GUIDO STOLFI

PROCESSOS DE COMPRESSÃO DE DADOS APLICADOS A IMAGENS MÉDICAS

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção de título de Mestre em Engenharia.

São Paulo
2000

GUIDO STOLFI

PROCESSOS DE COMPRESSÃO DE DADOS APLICADOS A IMAGENS MÉDICAS

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção de título de Mestre em Engenharia.

Área de Concentração:
Sistemas Eletrônicos

Orientador:
Prof. Dr. Geraldo Lino de Campos

São Paulo
2000

à memória dos meus pais.

A G R A D E C I M E N T O S

Ao amigo e orientador, Prof. Dr. Geraldo Lino de Campos, pelas diretrizes e aconselhamentos;

Ao prof. Dr. Paul Jean Etienne Jeszensky, pelo incentivo persistente;

Ao amigo Klaus Koster, pelas colaborações inestimáveis na programação;

Aos colegas do PTC pelo estímulo constante.

Índice

SIGLAS E ABREVIATURAS

CCITT Comité Consultatif Internationale de Téléphonie et de Télégraphie (Comitê Consultor Internacional de Telefonia e Telegrafia)

DAT Digital Audio Tape (Fita de Áudio Digital, também para gravação de dados)

DCT Discrete Cosine Transform (Transformada Discreta de Cosenos)

DMA Distorção Média Absoluta (Somatória do Módulo das Diferenças)

DVD Digital Versatile Disc (Disco Óptico Digital de Aplicações Múltiplas)

EOB End Of Block (Indicador de Fim de Bloco)

GOP Group Of Pictures (Grupo de Imagens em Seqüência MPEG)

HIS Hospital Information System (Sistema de Informação Hospitalar)

IEC International Electrotechnical Comission (Comissão Internacional de Eletrotécnica)

InCor Instituto do Coração

ISO International Standards Organization (Organização Internacional de Padronização)

JPEG Joint Photographers Experts Group (Grupo de Especialistas em Fotografia, CCITT - ISO)

MPEG Moving Picture Experts Group (Grupo de Especialistas em Imagens em Movimento, ISO - IEC)

PACS Picture Archiving and Communication Systems (Sistema de Arquivamento e Comunicação de Imagens)

RLE Run-lenght Encoding (Codificação por seqüência de símbolos)

RMS Root Mean Square (Raiz da Média Quadrática)

VHS Video Home System (Sistema de Vídeo Doméstico)

R E S U M O

O sistema de codificação de vídeo MPEG é largamente utilizado para transmissão de imagens na área de Televisão Digital e Multimídia, onde proporciona altas taxas de compressão de dados.

A distribuição e arquivamento de imagens utilizadas em diagnóstico médico, por outro lado, estabelecem outro nível de exigências quanto à qualidade das imagens e à adequação a características próprias.

Este trabalho propõe novos processos e parâmetros de codificação, baseados no padrão MPEG, direcionados para atender requisitos específicos de compressão de imagens médicas geradas de forma seqüencial.

A análise dos resultados experimentais obtidos, a partir de uma seqüência de Cineangiografia, mostra que estas técnicas permitem obter taxas de compressão de dados superiores a 10:1 com elevada qualidade de reconstrução de imagens, proporcionando melhor desempenho que o uso dos métodos padronizados.

A B S T R A C T

The MPEG Video encoding standard is widely used for Multimedia and Digital Television transmission, allowing high data compression rates with high quality.

On the other side, image archiving and distribution for medical purposes deserves a higher expectancy level, both in image quality and adequacy to specific requirements.

Here we propose new encoding processes and parameters, based on MPEG standards, tailored to the specific requirements for the compression of sequentially generated medical images.

Analysis of experimental data, obtained from a Cineangiography digital video series, shows that we can obtain over a 10:1 compression rate with high image rendering quality. Overall performance levels so obtained are better than using standard MPEG methods.

1. - Introdução

O diagnóstico através de imagens representa uma importante ferramenta da medicina moderna, principalmente por constituir-se de uma técnica de exame não-invasiva. A aplicação de métodos de análise e processamento digital de imagens proporciona ao médico informações extremamente importantes, sem a necessidade de intervenções cirúrgicas, ampliando assim a conveniência de exames convencionais, como a Radiologia e Ultra-sonografia.

Uma das técnicas mais importantes no diagnóstico de moléstias cardiovasculares é a **Cineangiografia**. Ela consiste na obtenção de uma seqüência cinematográfica de radiografias do miocárdio, sendo que, através da injeção de líquido contrastante nas coronárias, é possível efetuar um registro dinâmico do fluxo sangüíneo nestas artérias, permitindo assim o acompanhamento do processo de irrigação do músculo cardíaco.

Este exame auxilia na detecção de obstruções nas coronárias; porém, sua importância maior está no auxílio aos procedimentos cirúrgicos utilizados na remoção dessas obstruções, como por exemplo a angioplastia e o cateterismo. Nestas intervenções, é indispensável o acompanhamento dos processos, em tempo real, através das imagens fornecidas pelo equipamento de Cineangiografia.

Um equipamento de Cineangiografia consiste de um conjunto móvel de emissor e sensor de raio X, que é posicionado de forma a fornecer uma imagem adequada da região cardíaca do paciente. O sensor de imagem, normalmente utilizando intensificador de imagem e câmera CCD, possui resolução típica de 512 x 512 a 1024 x 1024 elementos, fornecendo em torno de 30 imagens por segundo. Para estudos pediátricos, devido às maiores freqüências cardíacas observadas, pode-se chegar a 60 quadros por segundo.

