as condições assumidas pela regra de Bayes, além da independência entre as observações, assume-se que todas as observações têm o mesmo peso, ou seja, influenciam igualmente no resultado (GARG & ROTH, 2001).

Esta última condição vem sendo questionada, uma vez que entre um conjunto de observações pode haver observações "não-confiáveis", como observações ruidosas, por exemplo.

Como solução, vem sendo proposta a adoção de graus de confiança das observações, que devem atuar de maneira tal que quando a confiança de uma observação é baixa, o efeito desta no resultado final seja diminuído e quando a confiança de uma observação é alta, seu efeito é realçado (MÁXIMO & FERNANDES, 2003).

1.5.1 *Variáveis aleatórias*

Como dito anteriormente, os resultados do experimento são comumente representados por uma variável aleatória. Isso se dá por conveniência ou devido à própria natureza quantitativa do evento.

Variável aleatória é uma função definida em um espaço amostral e que assume valores numéricos. Podendo ser discreta, quando admite valores enumeráveis, ou, as freqüentemente utilizadas, variáveis aleatórias contínuas (BEKMAN & NETO, 1980), em que a distribuição de probabilidade é caracterizada por uma função de densidade de probabilidade *h(x)* adiante.

Em termos gerais, para as variáveis discretas, sua média $\mu(X)$ é definida por:

$$\mu(X) = \sum_i x_i P(x_i) \tag{1.22}$$

onde $P(x_i)$ é a probabilidade da variável aleatória X da ocorrência x_i.

Outro conceito importante sobre variáveis aleatórias é o da variância $\sigma^2(X)$ definida como:

$$\sigma^2(X) = \sum_i x_i{}^2 P(x_i) - [\mu(X)]^2 \tag{1.23}$$

A variância dá idéia da extensão da variação da variável, podendo também ser utilizada para tal sua raiz quadrada, o desvio-padrão $\sigma(X)$, cuja dimensão é a mesma da variável aleatória.

Para as variáveis aleatórias contínuas, a média e a variância podem ser determinadas, respectivamente, por:

$$\mu(X) = \int_{-\infty}^{+\infty} x h(x) dx \tag{1.24}$$

onde *h(x)* é a função densidade de probabilidade.

$$\sigma^2(X) = \int_{-\infty}^{+\infty} x^2 h(x)dx - [\mu(X)]^2 \tag{1.25}$$

1.5.2 *Distribuição de probabilidades*

A distribuição de probabilidade pode ser definida de várias formas, porém, algumas distribuições específicas serão citadas por ser de maior importância teórica e prática (BEKMAN & NETO, 1980).

1.5.2.1 Distribuição binomial

Em uma situação em que são realizadas *n* provas independentes, cada uma delas podendo levar apenas a sucesso ou fracasso (prova de Bernoulli), sendo a probabilidade *p* de sucesso em cada prova constante a variável aleatória discreta.

X = número de sucessos obtidos nas *n* provas

Tem distribuição de probabilidade binomial, podendo-se verificar que:

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k},\ k = 0,1,2,...,n \tag{1.26}$$

onde $\binom{n}{k}$ é o número de combinações de *n* elementos tomados *k* a *k*, e dado por:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n(n-1)...(n-k+1)}{k(k-1)...3,2,1} \tag{1.27}$$

Desta forma, para a distribuição binomial, obtêm-se como média e variância, respectivamente:

$$\mu(X) = np \tag{1.28}$$

$$\sigma^2(X) = np(1-p) \tag{1.29}$$

1.5.2.2 Distribuição de Poisson

A variável aleatória discreta

X = número de ocorrências em um intervalo contínuo de observação t

terá distribuição de Poisson com $\mu(X) = \sigma^2(X) = \lambda t$, onde λ é a freqüência média de ocorrência, supostamente constante, se atender as seguintes condições:

- Em intervalos de observações muito pequenos, a probabilidade de mais uma ocorrência é desprezível.
- Em intervalos de observação muito pequenos, a probabilidade de uma ocorrência é proporcional ao tamanho do intervalo.
- As ocorrências em intervalos sem ponto comum dão-se independentemente.

Sendo assim, verifica-se que:

$$P(X = k) = \frac{\mu^k e^{-\mu}}{k!}, k = 0,1,2,3,... \tag{1.30}$$

1.5.2.3 Distribuição exponencial

É uma distribuição contínua de probabilidade definida pela função densidade

$$h(x) = \begin{cases} 0 & para \quad x < 0 \\ \lambda e^{-\lambda x} & para \quad x \geq 0 \end{cases} \tag{1.31}$$

O aspecto característico dessa função é mostrada na Figura 1.10

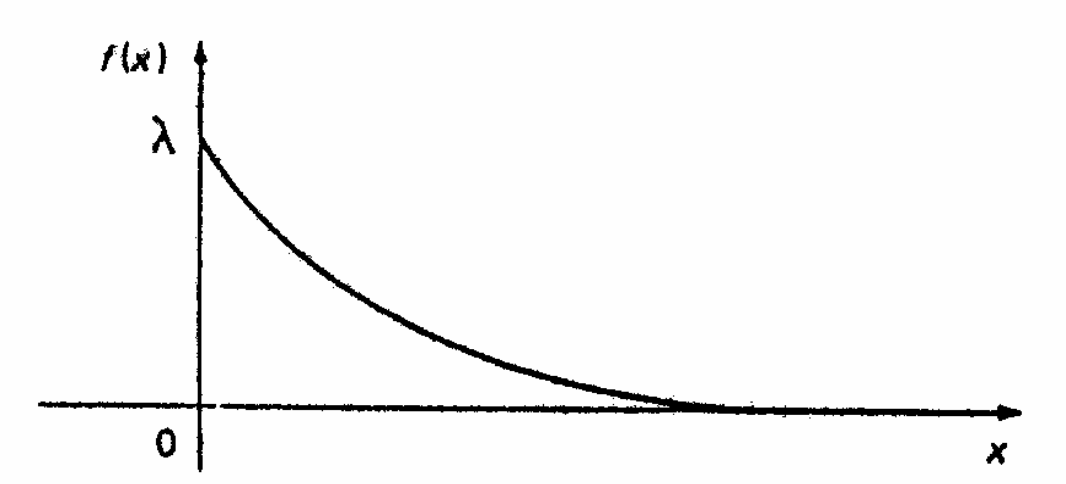


Figura 1.10- Curva característica da distribuição exponencial

1.5.2.4 Distribuição Multivariada Gaussiana

Densidades multivariadas gaussianas têm sido freqüentemente utilizadas para caracterizar classes. A forma geral dessa densidade é dada por:

$$h(x) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu)^t \boldsymbol{\Sigma}^{-1}(x-\mu)\right] \tag{1.32}$$

onde x é o vetor (d x 1) de atributos, μ é o vetor (d x 1) de médias, $\boldsymbol{\Sigma}$ é a matriz (d x d) de covariância, $(x-\mu)^t$ é a transposição de $x-\mu$, $\boldsymbol{\Sigma}^{-1}$ é a inversa de $\boldsymbol{\Sigma}$, e $|\boldsymbol{\Sigma}|$ é o determinante de $\boldsymbol{\Sigma}$.

A densidade multivariada normal é especificada pelos parâmetros $d + d(d+1)/2$, os elementos do vetor de médias μ e os elementos independentes da matriz de covariância $\boldsymbol{\Sigma}$. Amostras tiradas de uma população normal tendem a se juntar em um único agrupamento, como mostra a Figura 1.11.

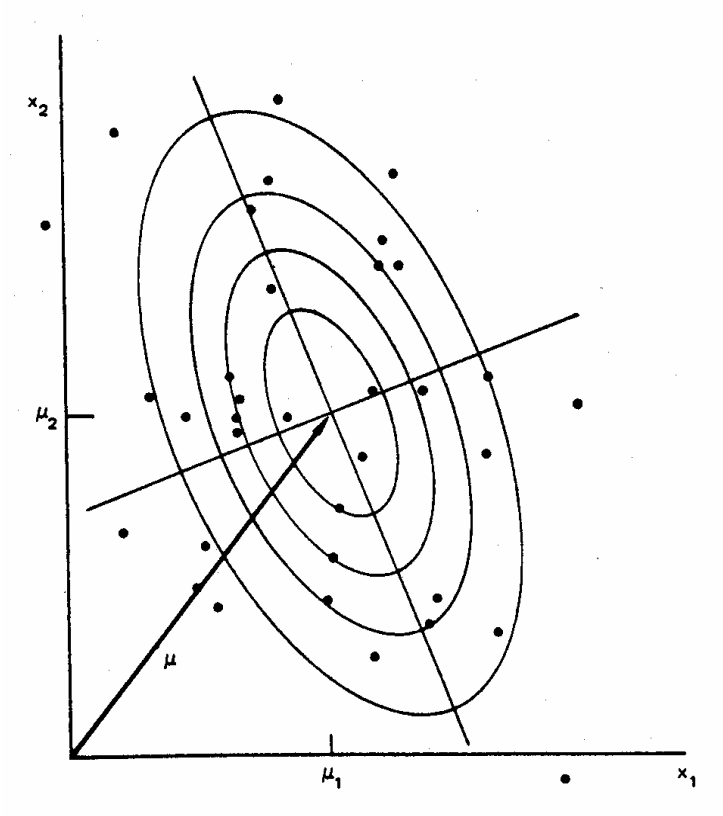


Figura 1.11 - Agrupamento de amostras com distribuição de Gauss

Na Figura 1.11 o centro do agrupamento é determinado pelo vetor de médias e a forma do agrupamento é determinada pela matriz de covariância. No caso multidimensional essas curvas de nível são elipsóides. Os eixos principais desse elipsóide são os vetores próprios de $\boldsymbol{\Sigma}$ e os valores próprios determinam o comprimento desses eixos. A equação (1.32) representa a expressão analítica dessas elipsóides.

$$r = (x - \mu)^t \, \Sigma^{-1} (x - \mu) \tag{1.33}$$

A quantidade r representa a chamada distância de Mahalanobis de x a μ.

1.5.3 *Função discriminante*

Existem várias formas de se representar classificadores de padrões (DUDA & HART, 1973). Uma delas é pelo uso de funções discriminantes $g_i(x),\ i = 1,...,c$. O classificador atribui um vetor de características *x* a uma classe ω_i se:

$$g_i(x) > g_j(x), \qquad \text{para todo} \qquad j \neq i \tag{1.34}$$

O classificador é visto como uma máquina que calcula *c* funções discriminantes e seleciona a classe que corresponde ao maior discriminante, como ilustra a Figura 1.12.

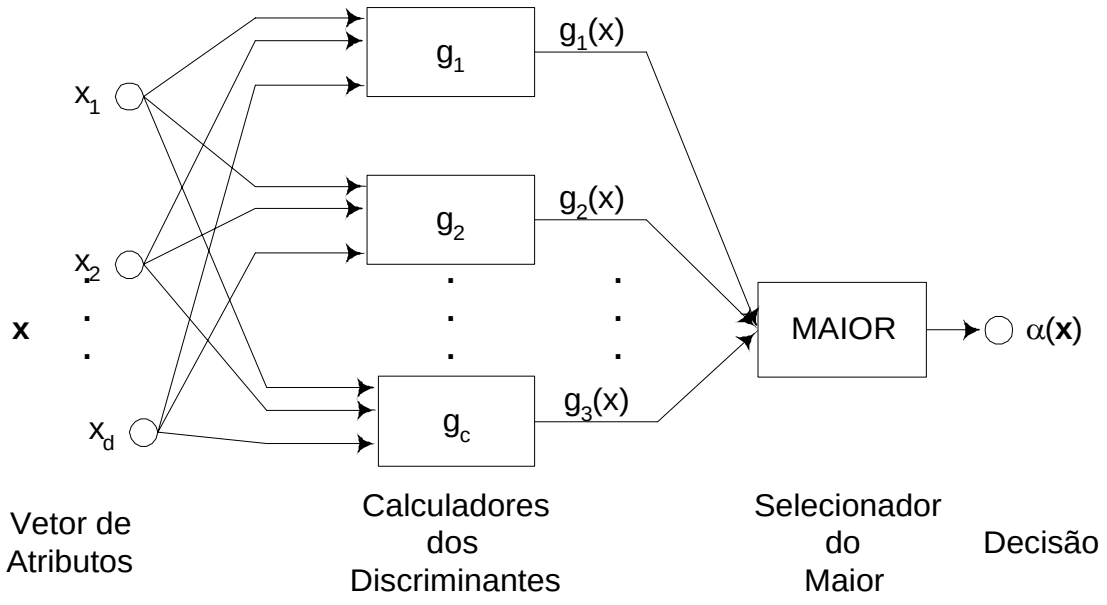


Figura 1.12 – Classificador de padrões

Para o caso geral, considerando $g_i(x) = -R(\alpha_i \mid x)$, de forma que o maior resultado das funções discriminantes corresponderá ao menor risco condicional. Para o caso MAP, pode–se considerar $g_i(x) = P(w_i \mid x)$, o maior resultado das funções discriminantes corresponderá à maior probabilidade *a posteriori*.

Utilizando-se a regra de Bayes com a equação (1.18), e como o denominador é independente de *j* pode-se atribuir o vetor *x* que maximize a expressão $p(x|w_j)P(w_j)$. Pode-se dessa maneira verificar que o processo de classificação é dado pelo cômputo das funções discriminantes:

$$g_i(x) = p(x|w_i)P(w_i), \qquad i = 1, 2, \ldots, c \tag{1.35}$$

para cada vetor de atributos *x* observado e pela atribuição desse vetor à classe w_j que forneça a máxima função discriminante.

Para a classificação por taxa mínima de erro, qualquer uma das expressões vistas anteriormente fornece um resultado idêntico. Também, é possível tomar a decisão computando-se as funções discriminantes pela função monotonicamente crescente de $g_i(x)$:

$$\log_e g_i(x) = \log_e p(x|w_j) + \log_e P(w_j) \tag{1.36}$$

É freqüente a situação em que se admitem as probabilidade *a priori* $P(w_i)$, $i = 1, 2, \ldots, c$. Nesse caso as funções discriminantes se resumem às funções densidade de probabilidade condicionais $p(x|w_i)$. Tal esquema de decisão é denominado decisão por máxima verossimilhança (conhecida como MAXVER). Em outras palavras, para a taxa mínima de erro, decide w_i se:

$$p(x|w_i) > p(x|w_j), \quad \text{para todo} \quad j \neq i \tag{1.37}$$

Para o método desenvolvido, utilizou-se a técnica de segmentação orientada à região. Como método de reconhecimento de padrões, utilizou-se Transformada de Hough e

para classificação foi utilizado Classificação Bayesiana, utilizando como medida entre as probabilidades a distância de Mahalanobis.

No próximo capítulo será visto os detalhes do padrão que foi adotado como forma de armazenamento, ou seja, o padrão DICOM.

2 O PADRÃO DICOM E SUA UTILIZAÇÃO EM IMAGENS DIGITAIS PARA FINS DE DIAGNÓSTICOS

O Padrão DICOM (*Digital Imaging and Communication in Medicine*) surgiu quando, nos anos 70, com a ascensão da computação em aplicações clínicas e a adoção de sistemas digitais para diagnóstico em imagens médicas, como a tomografia computadorizada, os dispositivos desenvolvidos para este fim não eram capazes de trocar informações quando de fabricantes diferentes.

Foi então que membros da ACR (*American College of Radiology*), juntamente com os fabricantes de equipamentos médicos, NEMA (*National Eletrical Manufactures Association*) formaram a comissão ACR-NEMA, em 1983 que tinha como objetivo desenvolver um padrão para:

- Promover a troca de informações de imagens entre dispositivos, independente do fabricante;
- Facilitar o desenvolvimento e expansão de sistemas de armazenamento e comunicação de imagens (PACS – Picture archiving and Communication System), além de permitir que estes se comuniquem também com os sistemas hospitalares;
- Permitir a criação de bases de dados para informações de diagnósticos que possam ser utilizados por dispositivos geograficamente distribuídos.

Em 1985 foi publicado o ACR-NEMA 300-1985 (também chamado de ACR-NEMA versão 1.0), primeiro resultado obtido da análise de inúmeras interfaces já existentes, como uma especificação que já vinha sendo adotado pela AAPM (*American Association of Physicists in Medicine*), que propusera um padrão para o armazenamento de

imagens em meio magnético. O formato utilizado pela AAPM era constituído por um cabeçalho contendo a descrição da imagem e outras informações relevantes, como nome do paciente. Alguns conceitos como este, de utilizar elementos de dados de tamanho variável, identificados por uma *tag* ou *key* foram incorporados ao padrão.

O padrão passou por duas revisões nos três anos seguintes, e em 1988 foi publicado o ACR-NEMA versão 2.0 (ACR-NEMA 300-1988) que corrigia inconsistências e erros da versão anterior, além de prover especificação para dispositivos de visualização, a introdução de um novo esquema de hierarquia para facilitar a identificação das imagens, inserção de novos elementos de dados, especificação de uma interface de *hardware* e um conjunto de comandos básicos de *software*.

Após três anos, a segunda versão passou por drásticas mudanças que vinham a adaptar o padrão em um ambiente distribuído, pois as versões anteriores utilizavam alguns conceitos de sistemas distribuídos, porém, somente com conexões ponto-a-ponto. Para um ambiente de rede foi criada o ACN-NEMA DICOM, também chamado de DICOM 3.0, que trazia novos métodos para troca de informações:

- Suporte a operações em ambientes de rede utilizando o protocolo TCP/IP;
- Especificação que não eram abordadas pelos dois ACN-NEMA anteriores, como um formato de arquivo, meios físicos como CD-R, por exemplo, e sistema de arquivos, como ISO 9660 e sistema de arquivo PC (FAT-16), que disponibilizava a troca de informações entre sistemas desconectados;
- Troca não somente de dados, como nas versões anteriores. Adotou-se conceitos como Classes de Serviços (*Service Classes*) que possibilitava a troca dos dados e métodos para a manipulação dos dados associados.

A especificação do padrão DICOM foi dividida em múltiplas partes para facilitar sua evolução e simplificar sua manutenção (NEMA, 2003).

2.1 O padrão DICOM

O padrão DICOM foi desenvolvido seguindo diretivas ISO que definem como estruturar documentações em múltiplas partes. Desta forma, o padrão DICOM é constituído de dezesseis partes.

2.1.1 *Partes constituintes*

2.1.1.1 Parte 1 (PS 3.1) – Introdução e Visão geral

É dada uma visão geral do padrão DICOM, além de citar as normas adotadas pelo padrão, são definidos os conceitos adotados, símbolos e abreviações, objetivos e metas do padrão e é dada uma breve descrição das partes restantes.

2.1.1.2 Parte 2 (PS 3.2) – Conformidade

Conformidade

Definição formal associada a uma implementação específica do padrão DICOM. Definição das Classes de Serviços (*Service Classes*), Objetos de Informações (*Information Objects*), protocolos de comunicação e mídias de armazenamento suportadas pela implementação, e medidas de seguranças suportadas pela aplicação.

Não são definidos pelo padrão:

- Método para teste ou validação para verificar se uma implementação segue as especificações DICOM
- Método para teste ou validação que verifique se a implementação segue o que foi definido pela conformidade do projeto
- Quais classes de serviços, objetos de informações devem ser suportados por um determinado dispositivo.

As Figura 2.1 e Figura 2.2 ilustram o processo de definição da conformidade de uma implementação para comunicação em rede e troca de informações em meio magnético, respectivamente.

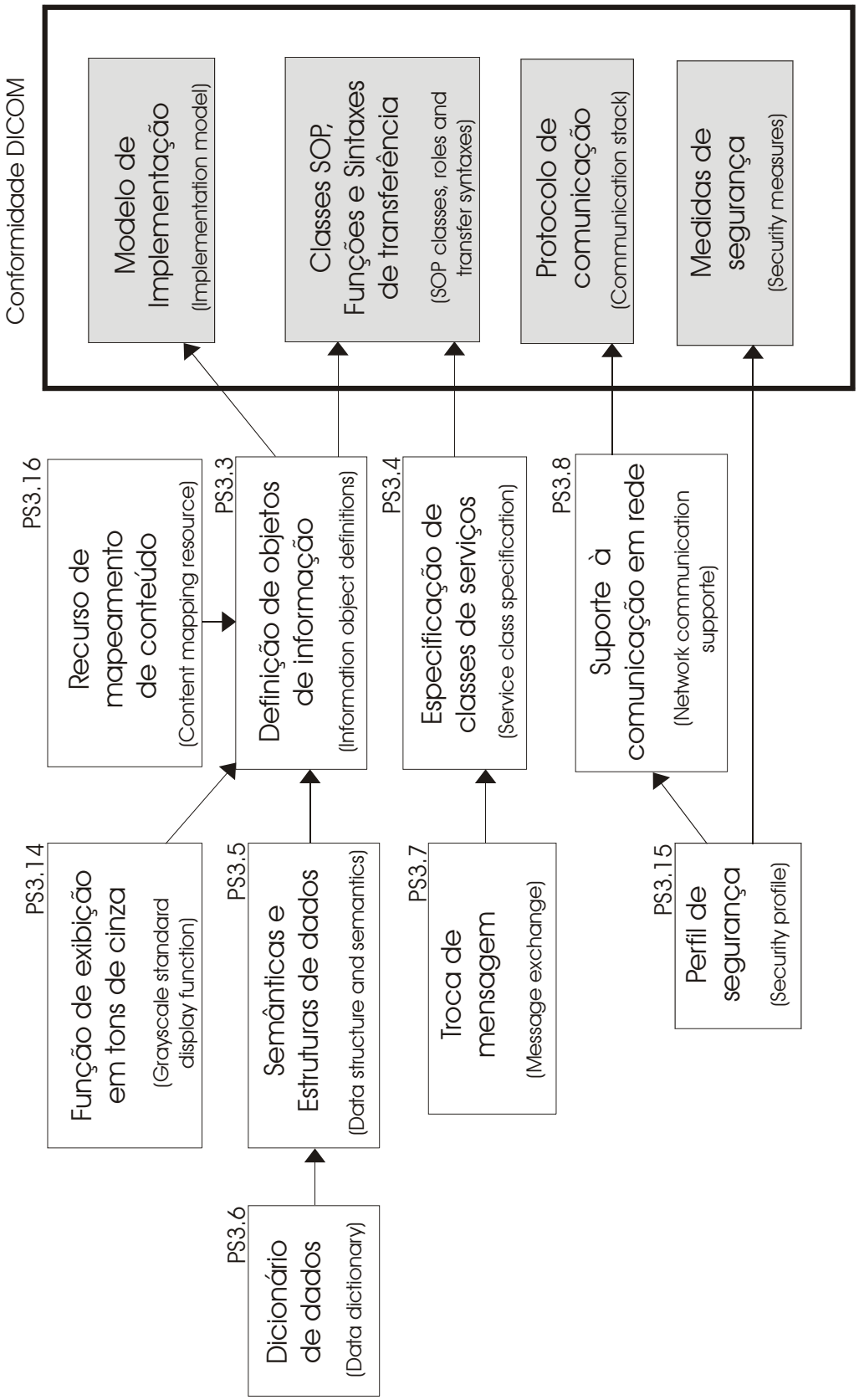


Figura 2.1 - Processo de construção da conformidade em ambiente de rede

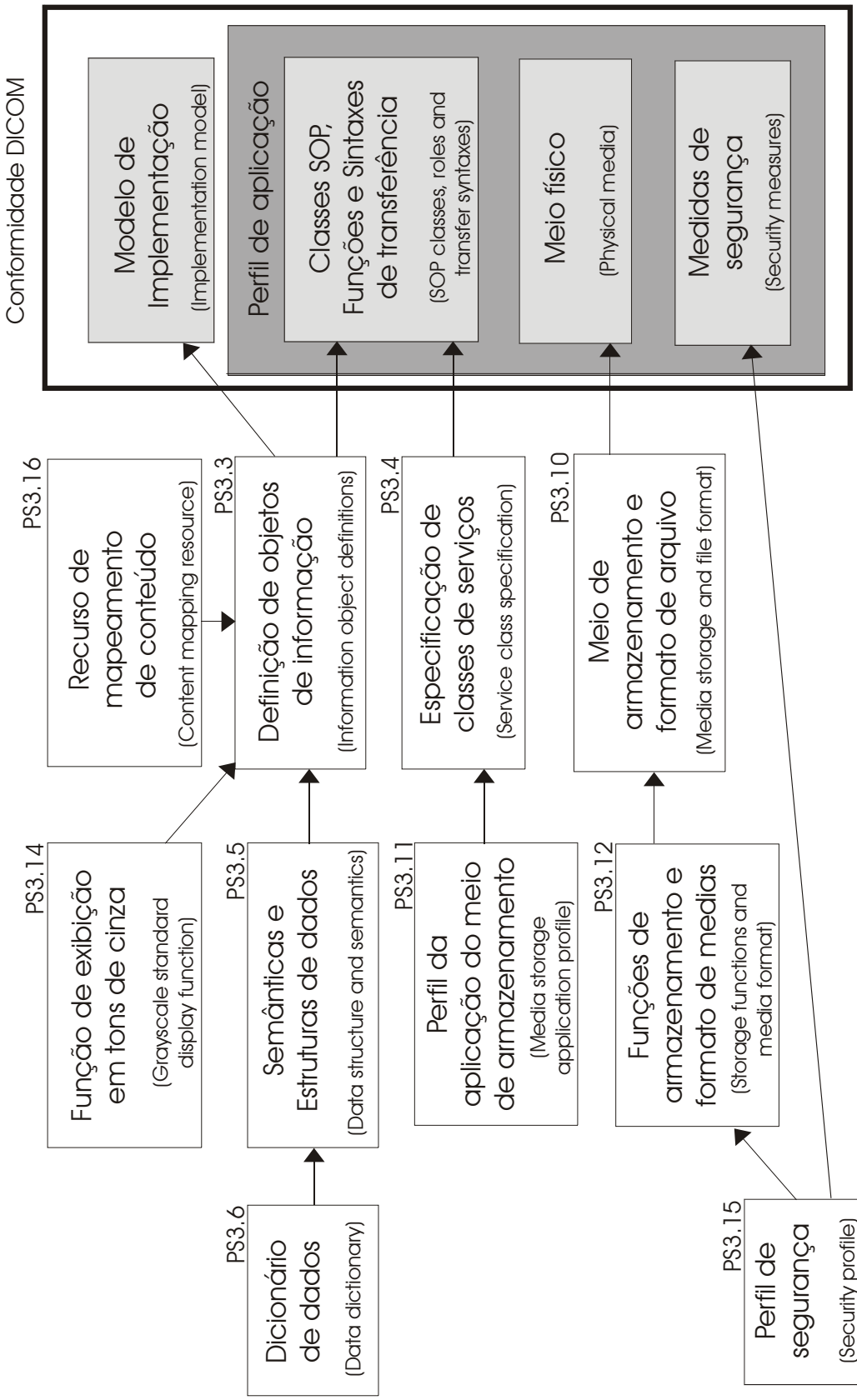


Figura 2.2 - Processo de construção da conformidade em meio magnético

2.1.1.3 Parte 3 (PS 3.3) – Definição dos Objetos de Informação

Na parte 3 do padrão são definidas as classes do objeto de informação (*Information Object Classes*) que oferecem uma definição das entidades do mundo real aplicadas na comunicação de imagens médicas digitais. Cada Classe de Objeto de Informação consiste na descrição de sua função (propósito) e de seus atributos que a constituem.

Duas classes de objeto de informação são definidas:

- ***Classes de Objeto de Informação Normalizadas:*** Inclui somente informações inerentes à entidade do mundo real que a classe se propõe a representar. Como exemplo, pode-se citar a classe de Exame, definida como normalizada, que possui atributos como data do exame, hora do exame e qualquer outra informação inerente ao exame; porém nome do paciente não pertence à classe Estudo, mas sim à classe Paciente.
- ***Classes de Objeto de Informação Compostas:*** Inclui informações que possuem alguma relação com a entidade do mundo real que a classe em questão representa. Por exemplo, a classe de Imagem de Tomografia Computadorizada, definida como composta, possui informações inerentes à imagem e também atributos como nome do paciente, que está relacionada à imagem, mas não é inerente à mesma.

As classes compostas oferecem uma estrutura que atende às necessidades da comunicação de imagens, no qual tanto os dados da imagem quanto os dados relacionados a ela precisam estar juntamente associados.

Como forma de organização, tais atributos podem ser agrupados. Estes grupos de atributos são especificados de forma independente, podendo ser utilizados por outras classes compostas.

Para operar sobre os atributos de um objeto de informação, mantendo assim, o objeto em questão consistente com a entidade do mundo real que está sendo representada, são necessários serviços que manipulem tais atributos. Estes serviços são definidos pela classe de Objeto de Serviços, que será vista a seguir.

2.1.1.4 Parte 4 (PS 3.4) – Especificação de Classes de Serviços

Possui especificações da Classe de Serviço. Uma Classe de Serviço é responsável por estabelecer a relação entre os Objetos de Informação e os Comandos (*commands*) que manipulam tais informações.

Nesta parte, são estabelecidos as características e comportamentos que toda Classe de Serviço deve possuir e como a conformidade pode ser conseguida para uma dada Classe de Serviço.

2.1.1.5 Parte 5 (PS 3.5) – Estrutura de Dados e Semânticas

Inclui especificações de como as aplicações constroem e codificam o conjunto de dados obtidos pelos Objetos de Informação e Classes de Serviços, definidos nas partes 3 e 4 do padrão, respectivamente.

O objetivo nesta parte, é estabelecer as regras para construção dos *Streams* de dados que serão transmitidos em uma mensagem, como especificado na parte 7, detalhada mais adiante.

É definido também suporte para algumas técnicas de compressão de imagem, como por exemplo, JPEG com e sem perda.

É definida ainda a semântica das funções principais que são utilizadas sobre os Objetos de Informação, além da definição do conjunto de caracteres utilizados, como forma de internacionalização do uso do padrão.

2.1.1.6 Parte 6 (PS 3.6) – Dicionário de dados

Define a coleção de todos os Elementos de Dados utilizados pelo padrão para representar alguma informação. Com indicação dos seus identificadores numéricos (*tags*), nomes, tipo de valor representado, como cadeia de caracteres, inteiros, etc. Informações como multiplicidade (se há ou não repetição de um Elemento de Dados), valores que são

permitidos um determinado Elemento, caso haja alguma restrição, e se o elemento permite valor nulo ou não, são também abordadas nesta parte.

2.1.1.7 Parte 7 (PS 3.7) – Troca de mensagem

Especifica o serviço e protocolo usado pela aplicação do ambiente de imagiamento médico para a troca de mensagem utilizando os serviços dos meios de comunicação estabelecidos na parte 8, a ser definida a seguir. Uma mensagem é composta por um *Stream* de comando, seguido do *Stream* de Dados opcional, como definido na parte 5.

São especificados:

- regras para o estabelecimento e finalização da comunicação
- regras para a troca de mensagem do tipo requisição/resposta

2.1.1.8 Parte 8 (PS 3.8) – Suporte à comunicação em Rede para troca de mensagem

Define os serviços e protocolos utilizados para a troca de mensagens entre aplicações DICOM em ambientes de redes, como OSI e TCP/IP. Estes serviços e protocolos garantem a perfeita comunicação entre as aplicações DICOM na rede.

2.1.1.9 Parte 9 (PS 3.9) – Suporte à comunicação ponto-a-ponto para troca de mensagem

Meio de comunicação obsoleto que é mantido no padrão somente por questões de compatibilidade com versões anteriores. Definia o uso de comunicação direta utilizando uma interface de 50 pinos (obsoleto).

2.1.1.10 Parte 10 (PS 3.10) – Meio de armazenamento e formato de arquivo

Define um modelo geral para o armazenamento de imagens médicas em meios magnéticos. São especificados:

- um modelo em camadas para o armazenamento de imagens médicas em meio físico , como é mostrado na Figura 2.3;
- formato de arquivo DICOM capaz de armazenar qualquer Objeto de Informação;
- algum nível de segurança sobre os dados armazenados, como criptografia;
- método de identificação de um conjunto de arquivos em um único meio físico de armazenamento;
- método para nomeação do arquivo DICOM, dado um sistema de arquivo específico.

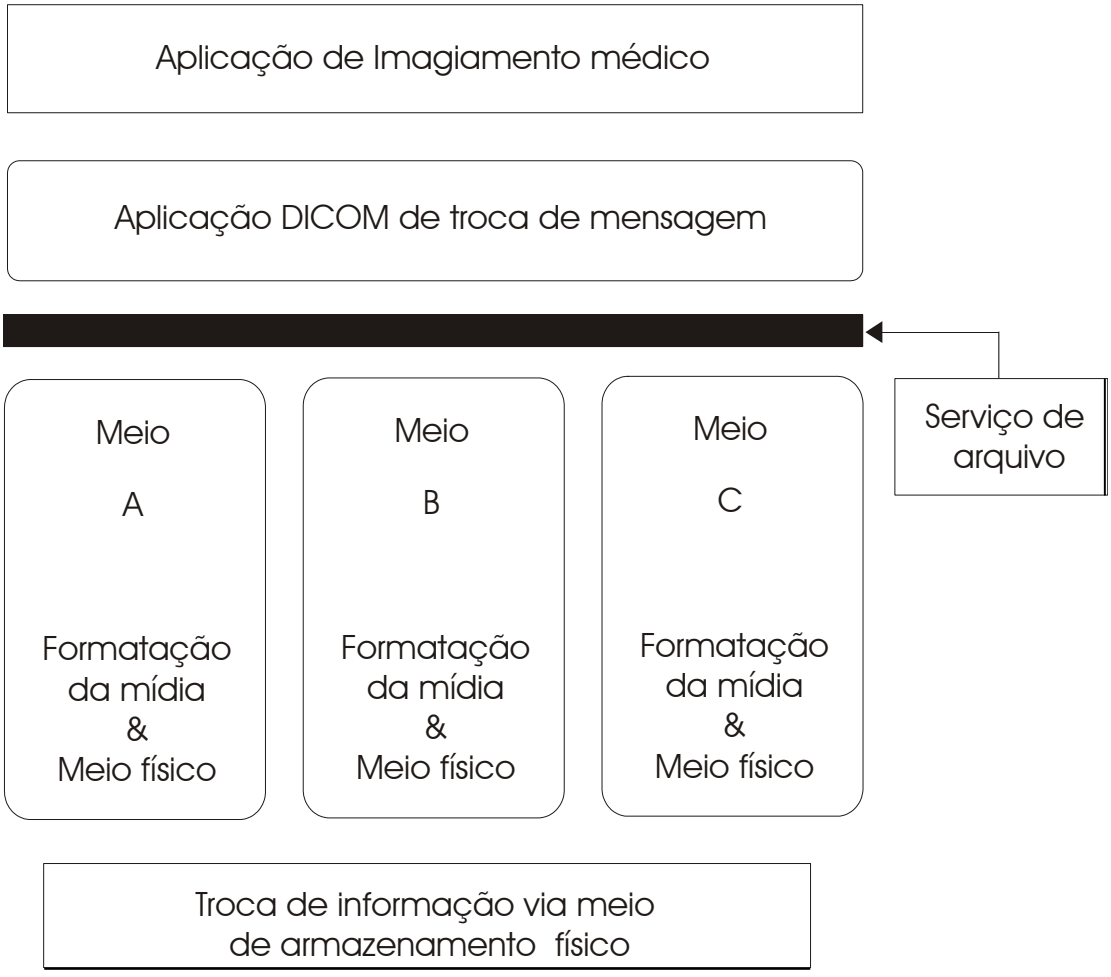


Figura 2.3 - Modelo em camadas para o sistema de armazenamento DICOM

2.1.1.11 Parte 11 (PS 3.11) – Perfil da aplicação do meio de armazenamento

Define uma forma dos usuários do padrão especificarem a seleção do meio de armazenamento entre os existentes na parte 12 e os Objetos de Informação, definidos na parte 3.

2.1.1.12 Parte 12 (PS 3.12) – Funções de armazenamento e formato de medias para transmissão de dados

Contém referências às especificações da indústria para os meios físicos de armazenamento e para os sistemas de formatação dos arquivos destes meios. Especifica, como principais meios de armazenamento, CD-R 650 Mb, 5.25" MOD 650 Mb, 5.25" MOD 1.3 Gb, 3.25" MOD 128 Mb e 3.5" Floppy Disk.

2.1.1.13 Parte 13 (PS 3.13) – Suporte a serviços de impressão em comunicação ponto-a-ponto

Especifica serviços e protocolos para impressão em comunicação ponto-a-ponto. Assim como a parte 9, que especifica tal comunicação, esta só é mantida por motivos de manter compatibilidade com versões anteriores.

2.1.1.14 Parte 14 (PS 3.14) – Função para exibição em tons de cinza

Define funções padronizadas para exibição de imagens em tons de cinza. Provê métodos para calibração para exibição de uma imagem de uma forma consistente, independente do dispositivo de exibição (monitor ou impressora, por exemplo).

A função de exibição é baseada na percepção visual humana.

2.1.1.15 Parte 15 (PS 3.15) – Aspectos de segurança

Provê algum nível de segurança utilizando padrões de segurança desenvolvido aparte, que vão desde utilização de chave pública até utilização de *smart cards*.

Não são definidas políticas de segurança, somente mecanismos que podem ser implementados.

2.1.1.16 Parte 16 (PS 3.16) –Recurso para mapeamento de conteúdo

Especifica modelos para estruturação de documentos como Objetos de Informação do padrão DICOM, conjunto de termos utilizados nos Objetos de Informação, tradução dos termos para um idioma específico.

No trabalho desenvolvido foram implementados os módulos referentes ao armazenamento em meio físico, deixando para trabalhos futuros a implementação dos módulos referentes à utilização do padrão em ambientes de rede.

A seguir, é detalhado o método desenvolvido e são apresentados estudos de caso de cinco pacientes selecionados de forma aleatória, os quais ilustram a aplicação do método e o uso da ferramenta.

3 MÉTODO PARA DIAGNÓSTICO PRECOCE DE MICRO-ESTRUTURAS DENTÁRIAS COM O USO DE TÉCNICAS DO PROCESSAMENTO DIGITAL DE IMAGENS

Neste capítulo é apresentada a plataforma para análise de microestruturas decorrentes da formação de cáries dentárias para seu diagnóstico precoce.

Técnicas do processamento de imagens, análise visual e modelagem matemática foram utilizadas para auxiliar na análise de microestruturas reconhecidas em imagens radiográficas de raios X, bem como técnica de classificação Bayesiana para a indicação do grau de desenvolvimento das estruturas diagnosticadas.

No desenvolvimento foi adotado como método de armazenamento o padrão DICOM 3.0, que define não somente a forma de armazenamento como a utilização de meta-informações referentes às imagens e a especificação de uma arquitetura para sistemas distribuídos aplicados à imagiologia e instrumentação nas áreas médica e odontológica, que encontram aplicação tanto para área humana como à pecuária.

Como ferramenta de implementação está sendo utilizada a linguagem Sun® Java, que além de favorecer a migração para outra plataforma, caso necessário, permite a geração de códigos tanto para aplicações *desktop* quanto para aplicações voltadas para o uso na *Web*. Dessa forma o uso de conteúdo distribuído, amparado pelo padrão DICOM, é satisfeito não somente pelo padrão adotado em si, mas também pela sua utilização em pontos de acesso que não possuam a aplicação instalada, mas somente navegadores de Internet.

A Figura 3.1 ilustra o diagrama esquemático básico do sistema, onde são observados os módulos para recepção e aquisição de imagens, segmentação, classificação e diagnóstico para auxílio à tomada de decisão. O módulo marcado em laranja foi implementado utilizando a ferramenta MATLAB e para os demais módulos, utilizou-se o Java.

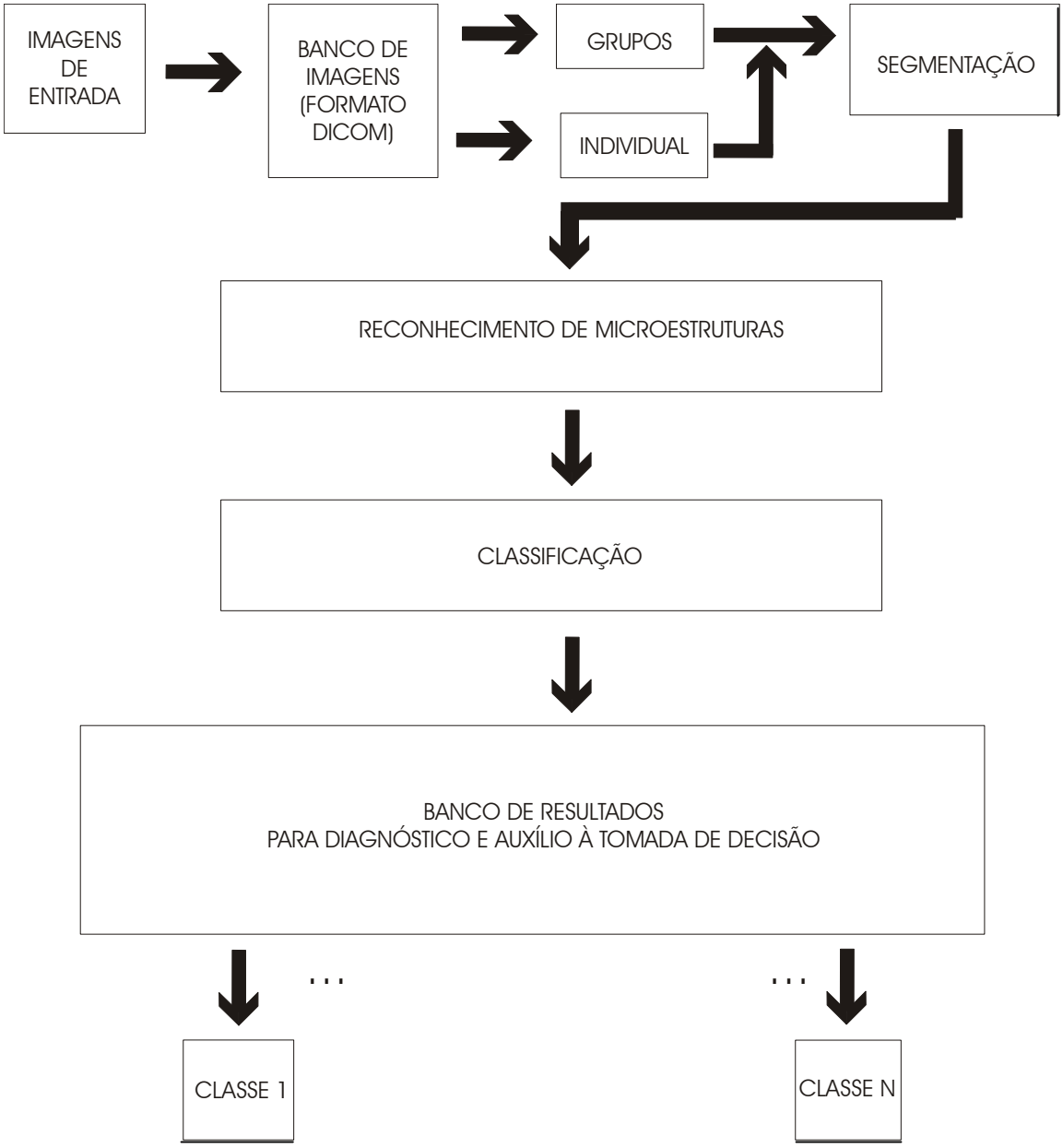


Figura 3.1 - Diagrama esquemático básico do sistema de caracterização de microestruturas em imagens digitais obtidas com técnicas de Raios X

Os blocos da Figura 3.1 são detalhados a seguir, sendo alguns destes detalhamentos seguidos de seus algoritmos:

- *Imagens de entrada:* Processo de aquisição das imagens que serão submetidas à análise.

- *Banco de imagens (Formato DICOM):* Nesta fase as imagens que são dadas como entrada ao sistema são convertidas para o formato DICOM 3.0 com auxílio do usuário que deve informar os dados referentes às mesmas, baseando-se no algoritmo ilustrado na Figura 3.2:

Leitura da imagem de entrada
Instância da classe de informação
Entrada dos dados pertinente à imagem
Compressão da imagem (quando aplicável)
Encapsulamento do conjunto de dados
Escrita em arquivo

Figura 3.2 - Algoritmo do módulo "Banco de imagens"

- *Grupos/Individual:* Uma vez as imagens no formato DICOM, uma análise pode ser iniciada a partir de um conjunto de pacientes agrupados por idade ou sexo, por exemplo, ou de forma individual. Trata-se da seleção das imagens que serão analisadas.

- *Segmentação:* Utilização de técnicas de segmentação de imagens digitais para isolar as micro-estruturas, baseando-se no algoritmo mostrado na Figura 3.3:

Seja f uma imagem, e R_1, R_2,... R_n um conjunto de regiões, onde cada uma possui um único pixel semente.
Repita:
  for (i=1...n)
    for (cada pixel p na borda de R_i)
      for (todos os vizinhos de p)
        Seja x, y a coordenada do vizinho
        Seja m_i a média do nível de cinza dos pixels de R_i
        Se vizinho não rotulado e $| f(x,y) - m_i | <= T$
        Adicione vizinho a R_i, atualize m_i
Até que mais nenhum pixel seja designado para regiões

Figura 3.3 - Algoritmo do módulo "Segmentação"

- *Reconhecimento de micro-estruturas:* Identificação da formação de micro-estruturas.