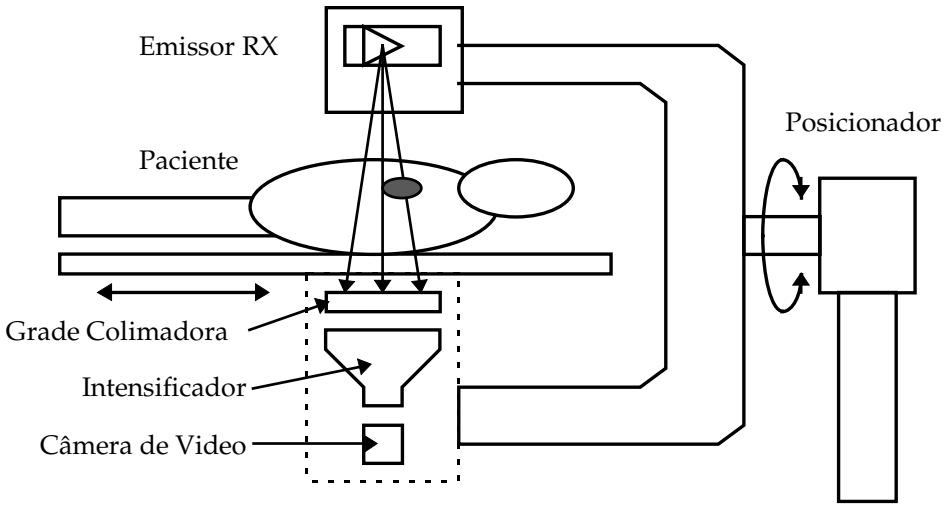


Fig. 1.1 - Equipamento de Cineangiografia

A formação da imagem e o dimensionamento da resolução por amostragem são condicionados, entre outros aspectos, pelos seguintes parâmetros:

- Resolução Limite: a imagem resultante apresenta uma resolução dimensional limitada pela abertura do feixe de raios X, uma vez que a área de emissão destes não é puntiforme. O diâmetro do foco de emissão pode variar de 0,2 a 1 mm, conforme a potência e aplicação do equipamento.

- Ruído: no sentido de limitar a dosagem de radiação à qual o paciente é submetido, a imagem apresenta alto ruído, devido ao baixo número de fótons aproveitáveis para cada elemento de imagem em cada fotograma (ruído quântico). A absorção dos raios X obedece a um processo de Poisson ; já a intensificação e a captação da imagem estão sujeitas a ruído gaussiano. Em comparação com exames radiológicos convencionais, a relação Sinal/Ruído é consideravelmente reduzida. Enquanto uma radiografia pode demandar quantização com 10 ou 12 bits para reproduzir adequadamente as gradações de intensidade, os equipamentos de cineangiografia normalmente operam com 8 bits de resolução, limitando a relação S/R em menos de 45 dB.

- Contraste: a ocorrência de espalhamento dos raios X pelo efeito Compton reduz o contraste final da imagem, sendo necessário o uso de uma grade de colimação entre o paciente e o sensor.

Este trabalho procura estudar a implementação de novos algoritmos de compressão de dados aplicados a imagens dinâmicas, baseados no processo MPEG. Este método de compressão de imagens, já bastante utilizado em televisão e multimídia, utiliza detecção e compensação de movimento (aproveitando a redundância de informação entre fotogramas consecutivos), codificação por transformada espacial, quantização e codificação estatística, obtendo taxas elevadas de compressão em imagens seqüenciais.

1.1 - Motivação

Um exame de Cineangiografia envolve a geração de um volume considerável de dados, que precisam ser armazenados satisfatoriamente de forma a constituir um registro da intervenção para análises e referências futuras. Considerando, por exemplo, uma seqüência de imagens geradas na resolução de 512 × 512 pixels, quantizados em 8 bits, à taxa de 30 quadros por segundo, teremos que armazenar 7,86 MBytes para cada segundo de filmagem, ou cerca de 470 MB por minuto. Outros exames envolvem a geração de imagens seqüenciais, como por exemplo Ressonância Magnética e Tomografia. Apesar da baixa resolução, estes exames envolvem quantidades de dados da ordem de 10 a 50 MB por sessão (RATIB, 1995, p.4).
Este volume de dados representa um inconveniente em duas situações:

⇒ Armazenamento (arquivamento) do exame para referências futuras, estudos e análises *a posteriori* ;
⇒ Transmissão das imagens para diagnóstico à distância ou acompanhamento remoto da intervenção (por exemplo, em aplicações didáticas).

Estas situações ocorrem conjuntamente em Sistemas de Informação Hospitalar (HIS - *Hospital Information Systems*), nos quais imagens de diagnóstico, conjuntamente com informações relacionadas com o paciente e com os procedimentos, são disponibilizadas através de redes de comunicação de dados para estações de trabalho dentro do âmbito de uma instituição hospitalar.

Estes sistemas evoluíram a partir da concepção de Sistemas de Arquivo e Comunicação de Imagens (PACS - *Picture Archiving and Communication Systems*), que são tecnologias em larga escala para intercâmbio de imagens médicas de forma integrada. A tendência de implantação dos PACS em instituições hospitalares tem impulsionado o estudo de técnicas de compressão de dados aplicadas a imagens médicas.

O desenvolvimento e implantação de redes de comunicação digital de banda larga, além de equipamentos de armazenamento de massa de grande capacidade (cartucheiras ópticas, fita DAT, DVD gravável) suportaria em princípio o volume de dados demandado pelo diagnóstico por imagem dentro de uma instituição; no entanto, os seguintes fatores se contrapõem a esta solução:

- A evolução da tecnologia de captura de imagens leva a um aumento de resolução nas imagens digitalizadas, com conseqüente aumento exponencial no volume de dados a ser tratado;

- Novos processos de diagnóstico por imagem tornar-se-ão comuns, ampliando a gama de exames médicos envolvendo registro de imagens digitalizadas;

- A tendência de ampliação do domínio das redes de comunicação de dados, integrando-se à Internet, aliada ainda ao uso incipiente de redes locais de comunicação sem fio, requer economia e escalabilidade de banda passante.

Deste modo pode-se prover acesso virtualmente global a imagens e informações de diagnóstico médico.

1.1.1 - Arquivamento das Imagens

Além do registro em película fotográfica, em vias de obsolescência, utiliza-se freqüentemente arquivamento em fita magnética VHS ou videodisco (meios analógicos). Estes meios introduzem degradações nas imagens, especialmente se os exames tiverem que ser transferidos novamente para formato digital para pesquisa ou outro processamento posterior. No caso do VHS, suas características de banda passante (2.5 MHz para luminância) e Relação Sinal/Ruído (45 dB máx., *ponderada*) degradam a resolução da imagem para não mais que 250 pixels/linha a 7 bits/pixel. Além disso, tanto o VHS quanto o videodisco são formatos de varredura entrelaçada, enquanto que a cineangiografia é inerentemente progressiva. Para gravação é necessário então o desdobramento de cada fotograma (ou quadro completo) em dois campos sucessivos. Como pode ser visto na figura 1.1.1, este processo introduz perda de resolução para objetos em movimento. Finalmente, ao ser congelada a imagem para exame detalhado de um determinado fotograma, tanto o VHS como o videodisco apresentam, para visualização, apenas um único campo da varredura entrelaçada, reduzindo a resolução vertical máxima da imagem para 240 linhas.