- *Classificação:* A classificação é realizada com base no número de pixels que compõem a micro-estrutura segmentada e identificada nos passos anteriores, representando, desta forma, o seu grau de desenvolvimento, baseando-se no algoritmo mostrado na Figura 3.4:

Leitura da imagem segmentada
Contagem do número de pixels que compõem cada segmento da imagem para classificá-los entre *n* classes, como por exemplo:
- Dente saudável (sem micro-estrutura)
- Dente carioso #1 (presença de micro-estruturas de 1 pixel de dimensão)
- Dente carioso #2 (micro-estruturas de 5 pixels de dimensão)
.
.
- Dente carioso #N (micro-estrutura de n pixels de dimensão)

Figura 3.4 - Algoritmo do módulo "Classificação"

- *Banco de resultados para diagnóstico e auxílio à tomada de decisão:* As imagens segmentadas são armazenadas no formato DICOM 3.0 para futuras consultas e auxilio ao diagnóstico.

O diagrama do sistema desenvolvido encontra-se no apêndice A.

3.1 Materiais e métodos

O método foi desenvolvido com o uso de técnicas de segmentação de imagens digitalizadas a partir de radiografias de raios X, dos prontuários dos pacientes que sofreram tratamento odontológico na Clínica Infantil do curso de Odontologia das Faculdades Unificadas da Fundação Educacional de Barretos.

Seguindo o Regimento nº 196 de 1996, foram realizadas, com base nas normas da CONEP (Comissão Nacional de Ética em Pesquisa), a aquisição das radiografias. Assim, após aprovação do projeto, foram cedidas 200 (duzentas) chapas radiográficas para a pesquisa.

As imagens são referentes a casos clínicos de 200 (duzentos) pacientes (crianças e adolescentes) e em cada imagem estão contidos os dois primeiros molares superiores e inferiores de um dos lados da arcada dentária do paciente, totalizando 400 (quatrocentos) dentes que foram estudados.

Optou-se trabalhar com os primeiros molares, superiores e inferiores, por se tratarem de dentes bastante grandes, possibilitando assim, uma maior área para análise. Outro motivo para se trabalhar com os primeiros molares se dá ao fato se serem um dos primeiros dentes a nascerem, o que resulta em um maior tempo exposição aos micro-organismos da cavidade bucal.

A Figura 3.5 ilustra exemplos de radiografias, que neste caso, foram digitalizadas com o uso de um *scanner* comum, enquanto a Figura 3.6 mostra as radiografias digitalizadas utilizando uma máquina fotográfica digital, tendo essas sido colocadas sobre um anteparo luminoso.

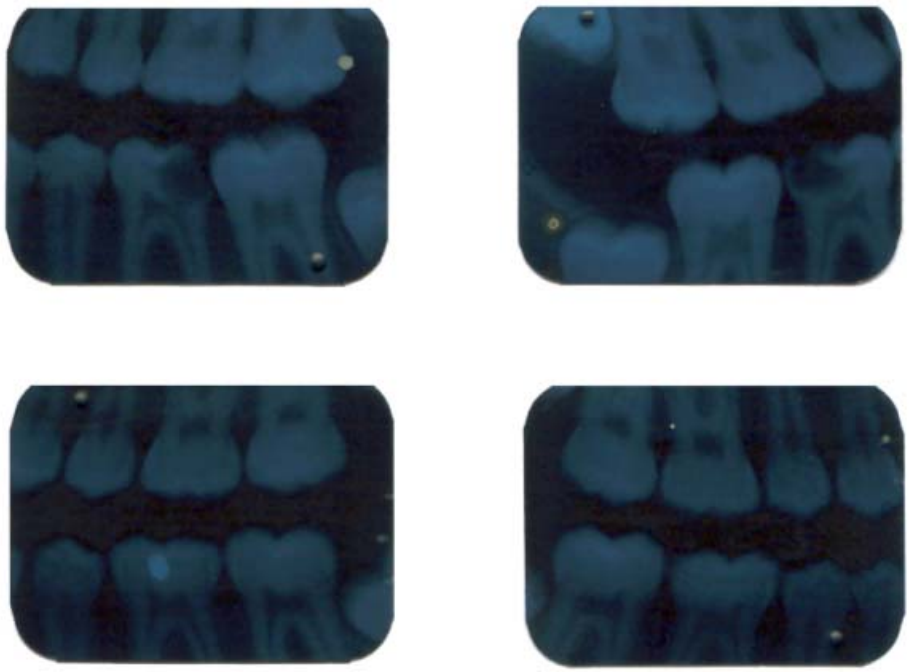


Figura 3.5 - Chapas radiográficas digitalizadas por meio de scanner comum

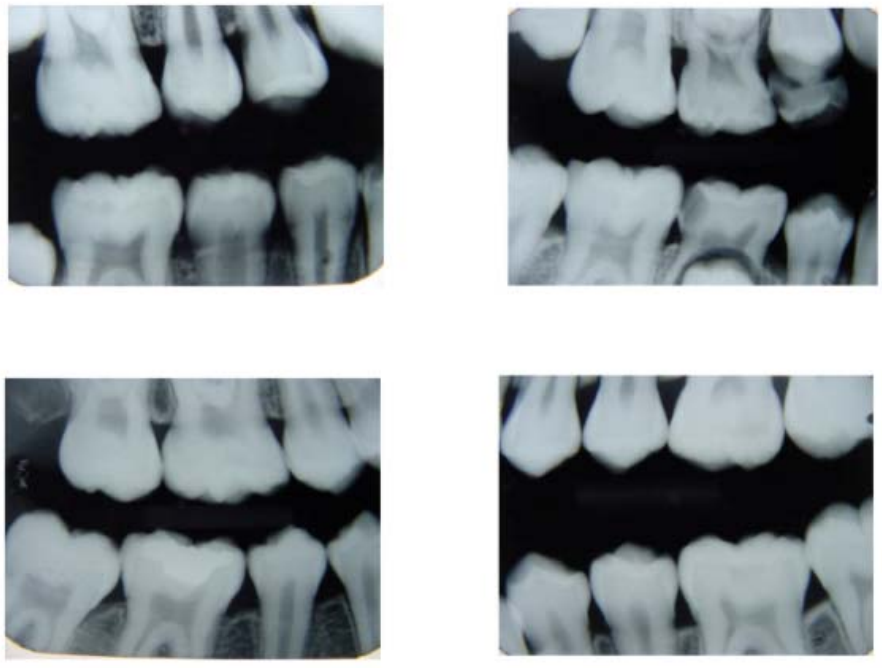


Figura 3.6 - Radiografias digitalizadas com uso de câmera fotográfica digital

3.2 Resultados

3.2.1 *Interface do sistema*

O aplicativo possui basicamente, como pode-se observar na Figura 3.7, três *menus* principais:

- Arquivo: possui as funções de entrada e saída, como Abrir, Salvar, Importar, Exportar e Imprimir.
- Ferramentas: possui as funções principais do aplicativo. Preenchimento/alteração das *tags* dos arquivos DICOM e Configuração de parâmetros para o processo de segmentação são algumas delas.
- Sobre: Informações de autoria e versão.

Figura 3.7 - Tela inicial do sistema com os três principais conjunto de funções: Arquivo, Ferramentas e Sobre.

É possível importar imagens dos formatos de armazenamento mais comuns como BMP, TIFF, JPEG e PNG e convertê-las pra o formato adotado pelo sistema: formato DICOM. Com a importação, o sistema dá a oportunidade de definir as *tags* que agregam informações referentes ao paciente e consulta. Sendo possível ainda, acrescentar qualquer outra informação relevante no campo "Anotações", como observa-se na Figura 3.8.

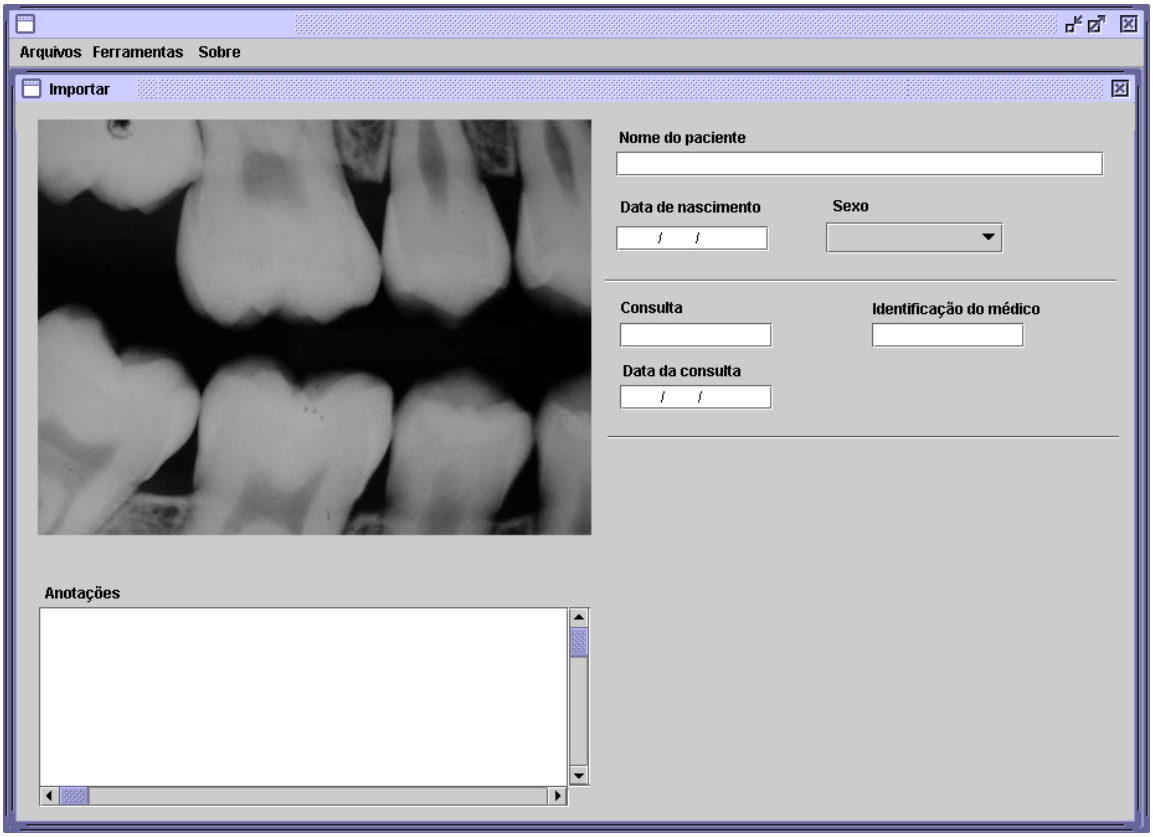


Figura 3.8 - Tela de importação de imagens dos formatos de armazenamento mais conhecidos para o formato DICOM, adotado pelo sistema.

A Figura 3.9 mostra a fase de seleção da área de interesse onde se deseja verificar a possível ocorrência de cáries, sendo possível, por motivos de manter o desempenho do sistema, definir no máximo três áreas simultaneamente.

Uma vez realizada a segmentação, é oferecida ao especialista a opção de selecionar quais regiões segmentadas devem ser consideradas pra a geração do vetor de características, como mostra a Figura 3.9.

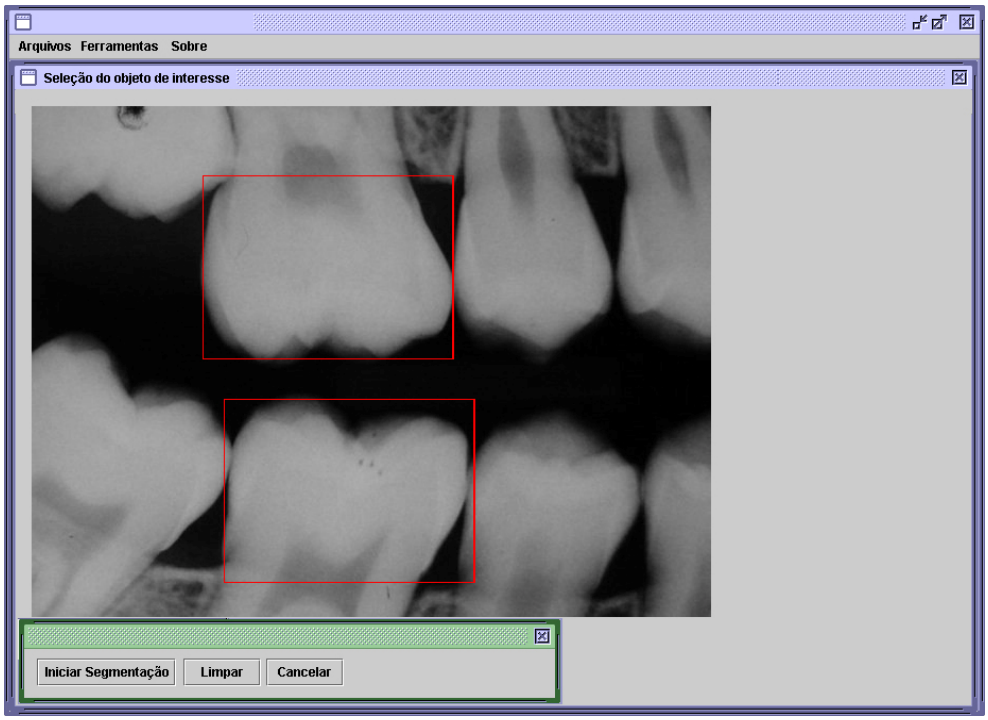


Figura 3.9 - Tela de seleção da área de interesse que serão submetidas ao processo de segmentação.

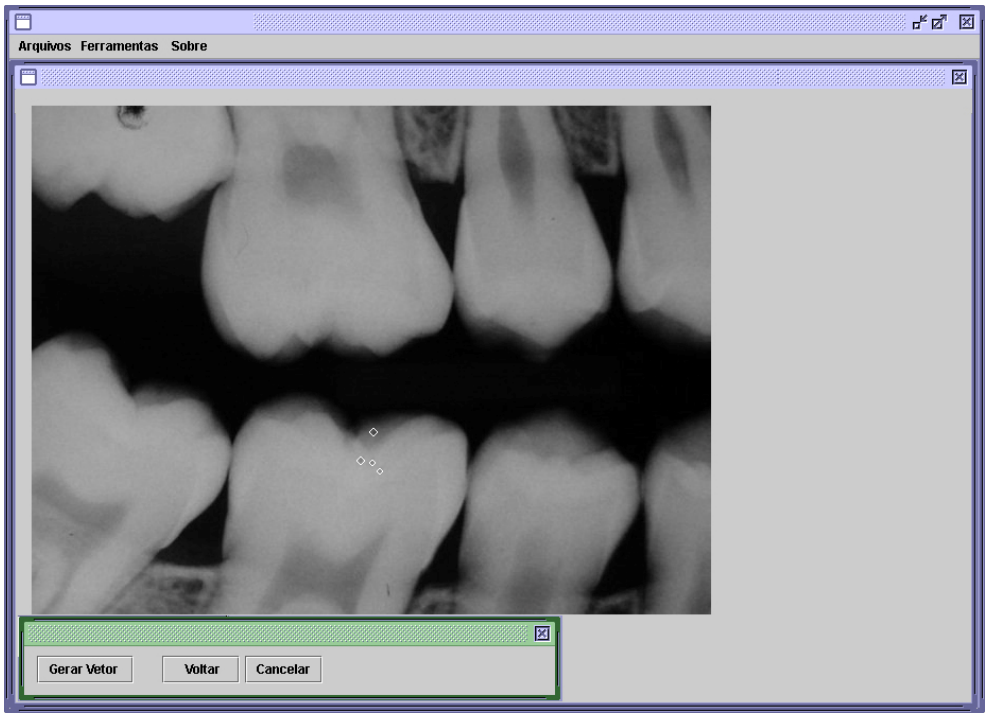


Figura 3.10 - Fase de seleção e exclusão dos segmentos que devem ser considerados para gerar o vetor de característica.

3.2.2 *Estudos de casos*

De um conjunto de 200 imagens que foram obtidas com base no protocolo da Comissão Nacional de Ética em Pesquisa (CONEP), serão apresentados, para fins documentais, cinco estudos de casos realizados com base no algoritmo ilustrado pela Figura 3.12.

As imagens segmentadas são apresentadas ao especialista do domínio para decisão quanto à submissão ao processo de classificação Bayesiana. Esta fase de intervenção do especialista se faz necessário para a observação de uma estrutura que não se trata de uma cárie, mas sim uma obturação, ou em casos de observação de uma estrutura em estágio bem avançado, onde a classificação se torna redundante.

Após a aprovação de quais estruturas devem ser consideradas na classificação, são extraídas informações do diâmetro da estrutura (em pixel) e armazenados em um vetor *X*.

Para determinar a dimensão da estrutura, foi realizada uma contagem horizontal dos pixels, obtendo, assim, a sua dimensão linear.

Este vetor é dado com entrada ao classificador, podendo classificar os pacientes, por predominância da ocorrência de estruturas com um dado tamanho de diâmetro, como um dos quatro quadros da doença sugeridos, os quais foram estabelecidos para os estudos de caso considerados, como mostra a Figura 3.11:

$$1 \text{ pixel} < \text{Microcáries} \leq 3 \text{ pixels}$$

$$4 \text{ pixel} < \text{Cáries pequenas} \leq 6 \text{ pixels}$$

$$7 \text{ pixels} \leq \text{Cáries médias} \leq 8 \text{ pixels}$$

$$\text{Cáries grandes} \geq 9 \text{ pixels}$$

Figura 3.11 - Estabelecimento das classes do grau de desenvolvimento das cáries dentárias

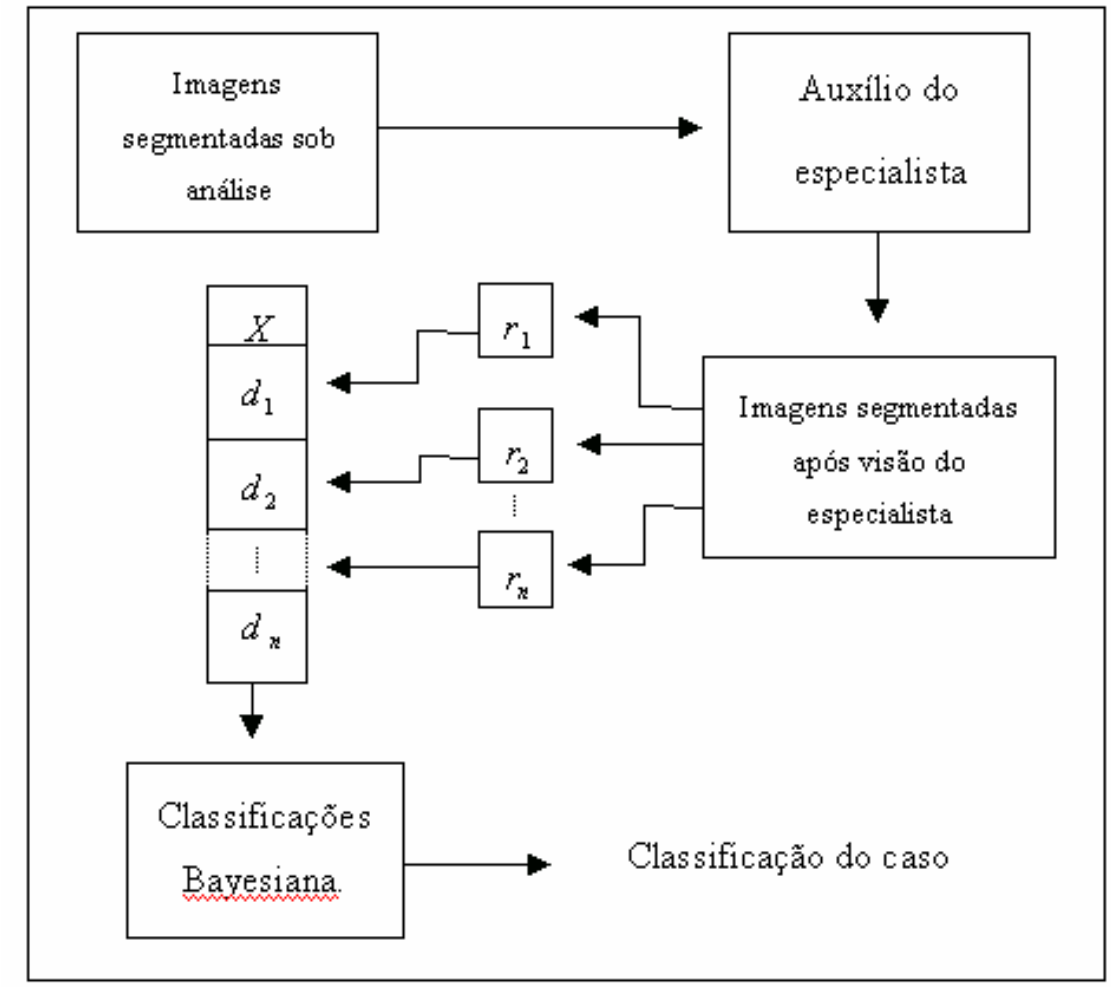


Figura 3.12 - Diagrama do algoritmo da classificação do estudo de caso, onde são sub-imagens da imagem segmentada com visão do especialista e , os valores em pixel do diâmetro das estruturas.

A seguir são apresentados o estudo de caso de cinco pacientes, sendo o primeiro caso (paciente #0) mostrado em detalhes para ilustrar os passos que são realizados para obtenção das estruturas detectadas

3.3 Estudo de caso para o paciente #0

A Figura 3.13 ilustra a imagem original. A Figura 3.17 a fase de seleção das áreas de interesse.

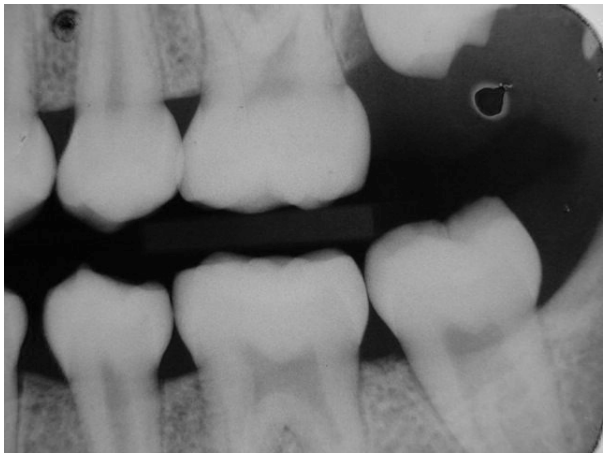


Figura 3.13 - Imagem original da radiografia do paciente #0

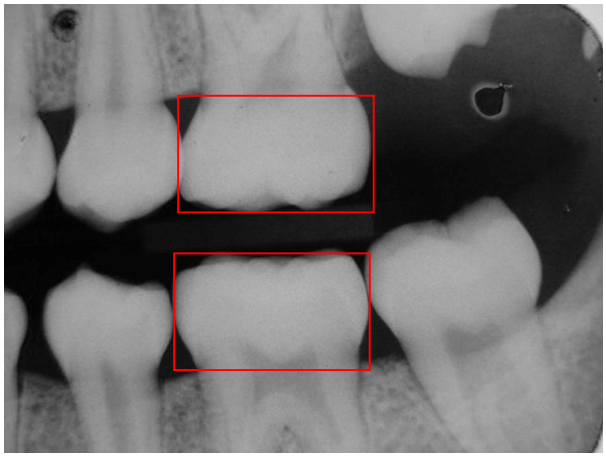


Figura 3.14- Seleção das áreas de interesse

A Figura 3.15 mostra o resultado obtido na etapa de segmentação com as regiões destacadas utilizando falsa cor.

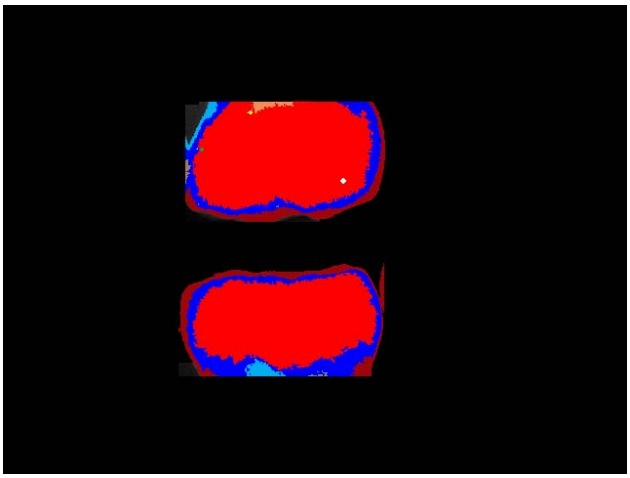


Figura 3.15 - Imagem segmentada das áreas selecionadas

A mostra o resultado da etapa de detecção de bordas por meio do uso da máscaras do tipo passa-alta mostrada na Figura 1.2.

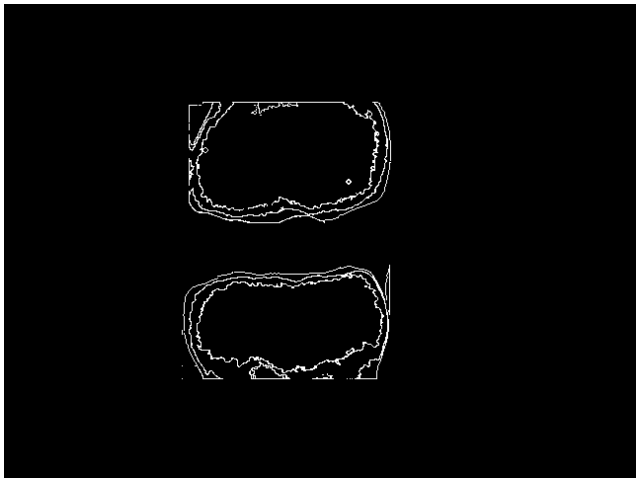


Figura 3.16 - Resultado do uso da máscara da Figura 1.2 na Figura 3.15

A Figura 3.17 mostra o resultado obtido pela etapa de reconhecimento de padrões circulares com uso da Transformada de Hough.

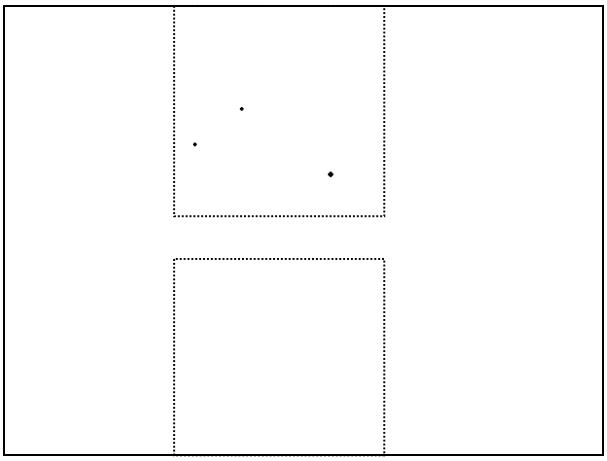


Figura 3.17 - Imagem segmentada do paciente #0 com seleção de área de interesse e visão especialista

A Figura 3.15 ilustra a seqüência de análise para o paciente #0.

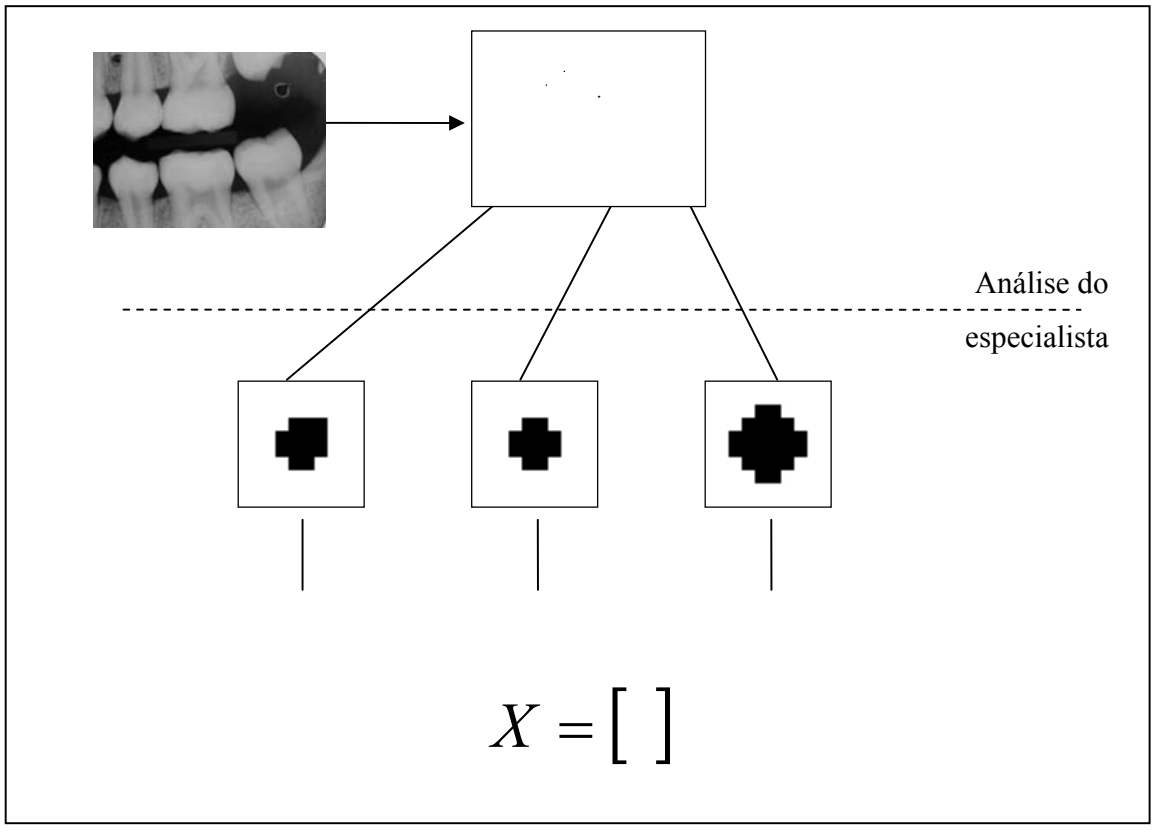


Figura 3.18 - Estudo de análise para o paciente #0, onde as sub-imagens são obtidas para fins de classificação.

Após as segmentações, foram obtidos os parâmetros para a composição do vetor *X*, ou vetor de classificação na forma:

$$X = \begin{bmatrix} 4 \\ 4 \\ 6 \end{bmatrix}$$

Os próximos casos foram simplificados mostrando somente as fases mostradas pelo sistema. A etapa de segmentação não é mostrada, deixando a detecção das estruturas transparente ao usuário.

3.4 Estudo de caso para o paciente #44

A Figura 3.19 ilustra a imagem original. A Figura 3.20 ilustra a imagem segmentada com inclusão da visão especialista para a geração das sub-imagens.

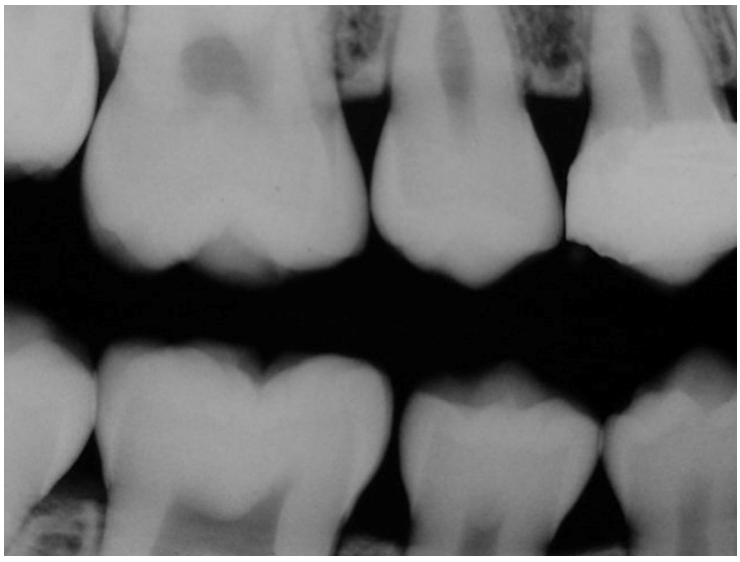


Figura 3.19 - Imagem original da radiografia do paciente #44

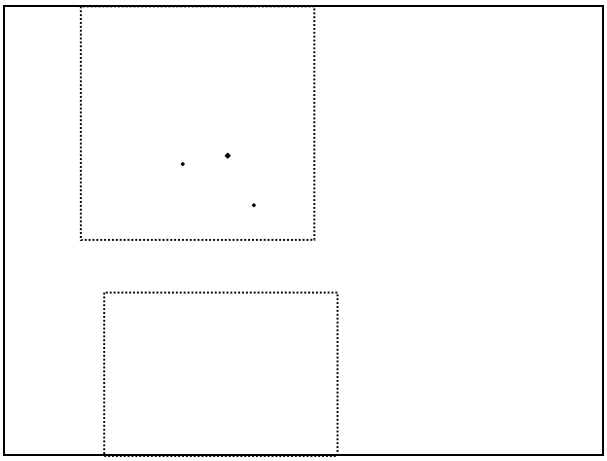


Figura 3.20 - Imagem segmentada do paciente #44 com seleção de área de interesse e visão especialista

A Figura 3.21 ilustra a seqüência de análise para o paciente #44.

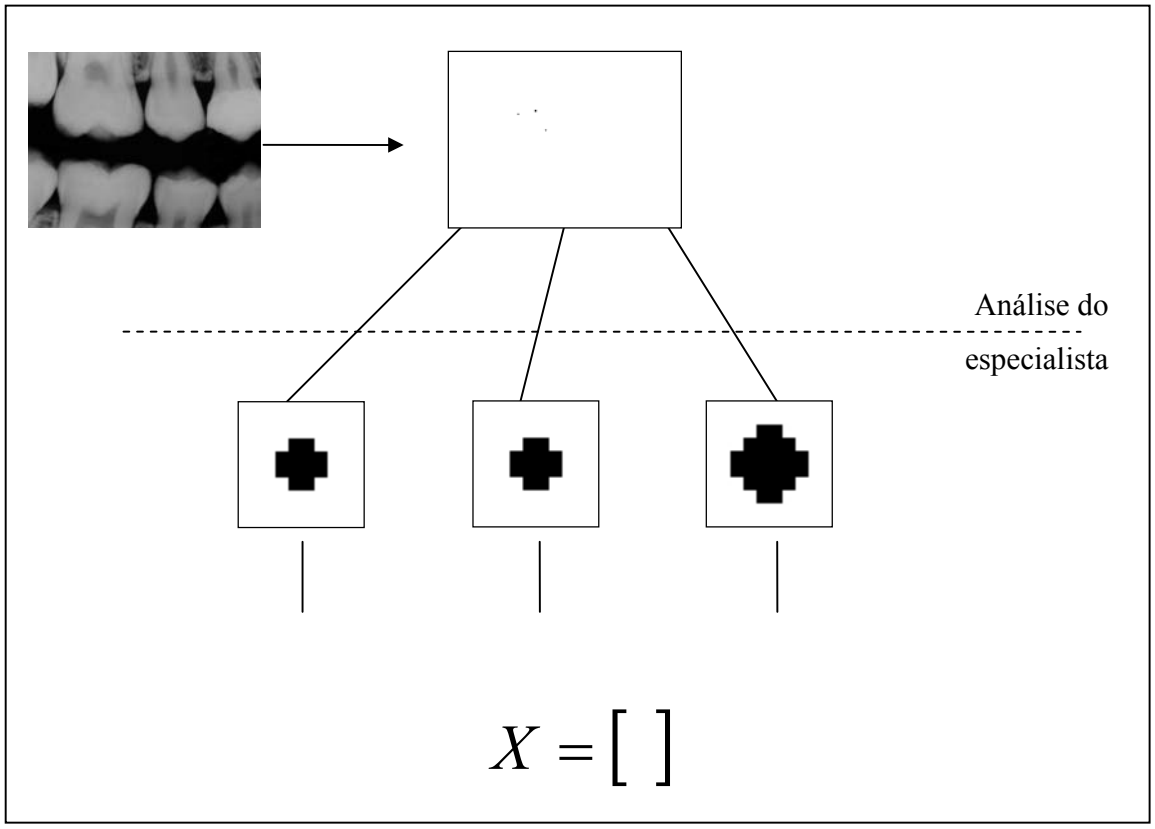


Figura 3.21 - Estudo de análise para o paciente #44, onde as sub-imagens são obtidas para fins de classificação

Após as segmentações, foram obtidos os parâmetros para a composição do vetor *X*, ou vetor de classificação na forma:

$$X = \begin{bmatrix} 4 \\ 4 \\ 6 \end{bmatrix}$$

3.5 Estudo de caso para o paciente #89

A Figura 3.22 ilustra a imagem original. A Figura 3.23 ilustra a imagem segmentada com inclusão da visão especialista para a geração das sub-imagens.

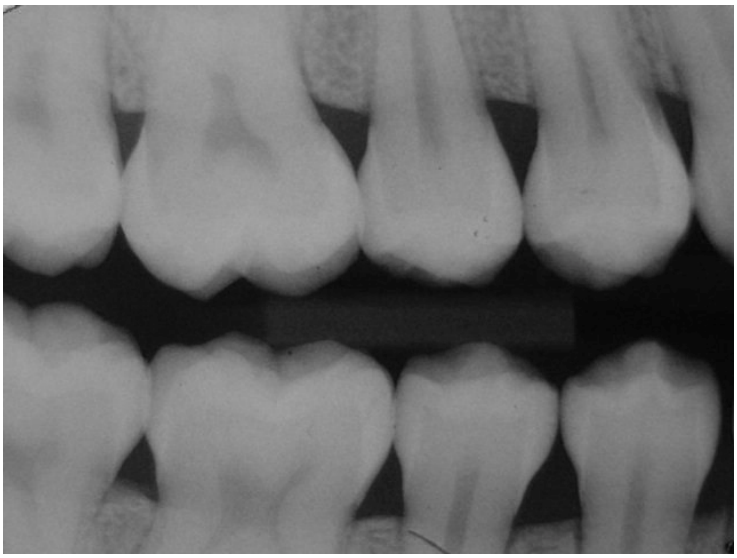


Figura 3.22 - Imagem original da radiografia do paciente #89

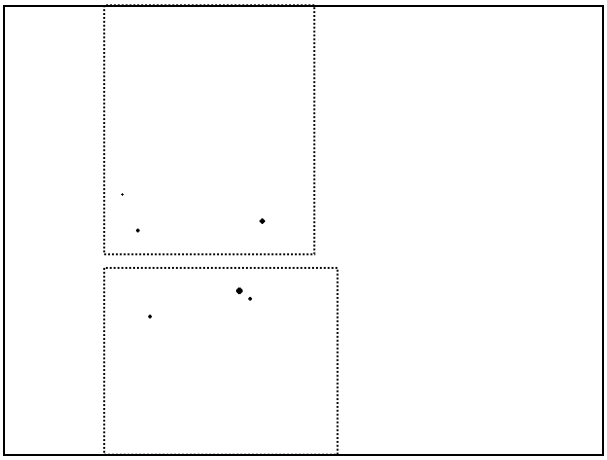


Figura 3.23 - Imagem segmentada do paciente #89 com seleção de área de interesse e visão especialista

A Figura 3.24 ilustra a seqüência de análise para o paciente #89.

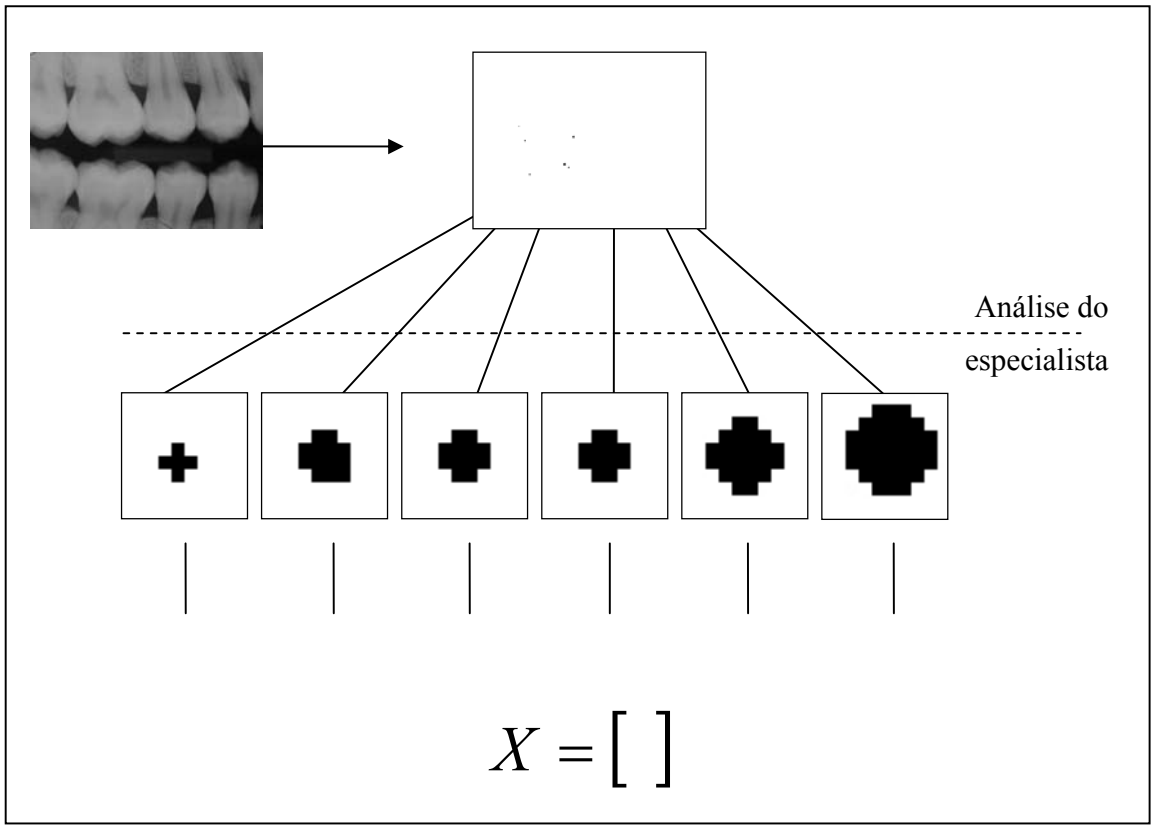


Figura 3.24 - Estudo de análise para o paciente #89, onde as sub-imagens são obtidas para fins de classificação

Após as segmentações, foram obtidos os parâmetros para a composição do vetor *X*, ou vetor de classificação na forma:

$$X = \begin{bmatrix} 3 \\ 4 \\ 4 \\ 4 \\ 6 \\ 7 \end{bmatrix}$$

3.6 Estudo de caso para o paciente #101

A Figura 3.25 ilustra a imagem original. A Figura 3.26 ilustra a imagem segmentada com inclusão da visão especialista para a geração das sub-imagens.

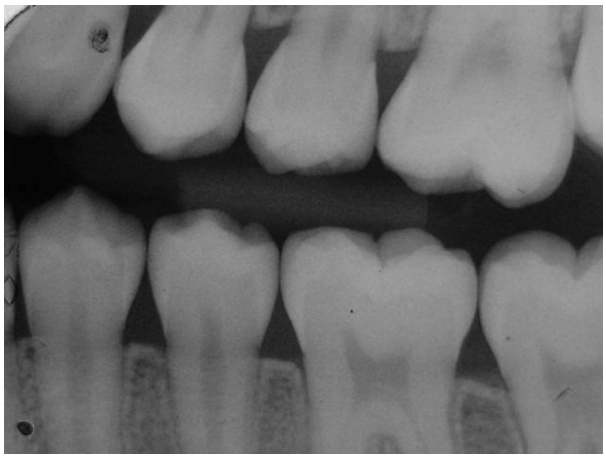


Figura 3.25 - Imagem original da radiografia do paciente #101

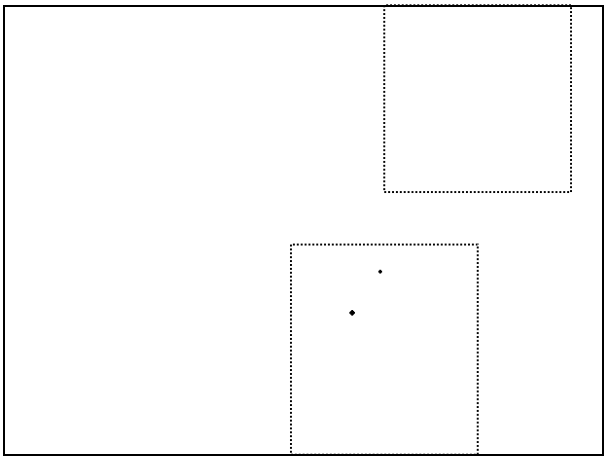


Figura 3.26 - Imagem segmentada do paciente #101 com seleção de área de interesse e visão especialista

A Figura 3.27 ilustra a seqüência de análise para o paciente #101.