Atualmente emprega-se também arquivamento no formato digital, em *Compact Disc* gravável ou fitas DAT de alta capacidade; por outro lado, por serem meios removíveis, dificultam o acesso imediato *on-line* das informações, especialmente o acesso remoto.

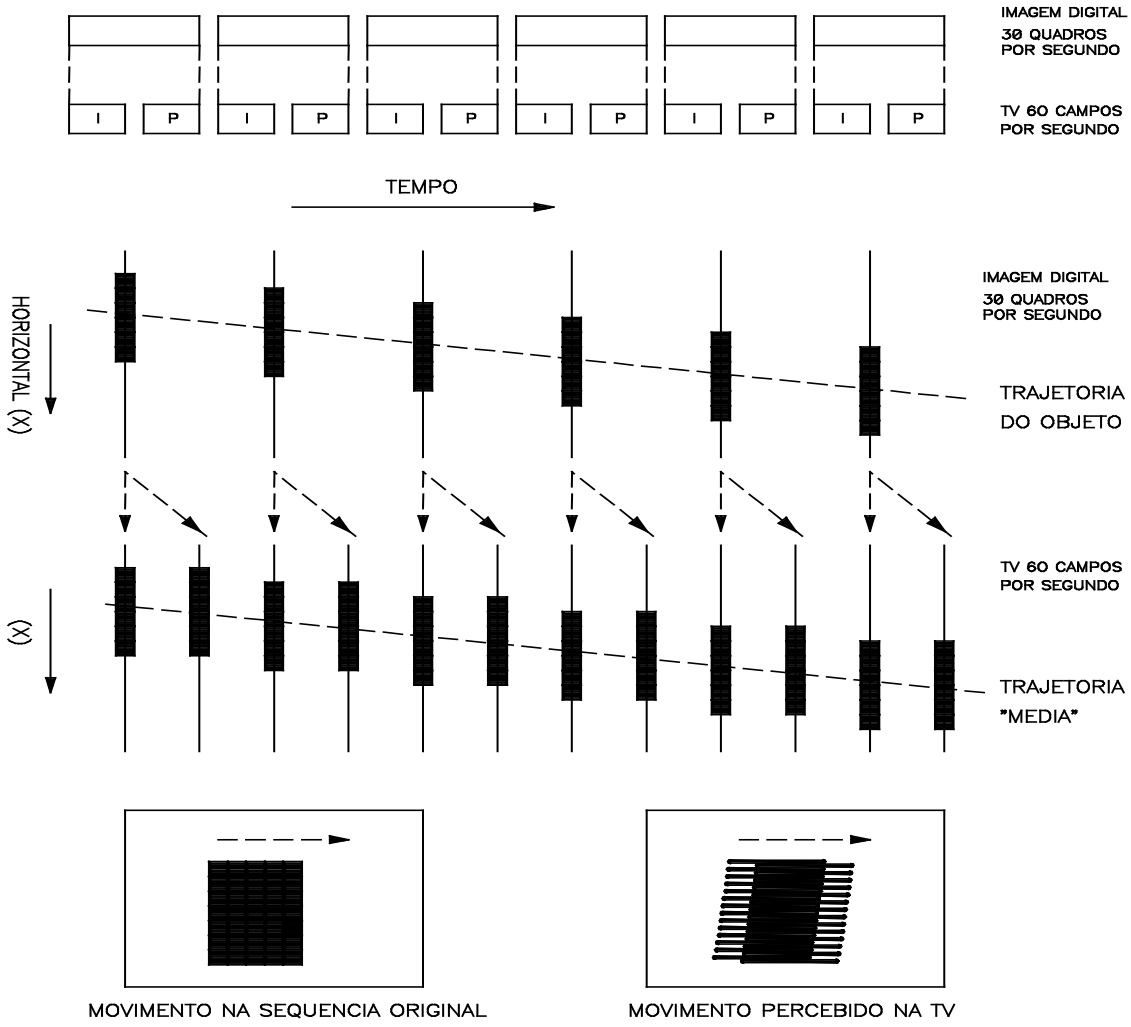


Figura 1.1.1 - Efeito da Duplicação de Campos na Conversão de Varredura Progressiva para Entrelaçada

1.1.2 - Transmissão das Imagens

Dentro de um Sistema Integrado de Informações Médicas, nos moldes de um HIS, torna-se necessário prover o acesso aos resultados de exames através de uma rede de comunicação de dados, tanto dentro do âmbito de uma instituição (via rede local) como remotamente (via Modem, conexão Internet ou rede local

Wireless) (RATIB, 1995, p.6). Embora o uso primordial da Cineangiografia encontre-se na sala de operação, há necessidade de transmissão posterior das imagens para análise dos procedimentos ou planejamento de futuras intervenções e tratamentos. Imediatamente após a realização do exame, antes do seu arquivamento, as imagens geradas são armazenadas temporariamente em um servidor de alta capacidade, sendo acessíveis através de estações de trabalho dentro do hospital.

A taxa de transmissão necessária para visualização em tempo real das imagens de Cineangiografia pode ser determinada por

$$Tr = F \times H \times V \times b / 8$$

Tr = Taxa de Transmissão (Bytes/s)
F = Freqüência de repetição de imagens
H = Número de pixels por linha
V = número de linhas por quadro
b = número de bits por pixel

Para situações típicas, temos $Tr = 30 \times 512 \times 512 \times 8 \ / \ 8 = 7{,}864$ MB/s.

Esta taxa pode ser suportada, de modo contínuo, apenas por redes de alta velocidade (100 Mb/s ou mais).

Quando as taxas de transmissão proporcionadas pelas redes locais convencionais não suportam o volume de dados necessário para transmissão em tempo real, o acesso normalmente é mais demorado, sendo que os dados são antes transferidos integralmente para a estação de trabalho para serem posteriormente visualizados na taxa de imagens correta.

No caso de acesso remoto, onde as taxas de transmissão por Modem muitas vezes não superam 10 a 20 kb/s, a transmissão de um exame completo é impraticável, a menos que se empreguem processos de compressão de dados.

1.2 - Considerações Quanto aos Processos de Compressão

Vimos que tanto a transmissão quanto o arquivamento das imagens geradas num exame de Cineangiografia podem beneficiar-se consideravelmente pela

aplicação de processos de compressão e compactação de dados. Estes processos normalmente consistem de um algoritmo **codificador**, com acesso aos dados brutos, e de um algoritmo **decodificador**, residente na estação de trabalho (figura 1.2.1). Para esta aplicação, são plenamente aceitáveis algoritmos **assimétricos**, ou especificamente, onde o codificador é mais complexo e exige um tempo de processamento maior do que o decodificador.

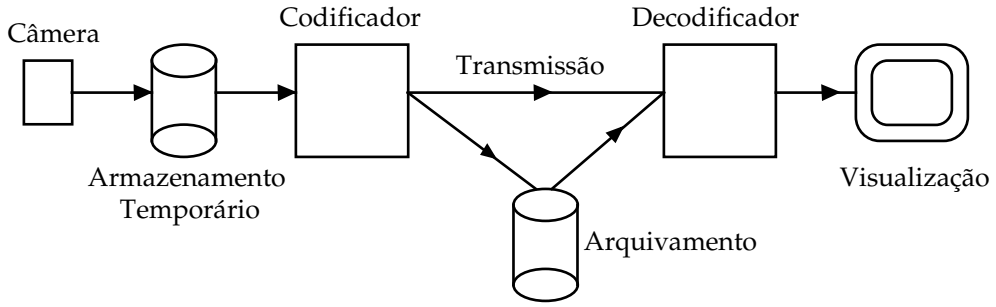


Figura 1.2.1 - Compressão e Visualização de Imagens

Por outro lado, demandam-se qualidade e fidelidade extremas na reprodução de imagens de uso médico, considerando a responsabilidade envolvida no diagnóstico e no auxílio a procedimentos médicos.

Idealmente, o processo de compressão deveria ser **sem perdas**, isto é, não deveríamos ter diferenças entre a imagem original e a imagem reconstruída. No entanto, isso implica em taxas de compressão reduzidas, especialmente no caso de imagens ruidosas (como é o caso nesta aplicação).

Mesmo considerando-se que aspectos legais podem limitar ou inibir o uso de imagens comprimidas, em virtude da hipótese de ocorrerem erros de diagnóstico motivados por artefatos oriundos do processo de compressão, podemos ainda assim aplicar métodos de compressão com perdas para a transmissão de imagens adotando particionamento do fluxo de dados (fig. 1.2.2).

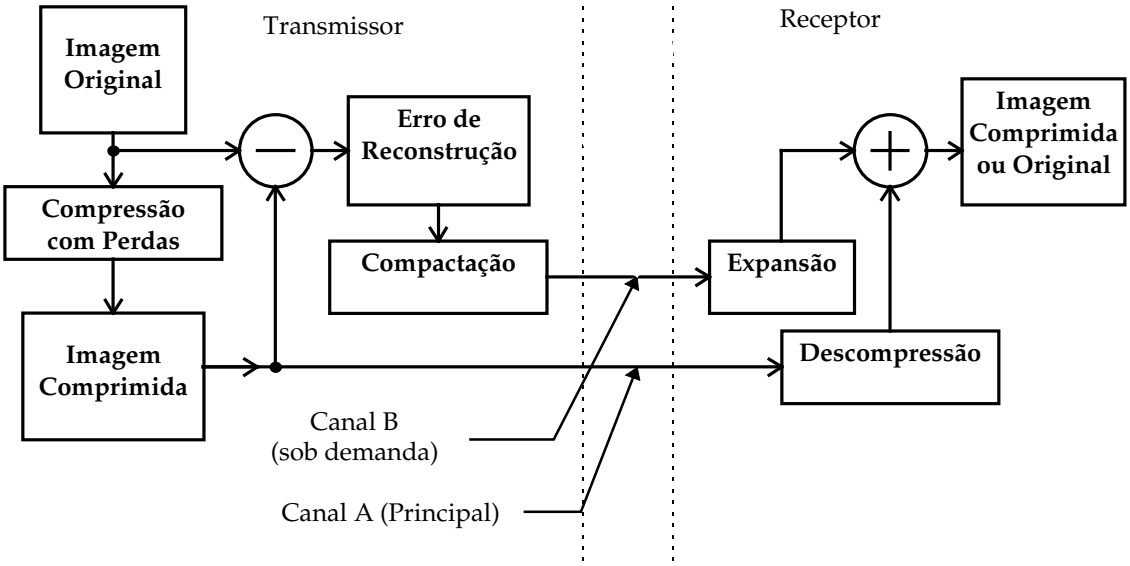


Fig. 1.2.2 - Particionamento do Canal para Transmissão Sem Perdas

Desta forma, uma imagem comprimida (com perdas) pode ser transmitida rapidamente ao receptor, permitindo uma visualização preliminar. As imagens (ou detalhes de uma imagem) que necessitarem de maior atenção seriam complementadas através da transmissão, sob demanda, do erro de reconstrução correspondente. Esta informação adicional será compactada (sem perdas), de modo que o receptor terá à disposição trechos de imagem idênticos às imagens originais.

1.2.1 - Taxa de Compressão × Perdas

A figura 1.2.3 mostra uma imagem típica, extraída de uma seqüência de cine-angiografia (vide também Capítulo 8). Na figura 1.2.4 temos um gráfico da amplitude (luminância) ao longo de uma linha horizontal dessa imagem, e na figura 1.2.5 é apresentado o histograma de intensidades.