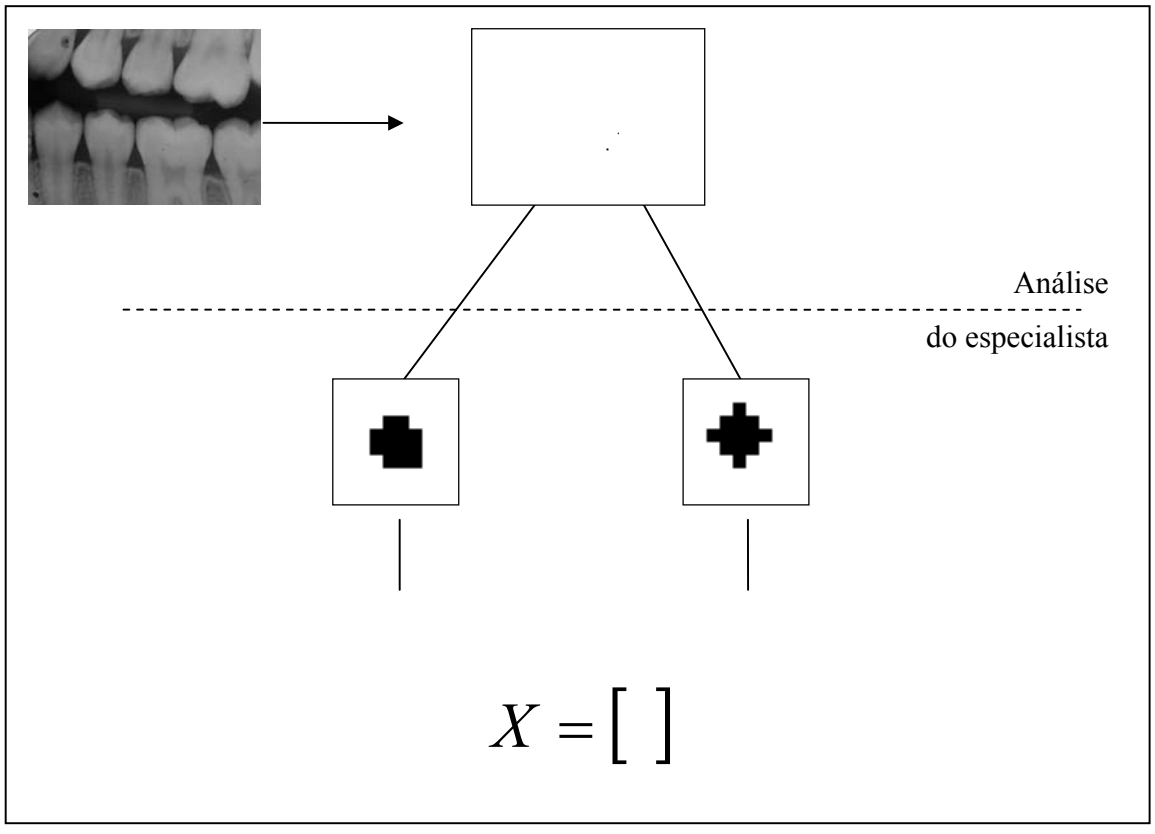


Figura 3.27 - Estudo de análise para o paciente #101, onde as sub-imagens são obtidas para fins de classificação

Após as segmentações, foram obtidos os parâmetros para a composição do vetor *X*, ou vetor de classificação na forma:

$$X = \begin{bmatrix} 4 \\ 5 \end{bmatrix}$$

3.7 Estudo de caso para o paciente #112

A Figura 3.28 ilustra a imagem original. A Figura 3.29 ilustra a imagem segmentada com inclusão da visão especialista para a geração das sub-imagens.

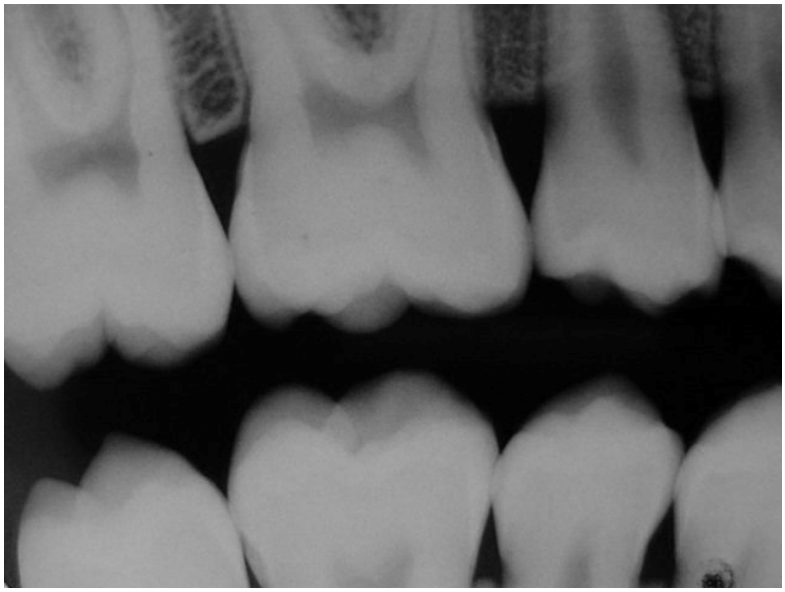


Figura 3.28 - Imagem original da radiografia do paciente #112

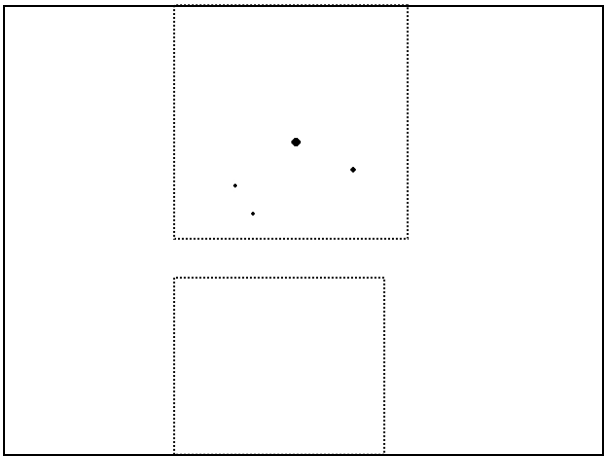


Figura 3.29 - Imagem segmentada do paciente #112 com seleção de área de interesse e visão do especialista

A Figura 3.30 ilustra a seqüência de análise para o paciente #112.

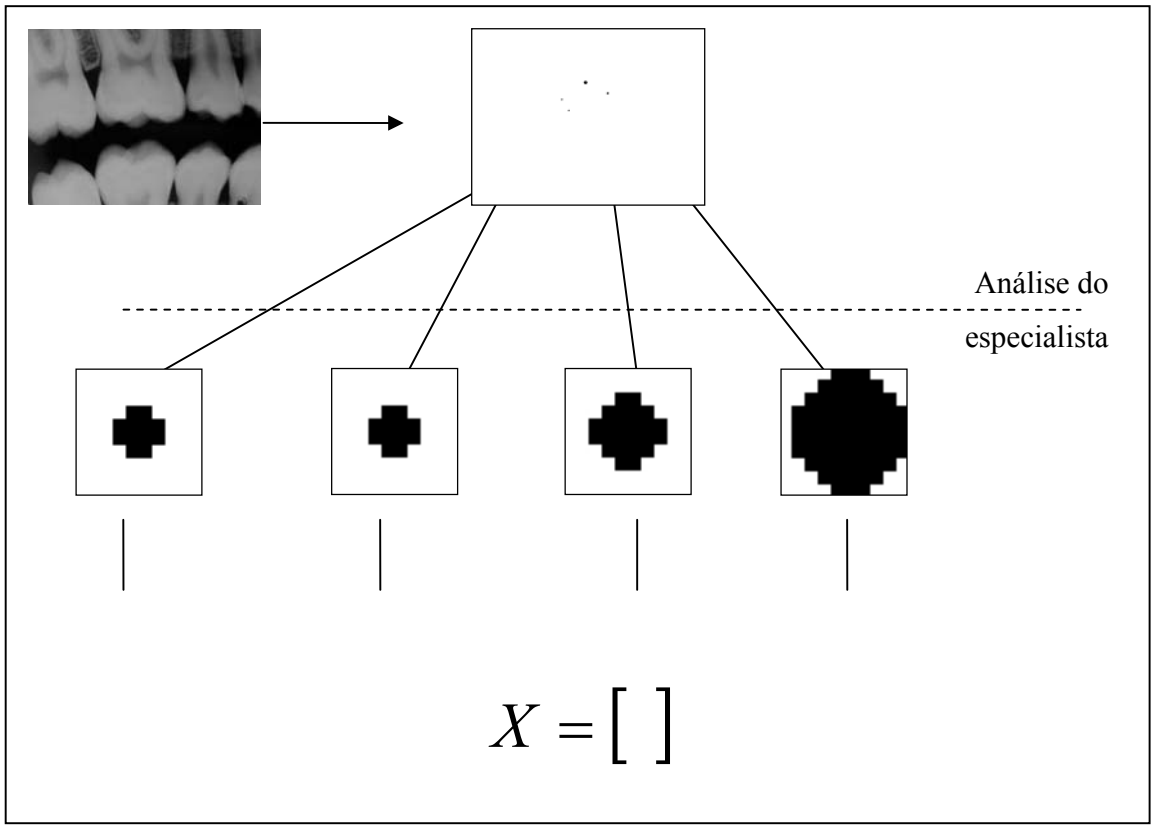


Figura 3.30 - Estudo de análise para o paciente #112, onde as sub-imagens são obtidas para fins de classificação

Após as segmentações, foram obtidos os parâmetros para a composição do vetor *X*, ou vetor de classificação na forma:

$$X = \begin{bmatrix} 4 \\ 4 \\ 6 \\ 9 \end{bmatrix}$$

3.8 Classificação com base no algoritmo Bayesiano

A partir dos estudos de caso realizados, uma seqüência classificatória foi estabelecida, como por exemplo, classes de pacientes em que se evidenciaram cáries nas quatro modalidades apresentadas.

Para a implementação desta etapa foi utilizada a plataforma MATLAB versão 7.0 com base no uso de um classificador Bayesiano e distância de Mahalanobis.

A Tabela 1 ilustra o resultado obtido para os estudos dos casos considerados:

Tabela 1- Classificação da presença de cáries por modalidade

Paciente	$1 \leq d \leq 3$	$4 \leq d \leq 6$	$7 \leq d \leq 8$	$d \geq 9$
#0	0	3	0	0
#44	0	3	0	0
#89	1	4	1	0
#101	0	0	2	0
#112	0	3	0	1

Para o conjunto de pacientes apresentados, observou-se a predominância de cáries pequenas.

No caso do paciente #89, onde foi possível a detecção de uma micro-cárie, o especialista tem como iniciar métodos preventivos para inibir a evolução da mesma.

Para casos em que a cárie já está bastante desenvolvida, como é possível observar pela presença de cáries médias e cáries grandes, o especialista tem, com base nas informações apresentadas, a aplicação de métodos curativos que impeçam o agravamento e/ou a perda do dente.

No capitulo seguinte são apresentados as conclusões obtidas a partir dos resultados analisados e sugestões de trabalhos futuros.

4 CONCLUSÕES E PROPOSTA DE TRABALHOS FUTUROS

4.1 Conclusões

O método apresentado, conforme resultados obtidos, encontra aplicação na identificação da formação de micro-estruturas dentárias, uma vez que viabilizou a caracterização de cáries em formação, com a dimensão da ordem de 1 a 3 pixels.

Outro aspecto relevante na metodologia desenvolvida, conforme resultados obtidos, foi o uso do classificador Bayesiano, que caracterizou o método para auxílio ao diagnóstico odontológico com precisão e segurança quanto à repetibilidade.

O uso de processamento digital de imagens se mostrou útil e eficaz no auxílio ao diagnóstico precoce da formação de cáries dentárias, o que é relevante não só para os indivíduos adultos, mas principalmente para crianças, dada a importância desta questão na área da saúde infantil, permitindo por meio de um diagnóstico mais apurado, aplicações de métodos preventivos e curativos mais eficazes, resultando na melhoria da qualidade da saúde bucal dos pacientes.

4.2 Proposta para trabalhos futuros

- Modularizar o sistema e portabilizá-lo para aplicação direta em clínicas odontológicas;
- Adequar o sistema para uso com múltiplos usuários, em ambiente colaborativo, via rede Web;
- Implementar o padrão DICOM em sua configuração completa para utilização da metodologia em ambientes compartilhados;
- Realizar os testes de conformidade do padrão DICOM implementado.

REFERÊNCIAS BIBLIOGRÁFICAS

ARAÚJO, F. B., FIGUEIREDO, M. C. *Promoção de Saúde em Odontopediatria*. In: Aboprev: Promoção de Saúde Bucal. 2 ed., São Paulo: Artes Médicas, 1999. p. 291.

BEKMAN, º R., NETO, P. L. O. C. *Análise Estatística da Decisão,* Editora Edgard Blücher Ltda., 1980.

DUDA, R., HART , P.E., (1973) Pattern Classification and Scene Analysis - John Wiley & Sons, Inc., NY.

GARG, A., ROTH, D. Understanding Probabilistic Classifiers, University of Illinois, USA, 2001.

GONZALEZ, R. C., WOODS, R. E. Digital Image Processing, Prentice-Hall, Inc., 2ª edição, 2001.

LEE, T., RICHARDS, J. A., SWAIN, P. H. Probabilistic and evidential approaches for multisource data analiysis, IEEE Transactions on Geoscience and Remote Sensing, vol.1 GE-25, n.3, 283-293, 1987.

LOW, A. *Introductory computer vision and image processing*. [S.l.]: McGraw-hill book company,1991.

MASCARENHAS, N. D., CRUVINEL, P. E., HOMEM, M. R. P. *The linear attenuation coeficients as features of multiple energy CT image classification.* Nuclear Instruments and Methods in Physics Research. Amsterdam:, v.452, n.A, p.35 – 360, 2000.

MÁXIMO, O. A., FERNANDES, D. Uso de graus de confiança das classes em Classificadores Bayesianos, XI Simpósio Brasileiro de Sensoriamento Remoto, Belo Horizonte, 2003.

MOREL, J. M. & SOLIMINI, S. *Variational Methods in Image Segmentation*: vol. 14, Birkhäuser, 1995.

MUMFORD, D. & SHAH, J. *Boundary detection by minimizing functionals*. IEEE Conference on Computer Vision and Pattern Recognition, San Francisco, 1985.

NEMA - National Electrical Manufecturers Association, Standard Documentation, http://medical.nema.org/dicom/2003.html (visitado em agosto de 2004).

NEWBRUN, E. *Cariologia*. 1 ed., São Paulo: Livraria e editora Santos Ltda., 1988. 326p.

PEREIRA, A. S. Processamento de imagens médicas utilizando a Transformada de Hough. Tese (Doutorado) — Universidade de São Paulo – Instituto de Física de São Carlos, 1995.

PINTO, G. P., *Epidemiologia das Doenças Bucais no Brasil*. In: Aboprev: Promoção de Saúde Bucal. 2 ed., São Paulo: Artes Médicas, 1999. p. 29-41.

ROSENFELD, A. & THURSTON, M. *Edge and curve detection for visual scene analysis.* IEEE Trans. Syst., Man, Cyber., SMC-6, 420-430, 1976.

SCHARF, L. L. Statistical signal processing – detection, estimation, and time series analysis, Addison-Wesley, 1991.

WEYNE, S. C. *A Construção do Paradigma de Promoção de Saúde – Um Desafio para as Novas Gerações*. In: Aboprev: Promoção de Saúde Bucal. 2 ed., São Paulo: Arte Médicas, 1999. p. 3-23.