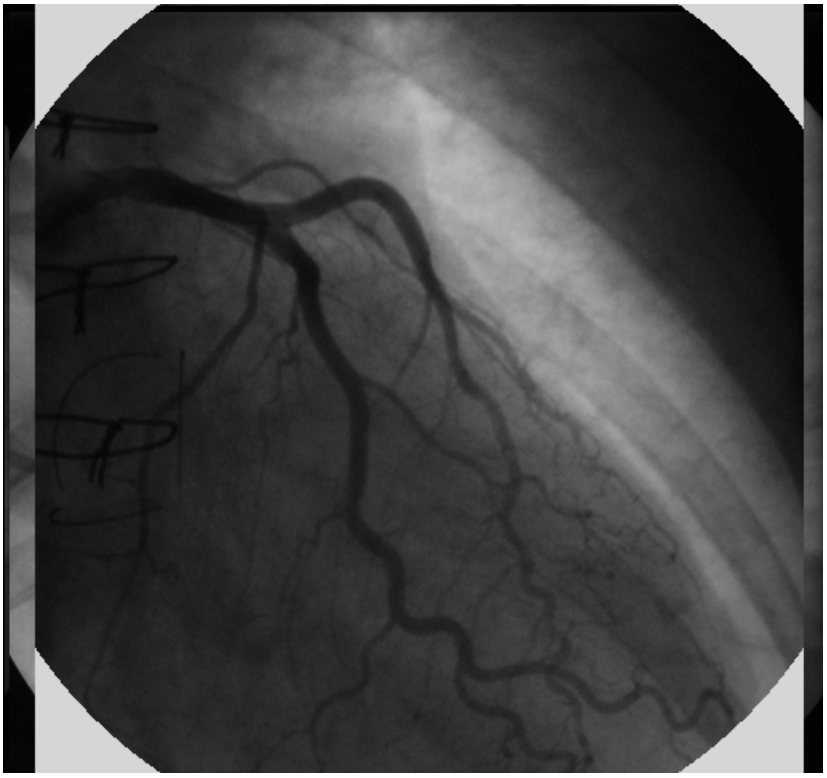


Fig. 1.2.3 - Imagem de Cineangio-coronariografia (XA031.BMP)

Podemos avaliar, pelo gráfico da Fig. 1.2.4, que esta imagem em particular comporta-se como se perturbada por um ruído aleatório aditivo com amplitude de ± 1 a ±2 níveis de quantização. Podemos então dizer que o sinal de luminância $z(x, y)$ representativo da imagem digitalizada pode ser estimado por um sinal "original" $\hat{z}(x, y)$ somado a um ruído aleatório:

$$z(x, y) = \hat{z}(x, y) + N_o$$

Portanto, por melhor que seja o estimador do sinal original $\hat{z}(x, y)$ incorporado ao algoritmo de compressão, haverá sempre uma diferença (aleatória) para a imagem adquirida. O ruído aleatório não pode ser modelado por um preditor, nem portanto comprimido. Considerando que o sinal $z(x, y)$ é expresso usualmente por 8 bits, e que a representação do ruído N_0 exige cerca de 2 bits, podemos afirmar que um processo de compressão sem perdas, aplicado a esta imagem, terá taxa de compressão sempre menor que 4 : 1.

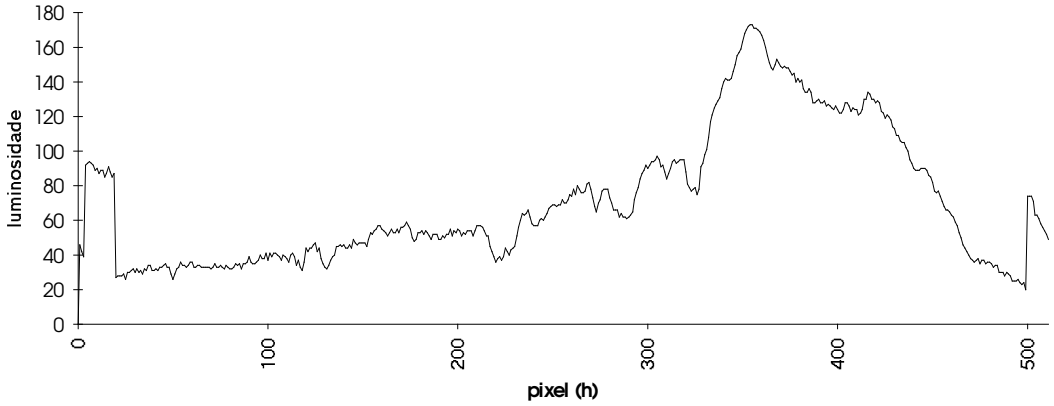


Figura 1.2.4 - Luminância ao longo de uma linha (XA031.BMP)

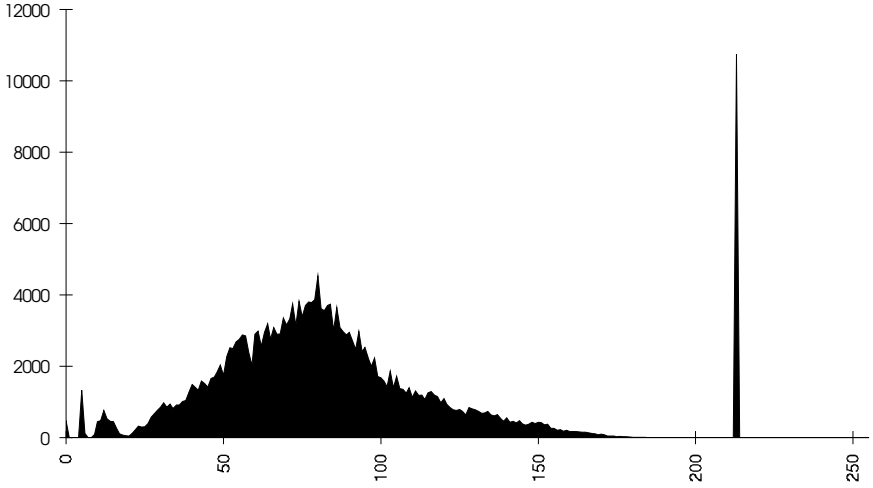


Fig. 1.2.5 - Histograma de Intensidades da Imagem (XA030.BMP)

Como exemplo, se utilizarmos o programa compressor PKZIP (compressão sem perdas de arquivos para uso geral) sobre o arquivo da figura 1.2.3, originalmente em formato BITMAP, teremos o seguinte desempenho:

Arquivo original (.BMP)	246.838 Bytes
Arquivo comprimido (.ZIP)	159.066 Bytes (1,55 : 1)

O rendimento deste processo de compressão não é suficiente para justificar sua aplicação para arquivamento e transmissão digital de imagens.

Uma vez que aceitarmos o uso de processos de compressão com perdas, a questão que deve ser estudada é a de minimizarmos a **visibilidade** dos erros de

reconstrução, de modo a obtermos a maior taxa de compressão que proporcione uma imagem aceitável do ponto de vista do diagnóstico médico. Uma avaliação deste processo envolve então o estudo da forma e das condições pelas quais a imagem é analisada dentro do procedimento clínico.

2. - Objetivos

Os objetivos do presente trabalho envolvem a implementação de um sistema de compressão de dados aplicado ao processamento de imagens seqüenciais para aplicações médicas, como é o caso da Cineangiografia. Na elaboração deste trabalho, propusemos a incorporação dos seguintes elementos:

1. **Detecção e Compensação de Movimento:** no intuito de reduzir a redundância temporal inerente ao processo de captação seqüencial de imagens, incorporamos um algoritmo de busca hierárquica para detecção de movimento, ainda com a finalidade de redução da complexidade computacional (visando aumento na velocidade de processamento).
2. **Detecção de Movimento por Máxima Semelhança:** este método de busca implementado permite reduzir erros de predição de movimento no caso de variações de luminosidade média da imagem, as quais provocam surgimento de vetores de movimento espúrios e conseqüente aumento dos efeitos de "blocagem" , característicos das implementações padrão dos algoritmos MPEG.
3. **Quantização Não-linear:** estudamos o efeito da aplicação de funções não lineares para quantização dos coeficientes resultantes da DCT.
4. **Modelamento do Erro de Reconstrução:** modelamos parâmetros estatísticos do erro de reconstrução, de modo que o decodificador possa equalizar a resposta em freqüência espacial do sistema, para que a imagem reconstruída apresente um ruído com mesmas características da imagem original.
5. **Análise Subjetiva :** além da caracterização por medidas quantitativas (amplitude do erro de reconstrução e taxa efetiva de compressão), os resultados dos vários algoritmos implementados serão avaliados, sempre que possível, por análises e comparações visuais das imagens.

2.1 Resumo das Atividades Realizadas:

Na elaboração do presente trabalho, foram realizadas as atividades abaixo descritas:

1. Coleta de material: foram obtidas no InCor imagens para trabalho, num total de 60 frames (formato BITMAP, resolução de 512 × 480 pixels × 8 bits);
2. Programas para visualização de dados: foi desenvolvido programa em Visual Basic (FSHOW.EXE, imagem completa na resolução de 512 × 480); este programa permite comparar as seqüências original e reconstruída, em movimento ou quadro a quadro, além de visualizar o erro de reconstrução, para várias opções de processamento;
3. Algoritmos de DCT: em linguagem "C", efetuando a transformada direta e inversa.
4. Programa para medida de erro de reconstrução: efetua a diferença entre frames originais e reconstruídos, permitindo análises das perdas do processo.
5. Programa para detecção e compensação de movimento, incorporando DCT direta e inversa, com ou sem busca por máxima semelhança;
6. Quantizador linear e não linear, com fator de quantização ajustável;
7. Algoritmo de reordenação dos coeficientes DCT;
8. Programa para levantamento da probabilidade de ocorrência de símbolos para codificador RLE;
9. Algoritmo para elaboração de um "Codebook" para codificação Huffman;
10. Algoritmo de codificação Huffman e formatação do arquivo comprimido na forma final;
11. Testes de desempenho (taxa de compressão e relação Sinal/Ruído) para várias combinações de processos e quantizadores;
12. Pós-processamento de imagens reconstruídas para redução de ruído.

3. - Estrutura do Processo de Compressão de Imagens Estudado

O processo de compressão pesquisado no presente trabalho baseia-se na codificação MPEG-1 (Moving Picture Experts Group), normatizado pelo ISO/IEC em 1993 (MITCHELL, 1996; HASKELL, 1997). Serão descritas a seguir as características relevantes deste padrão; já no item 3.2 serão apresentadas as características específicas que foram incorporadas e testadas nesta implementação específica.

É importante ressaltar que o processo MPEG foi desenvolvido para Televisão e Multimídia, alcançando nestas aplicações taxas de compressão da ordem de 50:1. Nestes casos, no entanto, a Relação Sinal/Ruído de reconstrução chega a ser menor que 30 dB (HASKELL, 1997, p.175-181). Este valor é tolerável para imagens de TV a cores, em movimento, onde não há a intenção de que o espectador possa examinar detalhadamente uma imagem isolada.

3.1 Padrão MPEG de Compressão de Imagens em Movimento

O padrão MPEG-1 para compressão de vídeo descreve de um codificador com preditor, para seqüências de imagens digitalizadas, baseado nos seguintes princípios:

- Detecção e compensação de movimento por blocos
- Transformada Discreta de Cosenos (DCT) do erro de predição
- Quantização escalável dos coeficientes da DCT
- Codificação entrópica (RLE + Huffman)

Combinando a compensação de movimento com transformada DCT e quantização, chega-se a taxas de compressão de 20:1 ou mais, com baixa *visibilidade* dos erros de reconstrução, dependendo da aplicação.

O padrão MPEG-1 (ISO/IEC 11172a) foi desenvolvido originalmente com vistas à compressão de imagens não-entrelaçadas, para taxas de informação até 1,5 Mb/s (aplicações: vídeo-conferência sobre troncos E1 / T1; multimídia em CD-ROM). Proporciona uma qualidade subjetiva de vídeo comparável à de um gravador doméstico (VHS).

Já o padrão MPEG-2 (ISO 13818) foi desenvolvido em seguida, especificamente para a compressão de imagens de TV entrelaçadas; admite vários níveis de desempenho para taxas entre 1,5 a 100 Mb/s. Encontra aplicações em TV convencional digital, e TV de Alta Definição.

A configuração do codificador MPEG-1, embora não especificada explicitamente no padrão (apenas a sintaxe do fluxo de dados é especificada na norma), segue o diagrama em blocos da figura 3.1.1.