APÊNDICE A

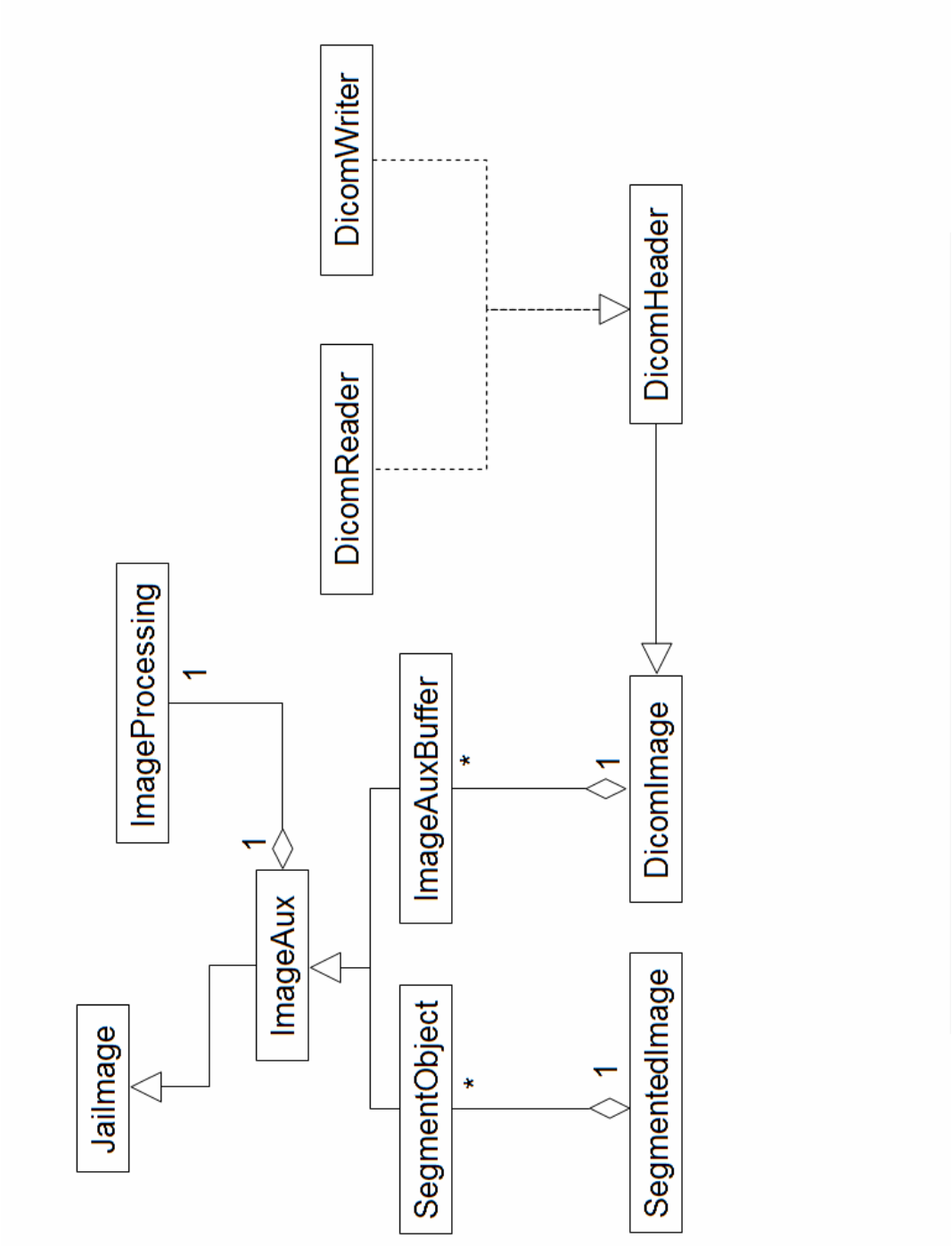


Diagrama de classes UML do sistema implementado

APÊNDICE B

```
/*=====================================================================*\
|*********************************************************************|
|  "Método baseado em processamento digital de imagens para diagnóstico |
|                  precoce de micro-estruturas dentárias"              |
| (Classe para leitura de cabeçalho DICOM  de imagens sem compressão:)  |
|              Mestrando: Yuji de Oliveira Ohnishi                      |
|              Orientador: Prof. Dr. Paulo Estevão Cruvinel             |
|              Co-Orientador: Prof. Dr. Walter Antônio de Almeida       |
|*********************************************************************|
\*=====================================================================*/


package DicomDecoder ;
import java.awt.* ;
import java.util.*;
import java.io.* ;
import tools.* ;


public  class DicomHeaderReader{

       byte[] data ;
       int index  ;
       private int dataLength;
       private static  final boolean debug = false ;

      public static final int MAX_HEADER_SIZE = 10000 ;
      private int maxHeaderSize  ;
      public    static    final    String       ImplicitVRLittleEndian    =
"1.2.840.10008.1.2"     ;
      public static final int _ImplicitVRLittleEndian = 0 ;
```

```
      public    static    final    String       ExplicitVRLittleEndian    =
"1.2.840.10008.1.2.1"   ;
      public static final int _ExplicitVRLittleEndian = 1 ;
      public    static    final    String        ExplicitVRBigEndian    =
"1.2.840.10008.1.2.2"   ;
      public static final int _ExplicitVRBigEndian = 2 ;
      public static final int _ImplicitVRBigEndian = 3 ;
      public     static     final     String          JPEGCompression     =
"1.2.840.10008.1.2.4." ;
      public static final int _JPEGCompression = 10 ;
      public static final int _notUnderstood = 1000 ;
      private  boolean oneSamplePerPixel = true ;
      private  boolean oneFramePerFile   = true ;
      private int errorDetector = -1 ;


      private String  VRString    ;
      private String  xSyntaxUID         = "unknown" ;//   {
      0x0002,0x0010     }
      private String  imageType          = "unknown" ;//{  0x0008,0x0008
      }
      private String  studyDate          = "unknown" ;//{  0x0008,0x0030
      }



      private String  modality           = "unknown" ;//{  0x0008,0x0060
      }
      private String  manufacturer  = "unknown" ;//{  0x0008,0x0070);   }
      private String    institution       =         "unknown"         ;//{
      0x0008,0x0080);

}
      private String    physician         =         "unknown"         ;//{
      0x0008,0x0090);

}
      private String  patientName        =         "unknown"         ;//{
      0x0010,0x0010);

}
      private String  patientID          = "unknown" ;//{  0x0010,
0x0020);

}
```

```
      private     String          patientBirthdate=     "unknown"     ;//{
      0x0010,0x0030);   }
      private String  sex                       = "unknown" ;//{

0x0010,0x0040);   }


      private int numberOfFrames         =  1;       //0x0028,0x0008    :
//IS :

integer string
      private int samplesPerPixel   =  1;             //0x0028, 0x0002
      private int h                              = -1;             //
0x0028,

0x0010)
      private int w                              = -1;       //      0x0028,
0x0011
      private int bitsAllocated          = -1;       // 0x0028, 0x0100
      private int bitsStored             = -1;       //0x0028, 0x0101
      private int highBit                = -1;             //0x0028,
0x0102);
      private int signed                        = -1;       //0x0028,

0x0103(pixelRepresentation )
      private int size                   = -1;       //0x7Fe0, 0x0010
      private int n                              = -1;       // = 1 ou 2
      private int VR                             = -1; //  representação  do
valor


      private boolean bE = false ;//bigEndian



/**
*        Passar  como  como  argumento  um  array  de  byte   ,  esse  array  é  a
imagem DICOM a ser

*       lida.
*/



      public DicomHeaderReader ( byte [] HeaderData ){
            this.data = HeaderData ;
```

```
            dataLength =  data.length ;
            index =0;
            initHeaderSize() ;
            getVR();
            getEssentialData() ;

      }

      protected void initHeaderSize(){
            maxHeaderSize = MAX_HEADER_SIZE * 1 ;
            if(maxHeaderSize > dataLength) maxHeaderSize = dataLength/2 ;

      }

      protected void getVR(){
            xSyntaxUID       = getaString (0x0002,0x0010, index);
            debug( "\n-------------Ê\nTransfertsyntaxUID              "
+

xSyntaxUID ) ;

            if(xSyntaxUID != "Unknown" ){
                  debug("  xSyntaxUID   Unknown dans boucle if" ) ;
                  xSyntaxUID = xSyntaxUID.trim() ;
                  if(xSyntaxUID.equals(ImplicitVRLittleEndian ) )
VR =

_ImplicitVRLittleEndian ;
                  else if (xSyntaxUID.equals(ExplicitVRLittleEndian ) )VR
=

_ExplicitVRLittleEndian ;
                  else if (xSyntaxUID.equals(ExplicitVRBigEndian ) )
VR =

_ExplicitVRBigEndian ;
                  else if (xSyntaxUID.startsWith(JPEGCompression ))    VR
=

_JPEGCompression ;
                  else VR = _notUnderstood ;
```

```
                switch (VR){
                      case _ImplicitVRLittleEndian  : VRString =

"ImplicitVRLittleEndian "     ; break;
                      case _ExplicitVRLittleEndian  : VRString =

"ExplicitVRLittleEndian "     ; break;
                      case _ExplicitVRBigEndian          : VRString =

"ExplicitVRBigEndian    "     ; break;
                      case _JPEGCompression              : VRString =

"JPEGCompression  "           ; break;
                      case _notUnderstood                :  VRString  =
"not

understood  "                 ; break;
                      default : VRString =" Something curious happened
!" ;
                }// end of switch
                debug( " Value representation  :               "       +
VRString ) ;
           }//end if !Unknown

           if( VR ==_notUnderstood){

                if ( (data[0] != 'D') || (data[1] != 'I' ) || (data[2]
!= 'C' )||

(data[3] != 'M' )){
                      if(data[0] < data[1] ){
                            VR= _ImplicitVRBigEndian ;
                            debug( "VR is implicitVRBigIndian ") ;
                            bE = true ;
                      }//endif(dat..
                }

           }// end if VR==_notUnderstood
```

```
	}//fim getVR


	protected void getEssentialData(){

	imageType         = getaString (0x0008,0x0008, index);
	studyDate         = getaString (0x0008,0x0020 ,index);
	modality          = getaString (0x0008,0x0060 ,index);
	manufacturer      = getaString (0x0008,0x0070  ,index);
	institution       = getaString (0x0008,0x0080  ,index);
	physician         = getaString (0x0008,0x0090  ,index);
	patientName       = getaString (0x0010,0x0010  ,index);
	patientID         = getaString (0x0010,0x0020 ,index);
	patientBirthdate= getaString (0x0010,0x0030    ,index);
	sex                     = getaString (0x0010,0x0040   ,index);

	h = getAnInt(0x0028, 0x0010, index ) ;
		if(h ==-1 ){
			index = 0 ;
			h =getAnInt(0x0028, 0x0010) ;
			debug( "Something is going wrong in this file  !!!, h
== -1" );
		}
	w = getAnInt(0x0028, 0x0011, index ) ;

		if(w ==-1 ){
			index = 0 ;
			w = getAnInt(0x0028, 0x0011) ;
			 debug( "Nouvelle valeur de  w avec  0x0028, 0x0011  :
" + w);
			debug( "Something is gooing wrong in this file  !!!, w
== -1" );
		}

	bitsAllocated          = getAnInt(0x0028, 0x0100, index ) ;
		debug("bitsAllocated ......."+ bitsAllocated );
		if(bitsAllocated ==-1 ){
			index = 0 ;
```

```
                  bitsAllocated = getAnInt(0x0028, 0x0100) ;
                   debug(  "Nouvelle  valeur  de    bitsAllocated:  "  +
bitsAllocated);
            }
      bitsStored             = getAnInt(0x0028, 0x0101, index ) ;
            debug("bitsStored ......."+ bitsStored );
            if(bitsStored ==-1 ){
                  index = 0 ;
                  bitsStored = getAnInt(0x0028, 0x0101) ;
                   debug(  "Nouvelle  valeur  de    bitsStored  :  "  +
bitsStored);
            }
      highBit                = getAnInt(0x0028, 0x0102, index ) ;
      debug("highBit ......."+ highBit );
            if(highBit ==-1 ){
                  index = 0 ;
                  highBit = getAnInt(0x0028, 0x0102) ;
                   debug( "Nouvelle valeur de  highBit  : " + highBit);
            }
      signed                       = getAnInt(0x0028, 0x0103, index )

;//(pixelRepresentation )
      debug("signed ......."+ signed );
            if(signed ==-1 ){
                  index = 0 ;
                  signed = getAnInt(0x0028, 0x0103) ;
                   debug( "Nouvelle valeur de  signed  : " + signed);
            }



      debug( " ImageType :::                  " + imageType ) ;
      debug( " studyDate :::                  " + studyDate ) ;
      debug( " modality :::                   " + modality ) ;
      debug( " manufacturer :::               " + manufacturer ) ;
      debug( " Institution :::                " + institution ) ;
      debug( " physician  :::                 " + physician ) ;
      debug( " patientName :::                " + patientName ) ;
```

```
      debug( " patientID :::                    " + patientID ) ;
      debug( " patientBirthdate :::             " + patientBirthdate ) ;
      debug( " sex                   :::        " + sex ) ;

      debug( " samplesPerPixel :::        " + samplesPerPixel ) ;
      debug( " h :::                            " + h ) ;
      debug( " w :::                       " + w ) ;
      debug( " bitsAllocated :::                " + bitsAllocated ) ;
      debug( " bitsStored :::                   " + bitsStored ) ;
      debug( " highBit :::                      " + highBit ) ;
      debug( " signed :::                       " + signed ) ;



      //size
      int pos                       =    lookForMessagePosition(    0x7Fe0,
0x0010, index);
      int tSize =  samplesPerPixel * w*h*bitsAllocated/8 ;// theoritical
Size
      if( pos != -1 )  size  =  readMessageLength(pos+4) ;
      debug( " size1 :::                        " + size ) ;
      int HeaderSize =  pos ;
      debug( "\nHeaderSize    : " + HeaderSize ) ;
      int figuredSize = HeaderSize + 8 + tSize ;
      errorDetector = dataLength - figuredSize ;
      debug( "Byte difference , error detector: dataLength - figuredSize
: "+

errorDetector ) ;

      if(errorDetector == 4 ){
            size = readMessageLength(pos+8) ; // try explicit VRStringize
            errorDetector = dataLength - size ;
      }

      if( errorDetector != 0){


            samplesPerPixel   = getAnInt(0x0028, 0x0002) ;// not mandatory
?
```

```
                        if(samplesPerPixel <0 || samplesPerPixel> 3 ){
                               samplesPerPixel = 1 ;// default value
                              debug( " NEW samplesPerPixel :::    " +
samplesPerPixel ) ;
                        }
                        else if (samplesPerPixel == 1) oneSamplePerPixel =
true ;
                        else oneSamplePerPixel = false ;



                  try{numberOfFrames                                    =
Integer.parseInt(getaString(0x0028,0x0008)) ; }
                  catch(NumberFormatException nFE){ numberOfFrames = 1 ;}
                  if (numberOfFrames > 1 ) oneFramePerFile = false ;

            tSize = numberOfFrames * tSize * samplesPerPixel;
            figuredSize = HeaderSize + 8 + tSize ;
            errorDetector = dataLength - figuredSize ;
            if (VR == _JPEGCompression)
                  debug( "_JPEGCompression , can't read that file" ) ;

            debug( "2 Byte difference , error detector: "  +
errorDetector
            +"\n Frame per file  " +      numberOfFrames
            +"\n samplesPerPixel "  +           samplesPerPixel) ;
            }



/*          size =readMessageLength(pos+6) ; // try explicit VRStringize
            debug("new size ::: pos+6     " + size ) ;
            size =readMessageLength(pos+8) ; // try explicit VRStringize
            debug("new size ::: pos+8     " + size ) ;
*/


      }// ENDOF   getInfo()
```

```
      private      int   lookForMessagePosition(int   groupElement,   int
dataElement ){

            int LenMax = data.length -3;
            boolean found = false ;
            byte testByte1 = (byte) (groupElement & 0xff);
            byte testByte2 = (byte) ((groupElement & 0xff00)>>8);
            byte testByte3 = (byte) (dataElement & 0xff);
            byte testByte4 = (byte) ((dataElement & 0xff00)>> 8);

            //debug(" recherche :"+

Tools.toHex(testByte1)+Tools.toHex(testByte2)+Tools.toHex(testByte3)+

Tools.toHex(testByte4));
            if(bE){
                  debug( "Strange file going on here ");
                  for ( int j =0 ;  j< LenMax || found ; j++){
                        if(  (  data[j]  ==  testByte2  )  &&  (data[j+1]  ==
testByte1) &&

(data[j+2]== testByte4) && (data[j+3]== testByte3)){
                              found = true ;
                              return j ;
                        }//endif

                  }//end if(bE)
            }//endifbE
            else{
                  for ( int j =0 ;  j< LenMax || found ; j++){
                        if(  (  data[j]  ==  testByte1  )  &&  (data[j+1]  ==
testByte2) &&

(data[j+2]== testByte3) && (data[j+3]== testByte4)){
                              found = true ;
                                    return j ;
                        }//endif
                  }
            }//end else
```

```
            return -1 ;
      }//endOfmethod

      private      int   lookForMessagePosition(int   groupElement,   int
dataElement, int  j ){

            int LenMax = data.length -3;
            boolean found = false ;
            byte testByte1 = (byte) (groupElement & 0xff);
            byte testByte2 = (byte) ((groupElement & 0xff00)>>8);
            byte testByte3 = (byte) (dataElement & 0xff);
            byte testByte4 = (byte) ((dataElement & 0xff00)>> 8);

            //debug(" recherche :"+

Tools.toHex(testByte1)+Tools.toHex(testByte2)+Tools.toHex(testByte3)+

Tools.toHex(testByte4));

            for ( ;  j< LenMax || found ; j++){
                  if(   (   data[j]   ==   testByte1   )   &&   (data[j+1]   ==
testByte2) &&

(data[j+2]== testByte3) && (data[j+3]== testByte4)){
                              found = true ;
                              return j ;
                  }//endif
            }
            return -1 ;
      }fim


       private int MessageKey( int i){
            int i0 =    (int) (data[i ] &0xff);
            int i1 =    (int) (data[i + 1 ] &0xff);
            int i2 =    (int) (data[i + 2 ] &0xff);
            int i3 =    (int) (data[i + 3 ] &0xff);

       return i1<<24| i0<<16 |i3<< 8|i2 ;
```

```
            //return  data[i  +1]<<24|data[i  ]  <<16  |data[i  +  3]  <<
8|data[i+2] ;
      }

       private int readMessageLength(int i){
            int i0 =    (int) (data[i ] &0xff);
            int i1 =    (int) (data[i + 1 ] &0xff);
            int i2 =    (int) (data[i + 2 ] &0xff);
            int i3 =    (int) (data[i + 3 ] &0xff);
            return i3<<24| i2<<16 |i1<< 8|i0 ;
            //return  data[i  +3]<<24|data[i+2]  <<16  |data[i  +  1]  <<
8|data[i];
      }

       private int readInt32(int i) {
            return readMessageLength(i);//same method !
      }

       private int readInt16( int i ){

                  int  i1 = data[i+1]&0xff ;
                  int  i0 = data[i]&0xff ;
            // int anInt = (data[i+1]&0xff)<< 8| (data[i]&0xff) ;
             int anInt =  i1<<8|i0 ;
             //debug(" depuis readInt16   iest egal a"+ i);
             //debug("On  lit  ::"  +  anInt    +"  data[i+1]    i1  "+i1+"
data[i] i0  "+ 0) ;
             if   (anInt   <   -1){   anInt=   (int)(data[i]*256)&0xff   +
data[i+1]&0xff ;//swap

byte
                        debug("On  relit  :attention  erreur  de  VR  !!!:" +
anInt) ;
                        }
            return anInt ;
      }

      private void skip (int length){
```

```
            index += length;
      }


      public int  getAnInt(int groupElement, int dataElement){
            int pos = lookForMessagePosition( groupElement, dataElement);
            if(pos < maxHeaderSize && pos != -1 ){
            //debug( "message length :" + readMessageLength(pos+4) );
                  if (readMessageLength(pos+4) == 2 )
                        return readInt16( pos + 8 );
                  else if(readMessageLength(pos+4) == 4 )
                        return readInt32(pos + 8 );
                  else { // no caso de VR explicito
                              int h = readInt16( pos + 8 );
                              if( h < 2500) return  h ;
                              else return -1 ;
                  }//endelse
                  }//endif(pos
            else return -1 ;
      }//endofMethod

            private int  getAnInt(int groupElement, int dataElement, int
j){
            int pos = lookForMessagePosition( groupElement, dataElement,
j);
            if(pos < maxHeaderSize && pos != -1 ){
                  index = pos ;
            //debug( "message length :" + readMessageLength(pos+4) );
                  if (readMessageLength(pos+4) == 2 )
                        return readInt16( pos + 8 );
                  else if(readMessageLength(pos+4) == 4 )
                        return readInt32(pos + 8 );
                  else { // case we meet an explicit VR case .
                              int h = readInt16( pos + 8 );
                              if( h < 2500) return  h ;
                              else return -1 ;
                  }
                  }
```

```
            else return -1 ;
      }//endofMethod



/*metodos para acessar os atributos da imagem */


      public int  getSize()   { return dataLength;}

      public int getNumberOfFrames(){ return numberOfFrames ;}

      public int getSamplesPerPixels() { return samplesPerPixel ;}


      public int getPixelDataSize(){
                  if (errorDetector == 0 )return size ;
                  else return -1 ;
      }



      public int  getRows()   {return h ;}

      public int  getColumns(){ return w;}

      public int  getBitAllocated(){return bitsAllocated;}

      public int  getBitStored(){return bitsStored;}

      public int  getHighBit(){return    highBit;}

      public int  getSamplesPerPixel(){return  samplesPerPixel;}

      public int  getPixelRepresentation(){return    signed;}
```

```
/* metodos para acessar as tags DICOM */

      public String  getPatientName(){    return patientName;          }
      public String  getPatientBirthdate(){ return patientBirthdate;   }
      public String  getManufacturer(){   return manufacturer ;        }
      public String  getPatientID(){            return patientID  ;
      }
      public String  getImageType(){            return imageType;
            }
      public String  getStudyDate(){            return studyDate;
            }
      public String  getModality(){             return modality;
      }


      public String  getaString(int groupElement, int dataElement){
            int pos = lookForMessagePosition(groupElement,dataElement);
            if(pos < maxHeaderSize  && pos != -1 ){
                  int length = readMessageLength(pos+4);


                  if (length >256) length = readInt16( pos + 6 );
                  if (length > 64 || length < 0 ) length = 64 ;
                  if (length> (dataLength - pos-8)) length = dataLength -
pos -9 ;
                  char[] patName = new char[length ];
                  pos += 8;
                  for (int i = 0; i < length ;i++)
                              patName[i] = (char)data[pos++];
                        return new String(patName );
                  }
            else return " Unknown " ;
      }


      private String  getaString(int groupElement, int dataElement, int j
){
            int pos = lookForMessagePosition(groupElement,dataElement,j);
            //debug(    "Position       =    "    +    pos+    "    of    "    +
tools.Tools.toHex(groupElement ) +"

"+ tools.Tools.toHex(dataElement ) );
```

```
            if(pos < MAX_HEADER_SIZE && pos != -1 ){
                  int length = readMessageLength(pos+4);
                  if (length >256)length = readInt16( pos + 6 ); ;
                  if (length > 64 || length < 0 ) length = 64 ;
                  if (length > (dataLength - pos-8)) length = dataLength
-pos -9 ;
                  index = pos  ;
                  pos += 8;
                  char[] patName = new char[length ];
                  //debug("groupElmt"+     Tools.toHex(groupElement)     +
"length"

+String.valueOf(length));
                  for (int i = 0; i < length ;i++)
                              patName[i] = (char)data[pos++];
                        return new String(patName );
                  }
            else return " Unknown " ;
      }



   public int  getFileDataLength(){
            int pos = lookForMessagePosition( 0x7Fe0, 0x0010);
            if ( pos != -1) return  readMessageLength(pos+4) ;
            else return -1 ;
      }




/**
*     <BR>
* This method return the pixels of the Dicom image ( yes  !)  provided it
is not compressed

and this is a known
* Dicom format, else it return null (please check for a null condition
or an index array
```

```
out of bounds exception).
*     </BR>
*/

      public byte[] getPixels() throws IOException{


            if (VR == _JPEGCompression )
                  throw new IOException("DICOM JPEG compression not yet
supported ") ;

            int w = getRows() ;
                  if ( w ==  -1){
                        throw new IOException("Format not recognized") ;

                         //return null ;
                  }
            int h = getColumns();
                  if(  h  ==  -1)  throw  new  IOException("Format  not
recognized") ;

            int ba = getBitAllocated();
                  if( ba%8 == 0) { ba = ba/8 ;}
                        else ba = (ba+8)/8 ;
            int fileLength = w * h * ba ;
            int offset = dataLength - fileLength ;

            byte[]  pixData = new byte[ fileLength ];
                  java.lang.System.arraycopy  (data,  offset,  pixData,0,
fileLength );
            return pixData ;

      }

      public byte[] getPixels(int number) throws IOException{

            if( number > numberOfFrames ) throw new IOException( "Doesn't
have such a
```

```
frame ! ") ;
            if (VR == _JPEGCompression )
                  throw new IOException("DICOM JPEG compression not yet
supported ") ;

            int w = getRows() ;
                  if ( w ==  -1){
                        throw new IOException("Format not recognized") ;

                         //return null ;
                  }
            int h = getColumns();
                  if(  h  ==  -1)  throw  new  IOException("Format  not
recognized") ;

            int ba = getBitAllocated();
                  if( ba%8 == 0) { ba = ba/8 ;}
                        else ba = (ba+8)/8 ;
            int fileLength = w * h * ba ;
            int offset = dataLength - (fileLength * number);
            byte[]  pixData = new byte[ fileLength ];
                  java.lang.System.arraycopy  (data,  offset,  pixData,0,
fileLength );
            return pixData ;

      }



      public String[] getInfo(){

            String [] info = new String[16];

            info[0]   =  "Patient  's  name                        :  "  +
getPatientName();
            info[1]   =  "Patient  's  ID                          :  "  +
getPatientID();
            info[2]    =  "Patient  's  birthdate                  :  "  +
getPatientBirthdate();
            info[3]  = "Patient 's sex              : " + sex;
```

```
            info[4]   =  "Study  Date                                  :  "  +
getStudyDate();
            info[5]  = "Modality                      : " + getModality();
            info[6]   =  "Manufacturer                                 :  "  +
getManufacturer();
            info[7]   =  "Number  of  frames                           :  "  +
getNumberOfFrames();
            info[8]  = "Width                         : " + getColumns();
            info[9]  = "Height                        : " + getRows();
            info[10]  =  "Bits  allocated                              :  "  +
getBitAllocated();
            info[11]  =  "Bits  stored                                 :  "  +
getBitStored();
            info[12]  =  "Sample  per  pixels                          :  "  +
getSamplesPerPixel();
            info[13] = "Physician                     : " + physician;
            info[14] = "Institution                   : " + institution;
            info[15] = "Transfert syntax UID          : " + xSyntaxUID ;


      return info ;
      }

      private void  debug(String message){
            if (debug )System.out.println(message);
      }
}// fim da classe
```

```
/*=====================================================================*\
|*********************************************************************|
|  “Método baseado em processamento digital de imagens para diagnóstico |
|                 precoce de micro-estruturas dentárias”               |
|          (Classe para leitura de imagens DICOM sem compressão)       |
|              Mestrando: Yuji de Oliveira Ohnishi                     |
|              Orientador: Prof. Dr. Paulo Estevão Cruvinel            |
|              Co-Orientador: Prof. Dr. Walter Antônio de Almeida      |
|*********************************************************************|
\*=====================================================================*/

      package DicomDecoder ;
      import java.awt.*;
      import java.awt.image.* ;
      import java.util.*;
      import tools.Tools;
      import java.net.* ;
      import java.io.*;


public class DicomReader{
      int w, h , highBit, n , size ; // highBit is for littleEndian
      boolean signed ;
      final static boolean DEBUG = false ;
      boolean ignoreNegValues ;
      int bitsStored, bitsAllocated ;
      int samplesPerPixel ;
      int numberOfFrames ;
      byte[] pixData;
      String filename ;
      DicomHeaderReader dHR ;


      public     DicomReader(     DicomHeaderReader     dHR     )throws
java.io.IOException{
            this.dHR = dHR ;
            h                    = dHR.getRows() ;
            w                    = dHR.getColumns() ;
```

```
            highBit          = dHR.getHighBit() ;
            bitsStored       = dHR.getBitStored() ;
            bitsAllocated    = dHR.getBitAllocated();
            n                      = (bitsAllocated/8)    ;// = 1 ou 2
            signed                 = (dHR.getPixelRepresentation() == 1)
;
            size             =  h * w;
            samplesPerPixel = dHR.getSamplesPerPixel()  ;
            this.pixData     = dHR.getPixels();
            ignoreNegValues = true  ;
            samplesPerPixel = dHR.getSamplesPerPixel()  ;
            numberOfFrames   = dHR.getNumberOfFrames() ;
            dbg("Number of Frames " + numberOfFrames) ;
      }


      public DicomReader(byte[] array )throws java.io.IOException{
            this(new DicomHeaderReader(array));
      }


      public DicomReader(URL url)throws java.io.IOException {

            URLConnection u = url.openConnection();
            int size         =          u.getContentLength() ;
            byte[] array     = new byte[size];
            int bytes_read   = 0;
            DataInputStream  in = new DataInputStream(u.getInputStream())
;
            while(bytes_read < size){
                  bytes_read   +=   in.read(array,   bytes_read,   size   -
bytes_read);
                              }
            in.close();

            this.dHR  = new DicomHeaderReader(array);
            h                      = dHR.getRows() ;
            w                      = dHR.getColumns() ;
```

```
            highBit           = dHR.getHighBit() ;
            bitsStored        = dHR.getBitStored() ;
            bitsAllocated     = dHR.getBitAllocated();
            n                       = (bitsAllocated/8)     ;// = 1 ou 2
            signed                  = (dHR.getPixelRepresentation() == 1)
;
            size              =  h * w;
            this.pixData      =  dHR.getPixels();
            ignoreNegValues = true  ;
            samplesPerPixel = dHR.getSamplesPerPixel()  ;
            numberOfFrames    = dHR.getNumberOfFrames() ;
            dbg("Number of Frames " + numberOfFrames) ;
            }


      public DicomReader( byte[] pixels,
                                 int w,
                                 int h,
                                 int highBit,
                                 int bitsStored,
                                 int bitsAllocated,
                                 boolean signed,
                                 int samplesPerPixel ,
                                 int numberOfFrames,
                                 boolean ignoreNegValues ){

            this.h                       =     h;
            this.w                       =     w;
            this.highBit           =     highBit;
            this.bitsStored   =     bitsStored ;
            this.bitsAllocated     =     bitsAllocated;
            this.n                       =     bitsAllocated/8   ;//  =  1
ou 2
            this.signed            =     signed ;
            this.size              =     h * w;
            this.pixData           =     pixels ;
            this.ignoreNegValues =  ignoreNegValues  ;
            this.samplesPerPixel =  samplesPerPixel  ;
```

```
            this.numberOfFrames     =    numberOfFrames      ;


            }
////////////////////////////////////////////////////////////////////////
//////////////////
      public int getNumberOfFrames(){ return numberOfFrames ;}


public String[]    getInfos(){

      return dHR.getInfo() ;
}




      public Image getImage(){

            if (w > 1024){
                  dbg(" w > 1024 " + "  width  : "+w+ "   height  : "+h)
;
                  return scaleImage() ;
             }

            ColorModel cm = grayColorModel() ;
            dbg("  width  : "+w+ "   height  : "+h) ;
            if( n == 1){// in case it's a  8 bit/pixel image
                  return       Toolkit.getDefaultToolkit().createImage(new
MemoryImageSource(w, h,cm, pixData, 0, w));

            }//endif



            else if ( !signed) {
                  byte[] destPixels = to8PerPix( pixData) ;
                  return       Toolkit.getDefaultToolkit().createImage(new
MemoryImageSource(w, h,cm,destPixels, 0, w));
                                    }
            else if (signed ){
```

```
                byte[] destPixels =     signedTo8PerPix( pixData) ;
                return       Toolkit.getDefaultToolkit().createImage(new
MemoryImageSource(w, h,cm,destPixels, 0, w));
                }

          else return null ;
     }// end of getImage .


////////////////////////////////////////////////////////////////////////
/////
     public DicomHeaderReader  getDicomHeaderReader() { return dHR; }


     public Image[]    getImages() throws IOException{
          Image[] images = new Image[numberOfFrames - 1 ] ;
          for( int  i = 1 ; i == numberOfFrames ; i++ ){
                pixData = dHR.getPixels(i);
                images[i-1] = getImage() ;
                }


          return images ;
     }
////////////////////////////////////////////////////////////////////////

     protected Image scaleImage(){
          ColorModel cm = grayColorModel() ;
          int scaledWidth = w/2 ;
          int scaledHeight = h/2 ;
          int index =0 ;
          int value = 0 ;
          byte[] destPixels = null;
          System.gc() ;

          if(n ==1 ){//1 byte/pixel
                destPixels = new byte[scaledWidth * scaledHeight];
                for(int i = 0 ; i<h ; i+=2){
```

```
                        for(int j = 0 ; j<w ; j+=2){
                              destPixels[index++] = pixData[(i*w) + j] ;
                        }
                  }
                  pixData = null;
                  Tools.gc("PIXDATA == NULL");
                  return       Toolkit.getDefaultToolkit().createImage(new
MemoryImageSource(w/2, h/2,cm, destPixels, 0, w/2));
            }     //endIf(n==1)
                  // suppose n == 2 and unsigned value
            else if(n==2 && bitsStored<=8){
                  dbg("w =   "+ w + "  h ==  "+ h);
                  dbg("PixData.length = "+ pixData.length );
                  dbg(" h * w  =  " + ( h * w )) ;
                  destPixels = new byte[w * h] ;
                  int len = w * h;
                  for (int i = 0 ; i< len ; i++ ){
                              value =     (int)(pixData[i*2])&0xff ;
                              destPixels[i] = (byte)value;

                  }//END FOR
                  pixData = null;
                  Tools.gc("PIXDATA == NULL");
                  return       Toolkit.getDefaultToolkit().createImage(new
MemoryImageSource(w, h,cm, destPixels, 0, w));

            }//end Elseif

            else if(!signed){
                  int[] intPixels = new int[scaledWidth * scaledHeight] ;
                  dbg(" !signed");
                  int maxValue = 0;
                  int minValue = 0xffff ;
                        if(highBit >= 8){
                        for(int i = 0 ; i<h ; i+=2){
                              for(int j = 0 ; j<w ; j+=2){
                              value  =  ((int)(  pixData[(2*(i*w+j))+1]  &
0xff )<<8)| (int)( pixData[2*(i*w+j)] & 0xff)  ;
```

```
                        if(value>maxValue) maxValue = value;
                        if(value< minValue) minValue = value;
                              intPixels[index++] = value ;
                        }
                  }

            }//endif
            int scale = maxValue-minValue ;
                  if( scale == 0 ) scale = 1;
            pixData = null;
            //Tools.gc( "pix Data null") ;
            destPixels = new byte[scaledWidth * scaledHeight];
            for (int i =0 ; i< intPixels.length ; i++){
                  value = (intPixels[i] - minValue )*256;
                  value /=scale ;
                  destPixels[i] = (byte)(value&0xff);
                  }
            intPixels = null ;
            //Tools.gc("iNTPixels ==null") ;
            return       Toolkit.getDefaultToolkit().createImage(new
MemoryImageSource(w/2, h/2,cm, destPixels, 0, w/2));
      }//endElseif(!signed...)
      else if(signed){
            byte[] pixels = signedTo8PerPix(pixData) ;
            pixData = pixels;
            for(int i = 0 ; i<h ; i+=2){
                  for(int j = 0 ; j<w ; j+=2){
                        destPixels[index++] = pixData[(i*w) + j] ;
                  }
            }
      pixData = null;
      //Tools.gc( "pixData = null") ;
      return             Toolkit.getDefaultToolkit().createImage(new
MemoryImageSource(w/2, h/2,cm, destPixels, 0, w/2));
      }

return null ;
}
```

```
//////////////////////////////////////////////////////////////////////
//////////////////
      private byte[] to8PerPix(byte[] pixData){
      // suppose n == 2 and unsigned value
            if(bitsStored<=8){//Special case for Philips
                  dbg("w =   "+ w + "  h ==  "+ h);
                  dbg("PixData.length = "+ pixData.length );
                  dbg(" h * w  =  " + ( h * w )) ;
                  byte[] destPixels = new byte[w * h] ;
                  int len = w * h;
                  int value = 0;
                  for (int i = 0 ; i< len ; i++ ){
                              value =     (int)(pixData[i*2])&0xff ;
                              destPixels[i] = (byte)value;
                  }
            return destPixels ;
            }//

            int[] pixels = new int[w*h] ;
            int value =0;
            // case littleEndian or highBit  and unsigned ;
            if(highBit >= 8){

                  for( int i = 0 ; i< pixels.length ; i++){
                        value  =  ((int)(  pixData[(2*i)+1]  &  0xff  )<<8)|
(int)( pixData[(2*i)] & 0xff)  ;//msb first !
                        pixels[i] = value  ;
                  }
            }
            // case bigEndian and unsigned :
            else if( highBit <= 7){
                  //System.out.println("DicomReader.to8PerPix  highBit  ==
7 ");
                  for( int i = 0 ; i< pixels.length ; i++){
                        value  =  ((int)(  pixData[(2*i)]  &  0xff  )<<8)|
(int)( pixData[(2*i)+1] & 0xff)  ;//lsb first !
                        pixels[i] = value ;
```

```
                }
            }
            //look for the Max value      and minValue
                int maxValue = 0;
                int minValue = 0xffff;
                for ( int i = 0 ; i < pixels.length ; i++){
                    if ( pixels[i] > maxValue) maxValue = pixels[i] ;
                    if ( pixels[i] < minValue ) minValue = pixels[i] ;
                }
            // setUp a new grayScale :
                int scale = maxValue - minValue ;
                if(scale == 0) {scale =1 ;
                    System.out.println("DicomReader.to8PerPix   :scale
== error ");}
                byte[] destPixels = new byte[w * h] ;
                for (int i = 0 ; i < pixels.length ; i++ ){
                    value = ((pixels[i] - minValue )*255 ) /scale ;
                    destPixels[i] = (byte)(value&0xff);
                    //pixels[i]  =  (255<<24)|(  value<<16)|(value<<8)|
value ;
                    }

                return destPixels ;

     }//endOfMethod to8PerPix
////////////////////////////////////////////////////////////////////////
//////////////////
            private  byte[] signedTo8PerPix    (byte[] pixData){
            int[] pixels = new int[w * h] ;
            short shValue = 0 ;
            int value = 0 ;
            // case signed and  littleEndian :
            if ( highBit >= 8 ){
                for (int i = 0 ; i < pixels.length ; i++ ){
                shValue = (short)((( pixData[(2*i)+1] & 0xff )<<8)| (
pixData[(2*i)] & 0xff) ) ;//msb first !
                    value = (int ) shValue ;
                    if(value<0 && ignoreNegValues ) value = 0 ;
                    pixels[i] = value  ;
```

```
                        }
                }
                // case signed and  bigEndian :
                if ( highBit <= 7 ){
                      for (int i = 0 ; i < pixels.length ; i++ ){
                            shValue  =  (short)(((  pixData[(2*i)+1]  &  0xff
)<<8)| ( pixData[(2*i)] & 0xff) ) ;//msb first !
                            value = (int) shValue ;
                            if(value<0 && ignoreNegValues ) value = 0 ;
                            pixels[i] = value  ;
                            }
                      }
                //look for the Max value      and minValue
                int maxValue = 0;
                int minValue = 0xffff;
                for ( int i = 0 ; i < pixels.length ; i++){
                      if ( pixels[i] > maxValue) maxValue = pixels[i] ;
                      if ( pixels[i] < minValue ) minValue = pixels[i] ;

                }
                byte[] destPixels = new byte[w*h] ;
                int scale = maxValue - minValue ;
                      if(scale == 0){ scale = 1 ; System.out.println(" Error
in VR form SignedTo8..DicomReader");}
                for (int i = 0 ; i < pixels.length ; i++ ){
                      value = ((pixels[i] - minValue )*255) /scale ;
                      //pixels[i]  =  (255<<24)|(  value<<16)|(value<<8)|  value
;
                      destPixels[i] =(byte)(value&0xff) ;
                }
                return destPixels ;
          }
//////////////////////////////////////////////////////////////////////////
//////////////////////
          protected  ColorModel grayColorModel(){
                byte[] r = new byte[256] ;
                for (int i = 0; i <256 ; i++ )
                      r[i] = (byte)(i & 0xff ) ;
          return (new IndexColorModel(8,256,r,r,r));
```

```
      }

////////////////////////////////////////////////////////////////////////
////////////////////
      public void flush(){
            pixData = null ;
            System.gc();
            System.gc();
      }
      void dbg(String s){
             if(DEBUG) System.out.println(this.getClass().getName() + s);
            }
}// fim da classe .
```

```
/*=========================================================================*\
|*************************************************************************|
|  “Método baseado em processamento digital de imagens para diagnóstico |
|                 precoce de micro-estruturas dentárias”                  |
|                     (Classe para segmentacao da imagem)                 |
|              Mestrando: Yuji de Oliveira Ohnishi                        |
|              Orientador: Prof. Dr. Paulo Estevão Cruvinel               |
|              Co-Orientador: Prof. Dr. Walter Antônio de Almeida         |
|*************************************************************************|
\*=========================================================================*/

import java.util.*;
import java.awt.*;
import javax.jai.*;
import java.awt.image.*;

/** Segmentation class **/
class SegmentedImage
{
      int width,snapshotWidth;
      int height,snapshotHeight;
      int centerX;
      int centerY;
      int[] pixels,snapshotPixels;
      int bgColor=0;

      ImageProcessor(Image picture)
      {
            //loading image
            waitForImage(picture);

            //Get the width and height of the image
            width=picture.getWidth(this);
            height=picture.getHeight(this);
            centerX=Math.round(width/2);
            centerY=Math.round(height/2);

        //Allocate buffer to hold the image's pixels
```

```
            pixels = new int[width * height];

            //Grab pixels
            PixelGrabber pg = new PixelGrabber(picture, 0, 0, width,
height, pixels, 0, width);
            try {
                  pg.grabPixels();
            } catch (InterruptedException e){};
      }

      void waitForImage(Image picture)
      {
            //Make sure the image is fully loaded
            MediaTracker tracker = new MediaTracker(this);
            tracker.addImage(picture, 0);
            try {
                  tracker.waitForID(0);
            } catch (InterruptedException e){};
      }

      Image createImage()
      {
          Image img= createImage(new MemoryImageSource(width, height,
pixels, 0, width));
        //Make sure the pixels pointer changes after createImage, silly
but necessary
      int[] sourcePixels=pixels;
      pixels = new int[width * height];
        System.arraycopy(sourcePixels,0,pixels,0,width*height);
        return img;
      }

      Color getColor(int x, int y)
      {
        int c=pixels[y*width+x];
        int r=(c&0xff0000)>>16;
        int g=(c&0xff00)>>8;
        int b=c&0xff;
```

```
        return (new Color(r,g,b));
      }

      void draw(Graphics gc,int xOffset, int yOffset)
      {
        //Java for Windows cannot take advantage of thousands or more
colors
        //when printing an image, but this routine will
        //It will make the images look pretty bad in 256 color though...
        int index=0;
        for (int y=yOffset;y<height+yOffset;y++)
             for (int x=xOffset;x<width+xOffset;x++)
             {
                  int c=pixels[index++];
                  int r=(c&0xff0000)>>16;
                  int g=(c&0xff00)>>8;
                  int b=c&0xff;

                  gc.setColor (new Color(r,g,b));
                  gc.drawLine(x,y,x,y);
             }
      }

      void snapshot()
      {
            //Make a snapshot of the current image
            snapshotWidth=width;
            snapshotHeight=height;
            snapshotPixels = new int[width * height];
        System.arraycopy(pixels,0,snapshotPixels,0,width*height);
      }

      void reset()
      {
            //Reset the image from snapshot
            if (snapshotPixels!=null)
            {
```

```
                width=snapshotWidth;
            height=snapshotHeight;
            pixels = new int[width * height];
            System.arraycopy(snapshotPixels,0,pixels,0,width*height);
            centerX=Math.round(width/2);
            centerY=Math.round(height/2);
      }
      }

      boolean reset(int percent)
      {
            //Check to see if the source and destination images have the
same propotions
            if((width==snapshotWidth)&&(height==snapshotHeight))
            {

        if (percent==50)
        {
            //Optimize for percent=50
             for (int index=0;index<width*height;index++)

pixels[index]=((pixels[index]&0xf0f0f0)>>1)+((snapshotPixels[index]&0xf0f
0f0)>>1)+(pixels[index]&0xff000000);
        }
        else
        {
                   int percent2=100-percent;
             for (int index=0;index<width*height;index++)
             {

                //Read colors from the destination image
                  int c1=pixels[index];
                  int r1=(c1&0xff0000)>>16;
                  int g1=(c1&0xff00)>>8;
                  int b1=c1&0xff;

                //Read colors from the source image
                  int c2=snapshotPixels[index];
```

```
                int r2=(c2&0xff0000)>>16;
                int g2=(c2&0xff00)>>8;
                int b2=c2&0xff;

                //Calculate the new color value
                int r=(r1*percent2+r2*percent)/100;
                int g=(g1*percent2+g2*percent)/100;
                int b=(b1*percent2+b2*percent)/100;

                pixels[index]=(c2&0xff000000)+(r<<16)+(g<<8)+b;
            }
           }
            //Successful reset
            return true;
      }
      else
      {
            //Reset failed
            return false;
      }
    }


    /* Filters start here */

//    void makeTransparent(int r,int g,int b,double percent)
//    {
//      percent*=1.28;
//
//      //Makes given color within the given range transparent
//      int rMin=(int)(r-percent);
//      int gMin=(int)(g-percent);
//      int bMin=(int)(b-percent);
//
//      int rMax=(int)(r+percent);
//      int gMax=(int)(g+percent);
//      int bMax=(int)(b+percent);
```

```
//
//      for (int index=0;index<width*height;index++)
//      {
//            int c=pixels[index];
//            r=(c&0xff0000)>>16;
//            g=(c&0xff00)>>8;
//            b=c&0xff;
//            if
((r>=rMin)&&(r<=rMax)&&(g>=gMin)&&(g<=gMax)&&(b>=bMin)&&(b<=bMax))
//                  pixels[index]=c&0xffffff;
//      }
//    }


      void invert()
      {
            for (int index=0;index<width*height;index++)
                  pixels[index]=pixels[index]^0xffffff;
      }


//    void gray()
//    {
//          for (int index=0;index<width*height;index++)
//          {
//            int c=pixels[index];
//            int r=(c&0xff0000)>>16;
//            int g=(c&0xff00)>>8;
//            int b=c&0xff;
//            int gray=(r*3+g*4+b*2)/9;  //calculate the right gray value
based on r, g and b intensities
//            pixels[index]=(c&0xff000000)+(gray<<16)+(gray<<8)+gray;
//          }
//    }


/*    void transparency(double percent)
      {
```

```
      int trans=(((int)(255.0-(percent*2.55)))<<24)&0xff000000;
      for (int index=0;index<width*height;index++)
           pixels[index]=trans+(pixels[index]&0xffffff);
    }*/

//  void transparency(int percent)
//  {
//    for (int index=0;index<width*height;index++)
//    {
//         int c=pixels[index];
//         int a=(c>>24)&0xff;
//
//         a=Math.min(255,(a*percent)/100);
//
//         pixels[index]=(c&0xffffff)+(a<<24);
//    }
//  }



    void brightness(int percent)
    {
      for (int index=0;index<width*height;index++)
      {
           int c=pixels[index];
           int r=(c&0xff0000)>>16;
           int g=(c&0xff00)>>8;
           int b=c&0xff;

           r=Math.min(255,(r*percent)/100);
           g=Math.min(255,(g*percent)/100);
           b=Math.min(255,(b*percent)/100);

           pixels[index]=(c&0xff000000)+(r<<16)+(g<<8)+b;
      }
    }
```

```
//    void brightness(int percentR,int percentG,int percentB)
//    {
//      for (int index=0;index<width*height;index++)
//      {
//           int c=pixels[index];
//           int r=(c&0xff0000)>>16;
//           int g=(c&0xff00)>>8;
//           int b=c&0xff;
//
//           r=Math.min(255,(r*percentR)/100);
//           g=Math.min(255,(g*percentG)/100);
//           b=Math.min(255,(b*percentB)/100);
//
//           pixels[index]=(c&0xff000000)+(r<<16)+(g<<8)+b;
//      }
//    }


      void contrast(double percent)
      {
        percent+=100.0;
        double i=1.0-(percent/100.0);
        double j=percent/100.0;

        for (int index=0;index<width*height;index++)
        {
             int c=pixels[index];
             int r=(c&0xff0000)>>16;
             int g=(c&0xff00)>>8;
             int b=c&0xff;

             r=Math.min(Math.max((int)(i+j*r),0),255);