Inicialmente a imagem é formatada (convertida para não-entrelaçada), com resolução típica de 320×240 pixels de luminância (a crominância é sub-amostrada na razão de 1:4); a seguir é subdividida em blocos. Para cada bloco é feita detecção de movimento, gerando *vetores de movimento* que serão transmitidos para o receptor. Os mesmos vetores são fornecidos ao preditor / compensador de movimento, que produz um bloco reconstruído como estimativa do bloco a ser codificado. A diferença entre a imagem original e o resultado da predição de movimento é o *erro de predição*, o qual apresentará usualmente baixa correlação com a imagem original, além de energia (amplitude) reduzida.

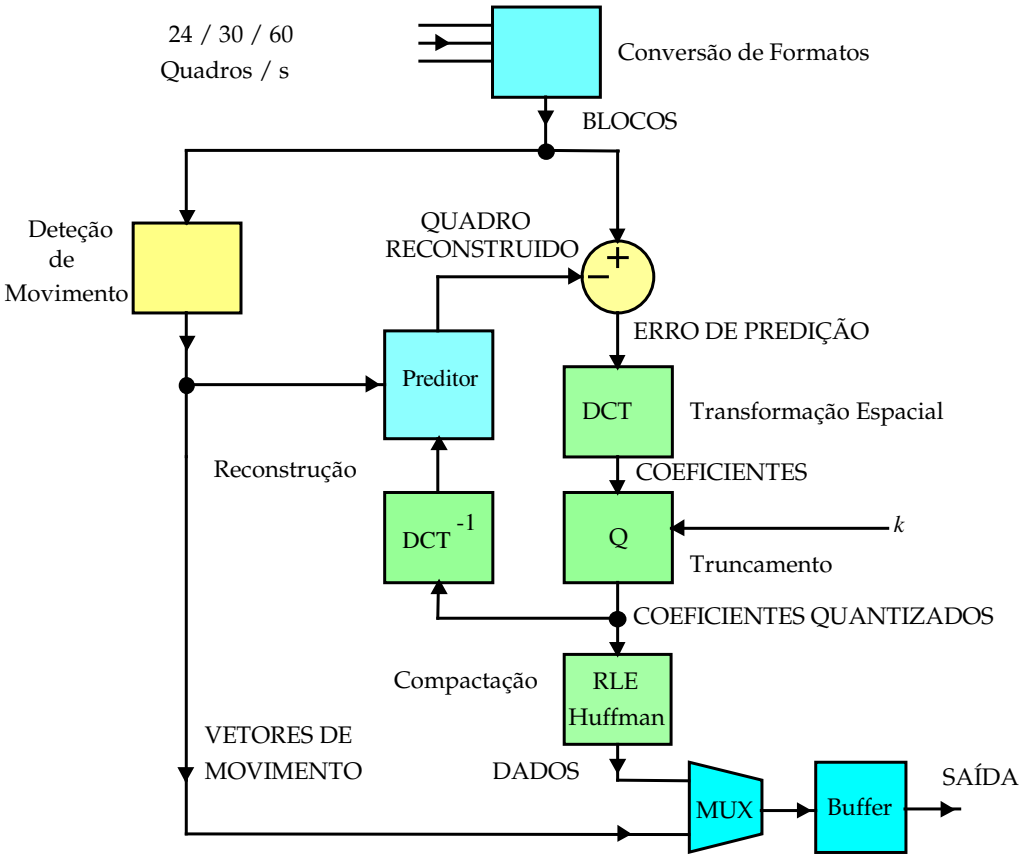


Fig. 3.1.1 - Codificador MPEG

Sobre o erro de predição resultante é aplicada uma transformada DCT 8x8, seguida de quantização dos coeficientes. Finalmente, é feita uma compactação dos dados resultantes, através de RLE e codificação Huffman com códigos de comprimento variável.

Para um bloco de entrada 8 x 8, $f(x,y)$, o resultado da DCT é um bloco de 8 x 8 coeficientes:

$$F(u,v) = \frac{1}{4} C(u) C(v) \left(\sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y) \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16} \right)$$

$$\begin{aligned} &onde \quad C(u),\ C(v) = 1/\sqrt{2} \quad para \quad u,v = 0 \\ &\qquad\quad C(u),\ C(v) = 1 \qquad\quad para \quad u,v \neq 0 \end{aligned}$$

A função inversa (DCT^{-1}) é dada por:

$$f(x,y) = \frac{1}{4}\left(\sum_{x=0}^{7}\sum_{y=0}^{7} C(u)C(v)F(u,v)\cos\frac{(2x+1)u\pi}{16}\cos\frac{(2y+1)v\pi}{16} \right)$$

A transformada DCT concentra a energia do sinal em componentes de freqüência espacial; a informação assim processada correlaciona-se melhor com a *visibilidade* do erro de predição. Após a quantização dos coeficientes, serão descartados elementos de menor importância quanto à percepção visual.

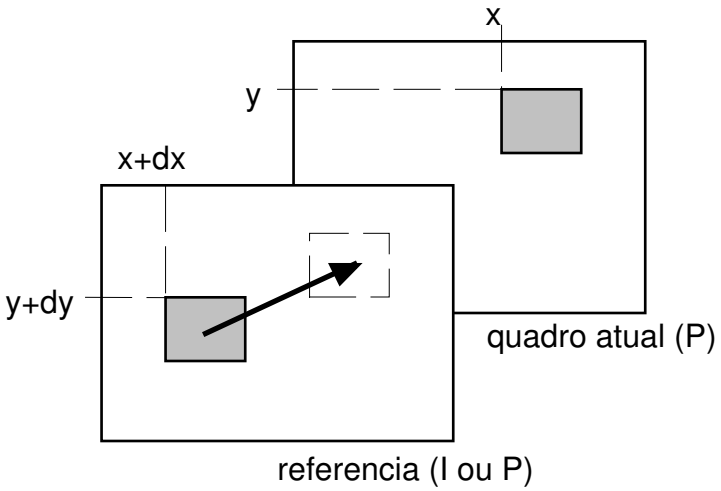


Fig. 3.1.2 - Predição de Movimento Progressiva

A etapa de quantização é responsável pelas perdas do processo; atuando no fator de escala do quantizador (**Q**) pode-se controlar a taxa de compressão média e a qualidade final da imagem. No entanto, a taxa efetiva de compressão é indeterminada *a priori*, pois depende fortemente do conteúdo da imagem.