             g=Math.min(Math.max((int)(i+j*g),0),255);
             b=Math.min(Math.max((int)(i+j*b),0),255);

             pixels[index]=(c&0xff000000)+(r<<16)+(g<<8)+b;
```

```
        }
      }


//    void button(int borderSize,int intensity, boolean pushed)
    //{
//      if (pushed) intensity=-intensity;
//
//      //Draw top and bottom border
//      for (int y=0;y<borderSize;y++)
//           for (int x=0;x<width-y;x++)
//           {
//                //Draw top border
//                int c=pixels[x+y*width];
//                int r=(c&0xff0000)>>16;
//                int g=(c&0xff00)>>8;
//                int b=c&0xff;
//
//                r=Math.max(Math.min(r+intensity,255),0);
//                g=Math.max(Math.min(g+intensity,255),0);
//                b=Math.max(Math.min(b+intensity,255),0);
//
//                pixels[x+y*width]=(c&0xff000000)+(r<<16)+(g<<8)+b;
//
//                //Draw bottom border
//                c=pixels[width-x-1+(height-y-1)*width];
//                r=(c&0xff0000)>>16;
//                g=(c&0xff00)>>8;
//                b=c&0xff;
//
//                r=Math.min(Math.max(r-intensity,0),255);
//                g=Math.min(Math.max(g-intensity,0),255);
//                b=Math.min(Math.max(b-intensity,0),255);
//
//                pixels[width-x-1+(height-y-
1)*width]=(c&0xff000000)+(r<<16)+(g<<8)+b;
//      }
//
```

```
//      //Draw side borders
//      for (int x=0;x<borderSize;x++)
//           for (int y=borderSize;y<height-x;y++)
//           {
//                //Draw left border
//                int c=pixels[x+y*width];
//                int r=(c&0xff0000)>>16;
//                int g=(c&0xff00)>>8;
//                int b=c&0xff;
//
//                r=Math.max(Math.min(r+intensity,255),0);
//                g=Math.max(Math.min(g+intensity,255),0);
//                b=Math.max(Math.min(b+intensity,255),0);
//
//                pixels[x+y*width]=(c&0xff000000)+(r<<16)+(g<<8)+b;
//
//                //Draw right border
//                c=pixels[width-x-1+(y-(borderSize-x))*width];
//                r=(c&0xff0000)>>16;
//                g=(c&0xff00)>>8;
//                b=c&0xff;
//
//                r=Math.min(Math.max(r-intensity,0),255);
//                g=Math.min(Math.max(g-intensity,0),255);
//                b=Math.min(Math.max(b-intensity,0),255);
//
//                pixels[width-x-1+(y-(borderSize-
x))*width]=(c&0xff000000)+(r<<16)+(g<<8)+b;
//
//      }
//    }


//
//    boolean merge(Image sourcePicture,int percent)
//    {
//          //First make sure the source image is fully loaded
//          waitForImage(sourcePicture);
//
```

```
//          //Check to see if the source and destination images have the
same propotions
//
      if((sourcePicture.getWidth(this)==width)&&(sourcePicture.getHeight(
this)==height))
//          {
//            //Allocate buffer to hold the source image's pixels
//            int[] sourcePixels = new int[width * height];
//
//            //Grab pixels
//            PixelGrabber pg = new PixelGrabber(sourcePicture, 0, 0,
width, height, sourcePixels, 0, width);
//            try {
//                  pg.grabPixels();
//            } catch (InterruptedException e){};
//
//           int percent2=100-percent;
//
//           for (int index=0;index<width*height;index++)
//           {
//                int c1=pixels[index];
//                int r1=(c1&0xff0000)>>16;
//                int g1=(c1&0xff00)>>8;
//                int b1=c1&0xff;
//
//                int c2=sourcePixels[index];
//                int r2=(c2&0xff0000)>>16;
//                int g2=(c2&0xff00)>>8;
//                int b2=c2&0xff;
//
//                int r=(r1*percent2+r2*percent)/100;
//                int g=(g1*percent2+g2*percent)/100;
//                int b=(b1*percent2+b2*percent)/100;
//
//                pixels[index]=(c2&0xff000000)+(r<<16)+(g<<8)+b;
//           }
//           //Successful merge
//           return true;
//      }
```

```
//      else
//      {
//           //Merge failed
//           return false;
//      }
//    }


    void noise(float percent)
    {
      Random rnd=new Random();

      for (int index=0;index<width*height;index++)
      {
       int c=pixels[index];
       int r=(c&0xff0000)>>16;
       int g=(c&0xff00)>>8;
       int b=c&0xff;

       int RandomBrightness=(int)(rnd.nextFloat()*percent)+100;

       r=Math.min((r*RandomBrightness)/100,255);
       g=Math.min((g*RandomBrightness)/100,255);
       b=Math.min((b*RandomBrightness)/100,255);

       pixels[index]=(c&0xff000000)+(r<<16)+(g<<8)+b;
      }
    }


      void pseudoColors(long seed)
      {
        Random rnd=new Random();
     // rnd.setSeed(seed);
        int random=rnd.nextInt();
        random= random&0xff00ff;
```

```
  for (int index=0;index<width*height;index++)
  {
       int c=pixels[index];
       pixels[index]=(c&0xff000000)+((c|random)&0xffffff);
  }
}

public void crop(int x, int y,int w, int h)
{
   //Duplicate image
   int[] sourcePixels=pixels;
 //System.arraycopy(pixels,0,sourcePixels,0,width*height);
   pixels = new int[w * h];

   int index=0;
   for (int ys=y;ys<y+h;ys++)
        for (int xs=x;xs<x+w;xs++)
             pixels[index++]=sourcePixels[ys*width+xs];
 width=w;
 height=h;
}

public void move(int dx, int dy)
{
  //Duplicate image
  int[] sourcePixels=pixels;
  pixels = new int[width * height];

  int index=0;
  for (int y=0;y<height;y++)
       for (int x=0;x<width;x++)
       {
            int xs=x-dx;
            int ys=y-dy;
            if ((xs>=0)&&(xs<width)&&(ys>=0)&&(ys<height))
              pixels[index++]=sourcePixels[width*ys+xs];
            else
```

```
                pixels[index++]=bgColor;
            }
        }


        public void move(int dx, int dy,int mask)
        {
          //Duplicate image
          int[] sourcePixels=pixels;
          pixels = new int[width * height];

        //Make an inverted color mask
        mask=mask&0xffffff;
        int iMask=(mask^0xffffff)+0xff000000;

          int index=0;
          for (int y=0;y<height;y++)
               for (int x=0;x<width;x++)
               {
                    int xs=x-dx;
                    int ys=y-dy;
                    if ((xs>=0)&&(xs<width)&&(ys>=0)&&(ys<height))

pixels[index]=(sourcePixels[index]&iMask)+(sourcePixels[width*ys+xs]&mask
);
                    else

pixels[index]=(sourcePixels[index]&iMask)+(bgColor&mask);
              index++;
               }
        }


        void zoom(int percent)
        {
          //Duplicate image
          int[] sourcePixels=pixels;
          pixels = new int[width * height];
```

```
  int index=0;
  for (int y=-centerY;y<centerY;y++)
       for (int x=-centerX;x<centerX;x++)
       {
            int xs=(x*percent)/100+centerX;
            int ys=(y*percent)/100+centerY;
            if (xs>=0&&xs<width&&ys>=0&&ys<height)
              pixels[index++]=sourcePixels[width*ys+xs];
            else
              pixels[index++]=bgColor;
       }
}


void reSize(int xsize, int ysize)
{
  //Duplicate image
  int[] sourcePixels=pixels;
  pixels = new int[xsize * ysize];

  int index=0;
  double Xstep=(double)width/(double)xsize;
  double Ystep=(double)height/(double)ysize;

  for (int y=0;y<ysize;y++)
       for (int x=0;x<xsize;x++)
       {
            int xs=(int)(x*Xstep);
            int ys=(int)(y*Ystep);
            pixels[index++]=sourcePixels[ys*width+xs];
       }

width=xsize;
height=ysize;
  centerX=Math.round(width/2);
  centerY=Math.round(height/2);
```

```
    }


  void explode(double time)
  {
    //Duplicate image
    int[] sourcePixels=pixels;
    pixels = new int[width * height];
    int[] zBuffer = new int[width * height];
    Random rnd=new Random();
      //Make sure the same sequence of random numbers are returned for
each call to explode
    rnd.setSeed(0);

    int index=0;
    //This is the y value added because of gravity
    double yOffset=((time+1.0)*(time*1.0))/10.0;

    for (int y=-centerY;y<centerY;y++)
          for (int x=-centerX;x<centerX;x++)
          {
                int c=sourcePixels[index];
                if ((c&0xff000000)!=0)
                {
                    //Calculate distance from center, or the 'bomb'
                      double d=Math.sqrt(x*x+y*y);
                      //Pixel's z, or depth, value
                      double z=(rnd.nextDouble()+1)*time/d;
                      //Calculate x and y destination, add perspective
                      int xd=(int)(x*z+x)+centerX;
                      int yd=(int)((y+yOffset)*z+y)+centerY;
                  //Calculate the pixels offset in the pixels array
                      int offset=xd+yd*width;
                      if (xd>=0&&xd<width&&yd>=0&&yd<height)
                       //Check to see if another pixel blocks this
                            if (zBuffer[offset]<=z)
                            {
```

```
                                    //The  pixel  is  unblocked,  plot  and
update z, or depth, value
                                      pixels[offset]=c;
                                      zBuffer[offset]=(int)z;
                                }
            }
            index++;
      }
    }


      void rotate(double angle)
      {
        //Duplicate image
        int[] sourcePixels=pixels;
        pixels = new int[width * height];

        //Convert  from  degrees  to  radians  and  calculate  cos  and  sin  of
angle
      //Negate the angle to make sure the rotation is clockwise
        double angleRadians=-angle/(180.0/Math.PI);
        double ca=Math.cos(angleRadians);
        double sa=Math.sin(angleRadians);

        int index=0;
        for (int y=-centerY;y<centerY;y++)
        {
             for (int x=-centerX;x<centerX;x++)
             {
                  int xs=(int)(x*ca-y*sa)+centerX;
                  int ys=(int)(y*ca+x*sa)+centerY;
                  if ((xs>=0)&&(xs<width)&&(ys>=0)&&(ys<height))
                    pixels[index++]=sourcePixels[width*ys+xs];
                  else
                    pixels[index++]=bgColor;
             }
        }
      }
```

```
    void spiral(double angle)
    {
      //Duplicate image
      int[] sourcePixels=pixels;
      pixels = new int[width * height];

      double angleRadians=-angle/(180.0/Math.PI);
      double maxDist=Math.sqrt(width*width+height*height);
      double scale=angleRadians/maxDist;

      int index=0;
      for (int y=-centerY;y<centerY;y++)
           for (int x=-centerX;x<centerX;x++)
           {
                double a=Math.sqrt(x*x+y*y)*scale;
                double ca=Math.cos(a);
                double sa=Math.sin(a);

                int xs=(int)(x*ca-y*sa)+centerX;
                int ys=(int)(y*ca+x*sa)+centerY;
                if (xs>=0&&xs<width&&ys>=0&&ys<height)
                  pixels[index++]=sourcePixels[width*ys+xs];
                else
                  pixels[index++]=bgColor;
           }
    }


//    void ripple(double nWaves,double percent,double offset)
//    {
//      //Duplicate image
//      int[] sourcePixels=pixels;
//      pixels = new int[width * height];
//
//      double angleRadians=(Math.PI*2.0*percent)/100.0;
```

```
//      double maxDist=Math.sqrt(width*width+height*height);
//      double scale=(Math.PI*2.0*nWaves)/maxDist;
//      offset=(offset*Math.PI*2.0)/100.0;
//
//      int index=0;
//      for (int y=-centerY;y<centerY;y++)
//           for (int x=-centerX;x<centerX;x++)
//           {
//                double
a=Math.sin(Math.sqrt(x*x+y*y)*scale+offset)*angleRadians;
//                double ca=Math.cos(a);
//                double sa=Math.sin(a);
//
//                int xs=(int)(x*ca-y*sa)+centerX;
//                int ys=(int)(y*ca+x*sa)+centerY;
//                if (xs>=0&&xs<width&&ys>=0&&ys<height)
//                  pixels[index++]=sourcePixels[width*ys+xs];
//                else
//                  pixels[index++]=bgColor;
//           }
//    }
//
//
//    void verticalWave3D(double nWaves,double percent, double offset)
//    {
//        //Duplicate image
//        int[] sourcePixels=pixels;
//        pixels = new int[width * height];
//
//        double angleRadians=(offset*3.6)/(180.0/Math.PI);
//
//        double angleStep=((Math.PI*2)*nWaves)/height;
//        double amplitude=(percent*height/2)/100.0;
//
//        int y=0;
//        for (int offset1=0;offset1<width*height;offset1+=width)
//        {
```

```
//              int                       offset2=(Math.max(Math.min((int)(
(Math.cos(angleRadians)*amplitude+y)  ),height-1),0))*width;
//            y++;
//
System.arraycopy(sourcePixels,offset2,pixels,offset1,width);
//            angleRadians+=angleStep;
//        }
//    }
//
//    void horizontalWave3D(double nWaves,double percent, double offset)
//    {
//        //Duplicate image
//        int[] sourcePixels=pixels;
//        pixels = new int[width * height];
//
//        double angleRadians=(offset*3.6)/(180.0/Math.PI);
//
//        double angleStep=((Math.PI*2)*nWaves)/width;
//        double amplitude=(percent*height/2)/100.0;
//
//        for (int x=0;x<width;x++)
//        {
//              int                       xOffset=Math.max(Math.min((int)(
(Math.cos(angleRadians)*amplitude+x) ),width-1),0);
//            for (int yOffset=0;yOffset<width*height;yOffset+=width)
//                pixels[x+yOffset]=sourcePixels[xOffset+yOffset];
//            angleRadians+=angleStep;
//        }
//    }
//
//
//    void horizontalWave(double nWaves,double percent,double offset)
//    {
//        //Duplicate image
//        int[] sourcePixels=pixels;
//        pixels = new int[width * height];
//
//          double waveFrequency=(nWaves*Math.PI*2.0)/height;
```

```
//          double waveOffset=(offset*nWaves*Math.PI*2.0)/100.0;
//          double radius=(width*percent)/100.0;
//
//          int index=0;
//          for (int y=0;y<height;y++)
//        {
//                int
xOffset=(int)Math.round(Math.sin(y*waveFrequency+waveOffset)*radius);
//                for (int x=0;x<width;x++)
//                {
//                      if (xOffset>=0&&xOffset<width)
//                          pixels[index++]=sourcePixels[xOffset+width*y];
//                      else
//                          pixels[index++]=bgColor;
//                      xOffset++;
//                }
//        }
//    }
//
//
//    void verticalWave(double nWaves,double percent,double offset)
//    {
//      //Duplicate image
//      int[] sourcePixels=pixels;
//      pixels = new int[width * height];
//
//      double waveFrequency=(nWaves*Math.PI*2.0)/height;
//      double waveOffset=(offset*nWaves*Math.PI*2.0)/100.0;
//      double radius=(width*percent)/100.0;
//
//      int index=0;
//      for (int x=0;x<width;x++)
//      {
//           int
yOffset=(int)Math.round(Math.sin(x*waveFrequency+waveOffset)*radius);
//           for (int y=0;y<height;y++)
//           {
//                if (yOffset>=0&&yOffset<height)
```

```
//                  pixels[width*y+x]=sourcePixels[width*yOffset+x];
//                else
//                  pixels[width*y+x]=bgColor;
//                yOffset++;
//           }
//      }
//    }


      void flipVertical()
      {
          //Duplicate image
            int[] sourcePixels=pixels;
            pixels = new int[width * height];

        int lastLine=width*height-width;
          for (int offset=0;offset<width*height;offset+=width)
              System.arraycopy(sourcePixels,offset,pixels,lastLine-
offset,width);
    }


      void flipHorizontal()
      {
          //Duplicate image
        int[] sourcePixels=pixels;
          pixels = new int[width * height];

          for (int x=0;x<width;x++)
          {
                int offset=width-1-x;
              for (int yoffset=0;yoffset<width*height;yoffset+=width)
                  pixels[x+yoffset]=sourcePixels[offset+yoffset];
          }
    }


      void lineArt(int intensity)
```

```
      {
            //Duplicate image
            int[] sourcePixels=pixels;
            pixels = new int[width * height];

            //The first line is undefined, give it the color of bgColor
                  for           (int           index=0;index<width;index++)
pixels[index]=bgColor;

            for (int index=width;index<width*height;index++)
            {
                  //Read pixel to the left
                  int c=sourcePixels[index-1];
                  int r1=(c&0xff0000)>>16;
                  int g1=(c&0xff00)>>8;
                  int b1=c&0xff;

                  //Read pixel above
                  c=sourcePixels[index-width];
                  int r2=(c&0xff0000)>>16;
                  int g2=(c&0xff00)>>8;
                  int b2=c&0xff;

                  //Read current pixel
                  c=sourcePixels[index];
                  int r=(c&0xff0000)>>16;
                  int g=(c&0xff00)>>8;
                  int b=c&0xff;

                  r=Math.min((Math.abs(r2-r)+Math.abs(r1-
r))*intensity,255);
                  g=Math.min((Math.abs(g2-g)+Math.abs(g1-
g))*intensity,255);
                  b=Math.min((Math.abs(b2-b)+Math.abs(b1-
b))*intensity,255);

                  pixels[index]=(c&0xff000000)+(r<<16)+(g<<8)+b;
            }
      }
```

```
      void emboss(double angle,double power,int red,int green, int blue)
      {
            //Duplicate image
            int[] sourcePixels=pixels;
            pixels = new int[width * height];

          double angleRadians=angle/(180.0/Math.PI);
        int
light=(int)(Math.round(Math.sin(angleRadians))*width)+(int)(Math.round(Ma
th.cos(angleRadians)));

            for (int index=width+1;index<width*height-width-1;index++)
            {
                  //Read current pixel
                  int c=sourcePixels[index];
                  int r1=(c&0xff0000)>>16;
                  int g1=(c&0xff00)>>8;
                  int b1=c&0xff;

                  //Read pixel in the direction given by angle
                  c=sourcePixels[index-light];
                  int r2=(c&0xff0000)>>16;
                  int g2=(c&0xff00)>>8;
                  int b2=c&0xff;

                  int                   r=Math.min(Math.max(red+(int)((r2-
r1)*power),0),255);
                  int                 g=Math.min(Math.max(green+(int)((g2-
g1)*power),0),255);
                  int                  b=Math.min(Math.max(blue+(int)((b2-
b1)*power),0),255);

                  pixels[index]=(c&0xff000000)+(r<<16)+(g<<8)+b;
            }

            //Borders are undefined, fill with specified color
        int color=(red<<16)+(green<<8)+blue;
        for (int index=0;index<width;index++)
```

```
        {
            pixels[index]=(sourcePixels[index]&0xff000000)+color;
            pixels[width*height-index-1]=(sourcePixels[width*height-
index-1]&0xff000000)+color;
        }
        for (int index=0;index<width*height;index+=width)
        {
            pixels[index]=(sourcePixels[index]&0xff000000)+color;
            pixels[index+width-1]=(sourcePixels[index+width-
1]&0xff000000)+color;
        }
    }


    void blur(int power)
    {

            int[] pixels2;
            int i, x, y, offset;
            int  rgb,  rgb1,  rgb2,  rgb3,  rgb4,  rgb5,  rgb6,  rgb7,  rgb8,
rgbmean;
            int rsum = 0, gsum = 0, bsum = 0;

            pixels2 = new int[width * height];
            for (i = 0; i < width * height; i++)
                  pixels2[i] = pixels[i];
            for (y = 1; y < (height - 1); y++) {
                  for (x = 1; x < (width - 1); x++) {
                        offset = x + y * width;
                        rgb1 = pixels2[offset - width - 1];
                        rgb2 = pixels2[offset - width];
                        rgb3 = pixels2[offset - width + 1];
                        rgb4 = pixels2[offset + 1];
                        rgb5 = pixels2[offset + width + 1];
                        rgb6 = pixels2[offset + width];
                        rgb7 = pixels2[offset + width - 1];
                        rgb8 = pixels2[offset - 1];
                        rgb = pixels2[offset];
```

```
                        rsum = (rgb1 & 0xff0000) + (rgb2 & 0xff0000) +
(rgb3 & 0xff0000) + (rgb4 & 0xff0000) + (rgb5 & 0xff0000)
                                + (rgb6 & 0xff0000) + (rgb7 & 0xff0000) +
(rgb8 & 0xff0000) + (rgb & 0xff0000);
                        gsum = (rgb1 & 0xff00) + (rgb2 & 0xff00) + (rgb3 &
0xff00) + (rgb4 & 0xff00) + (rgb5 & 0xff00)
                                + (rgb6 & 0xff00) + (rgb7 & 0xff00) + (rgb8
& 0xff00) + (rgb & 0xff00);
                        bsum = (rgb1 & 0xff) + (rgb2 & 0xff) + (rgb3 &
0xff) + (rgb4 & 0xff) + (rgb5 & 0xff)
                                + (rgb6 & 0xff) + (rgb7 & 0xff) + (rgb8 &
0xff) + (rgb & 0xff);
                        rgbmean = ((rsum/9) & 0xff0000) | ((gsum/9) &
0xff00) | (bsum/9);
                        pixels[offset] = (rgb & 0xff000000) | (rgbmean &
0xffffff);
                }
            }
//          //Duplicate image
//          int[] sourcePixels=pixels;
//          pixels = new int[width * height];
//
//        int index=0;
//        for (int y=0;y<height;y++)
//            for (int x=0;x<width;x++)
//                {
//                    int red=0, green=0, blue=0,nColors=0;
//                    for (int dy=-power+y;dy<=power+y;dy++)
//                        for (int dx=-power+x;dx<=power+x;dx++)
//                            if
((dx>=0)&&(dx<width)&&(dy>=0)&&(dy<height))
//                            {
//                                  int c=sourcePixels[dx+dy*width];
//                                  red+=(c&0xff0000)>>16;
//                                  green+=(c&0xff00)>>8;
//                                  blue+=c&0xff;
//                                nColors++;
//                            }
//                red/=nColors;
//                green/=nColors;
```

```
//                blue/=nColors;
//
      pixels[index]=(sourcePixels[index]&0xff000000)+(red<<16)+(green<<8)
+blue;
//                      index++;
//                }
   }

      void sharpen() {
            int[] pixels2;
            int i, x, y, offset;
            int  rgb,  rgb1,  rgb2,  rgb3,  rgb4,  rgb5,  rgb6,  rgb7,  rgb8,
newrgb;
            int rsum = 0, gsum = 0, bsum = 0;

            pixels2 = new int[width * height];
            for (i = 0; i < width * height; i++)
                  pixels2[i] = pixels[i];
            for (y = 1; y < (height - 1); y++) {
                  for (x = 1; x < (width - 1); x++) {
                        offset = x + y * width;
                        rgb1 = pixels2[offset - width - 1];
                        rgb2 = pixels2[offset - width];
                        rgb3 = pixels2[offset - width + 1];
                        rgb4 = pixels2[offset + 1];
                        rgb5 = pixels2[offset + width + 1];
                        rgb6 = pixels2[offset + width];
                        rgb7 = pixels2[offset + width - 1];
                        rgb8 = pixels2[offset - 1];
                        rgb = pixels2[offset];
                        rsum = ((rgb & 0xff0000) >> 16) * 12 - ((rgb1 &
0xff0000) >> 16) - ((rgb2 & 0xff0000) >> 16)
                                    - ((rgb3 & 0xff0000) >> 16) - ((rgb4 &
0xff0000) >> 16) - ((rgb5 & 0xff0000) >> 16)
                                    - ((rgb6 & 0xff0000) >> 16) - ((rgb7 &
0xff0000) >> 16) - ((rgb8 & 0xff0000) >> 16);
                        rsum /= 4;
                        if (rsum < 0)
                              rsum = 0;
```

```
                        if (rsum > 255)
                              rsum = 255;
                        gsum =  ((rgb & 0xff00) >> 8) * 12 - ((rgb1 &
0xff00) >> 8) - ((rgb2 & 0xff00) >> 8)
                              - ((rgb3 & 0xff00) >> 8) - ((rgb4 & 0xff00)
>> 8) - ((rgb5 & 0xff00) >> 8)
                              - ((rgb6 & 0xff00) >> 8) - ((rgb7 & 0xff00)
>> 8) - ((rgb8 & 0xff00) >> 8);
                        gsum /= 4;
                        if (gsum < 0)
                              gsum = 0;
                        if (gsum > 255)
                              gsum = 255;
                        bsum =  ((rgb & 0xff) * 12) - (rgb1 & 0xff) -
(rgb2 & 0xff) - (rgb3 & 0xff) - (rgb4 & 0xff)
                               - (rgb5 & 0xff) - (rgb6 & 0xff) - (rgb7 &
0xff) - (rgb8 & 0xff);
                        bsum /= 4;
                        if (bsum < 0)
                              bsum = 0;
                        if (bsum > 255)
                              bsum = 255;
                        newrgb = ((rsum << 16) & 0xff0000) | ((gsum << 8 )
& 0xff00) | bsum;
                        pixels[offset] = (rgb & 0xff000000) | (newrgb &
0xffffff);

                  }
            }
      }
}
```