O padrão MPEG-1 adota 3 tipos de imagens, classificadas conforme o processo usado na compensação de movimento:

Imagens I (Independentes, ou *Intra-frame*): são codificadas sem predição de movimento, ou seja, apenas por DCT, quantização e compactação. São usadas no preditor como imagens de referência para compensação de movimento de quadros futuros. Apresentam menor taxa de compressão média, mas são necessárias quando há cortes de cenas, ou para evitar propagação de erros de transmissão;

Imagens P (com predição *Progressiva* de movimento - fig. 3.1.2): são reconstruídas através de compensação de movimento, baseando-se em imagens de referência anteriormente codificadas, que podem ser imagens tipo **I** ou tipo **P**. Apresentam taxa de compressão elevada;

Imagens B (com predição *Bidirecional*): o preditor baseia-se em duas imagens de referência (anterior e posterior, tipo **I** ou **P**); admitem até dois pares de vetores de movimento (progressivos e regressivos) para cada bloco, sendo que neste caso a estimativa adotada é a média das estimativas individuais. Apresentam a maior taxa de compressão dentre os 3 tipos de imagem.

Uma seqüência de imagens de vídeo é subdividida em **Grupos de Imagens** (GOP's - *Groups of Pictures*) (fig. 3.1.3). Um GOP (que pode conter imagens tipo **I**, **P** e/ou **B** em várias proporções) é *fechado* se as predições de movimento das suas imagens são efetuadas sem necessitar de quadros de referência externos ao Grupo; desta forma, seqüências de vídeo podem ser editadas tomando-se como pontos de corte os inícios destes GOP's.

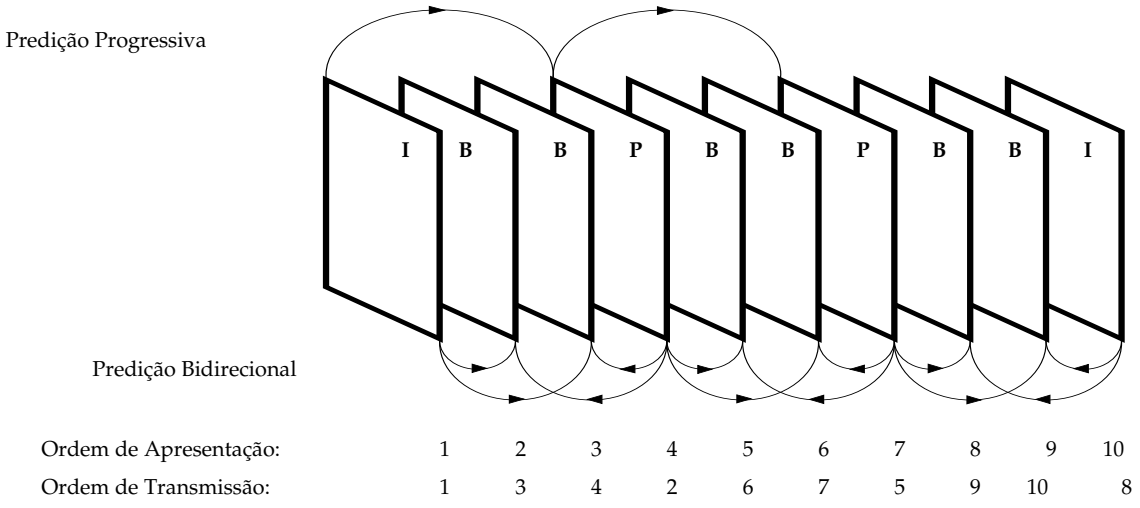


Fig. 3.1.3 - Estrutura de um Grupo de Imagens MPEG

No caso de transmissão simplex (radiodifusão), o tamanho do GOP influencia o tempo de aquisição de uma seqüência; a escolha da proporção de imagens **I**, **B** e **P** é um compromisso entre taxa de bits disponível, qualidade final da imagem e

tempo de aquisição e recuperação de erros de transmissão. Normalmente, adota-se uma imagem **I** a cada 15 quadros (1/2 segundo).

O elemento básico de predição de movimento é o ***macrobloco***, que consiste de um conjunto de 4 blocos de 8×8 amostras de luminância (Componente **Y**), e 2 blocos de crominância correspondentes (Componentes **U** e **V**), conforme mostrado na fig. 3.1.4. As componentes de crominância são sub-amostradas por um fator de 1/2 na vertical e na horizontal, de modo que cada bloco de crominância cobre a mesma área que os 4 blocos de luminância. Será então determinado um conjunto de vetores de movimento para cada macrobloco, afetando simultaneamente os 6 blocos correspondentes.

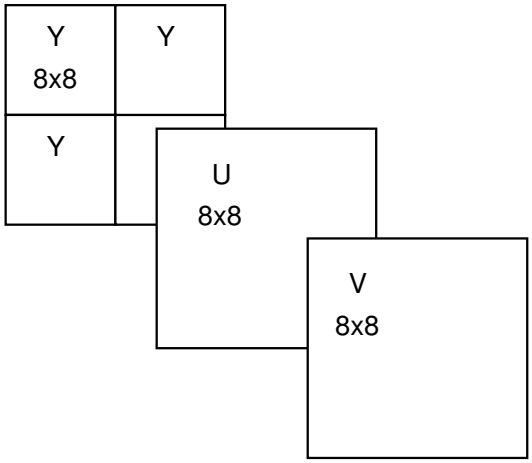


Fig. 3.1.4 - Estrutura do Macrobloco MPEG

3.2 Implementação do Algoritmo de Compressão com Preditor de Movimento

As características de captação e as condições de visualização de imagens de angiografia são diferentes das usualmente encontradas nas aplicações usuais do padrão MPEG (imagens de vídeo, TV, vídeo-conferência, multimídia). Em particular, podemos considerar os seguintes aspectos nas aplicações médicas:

1. As imagens capturadas são monocromáticas;
2. Não há entrelaçamento (a varredura é progressiva);
3. Os objetos são geralmente de baixo contraste e baixa definição;
4. Não há cortes de cenas;

5. Não há erros de transmissão;
6. Não há edição (montagem de seqüências) da imagem;
7. As imagens podem ser visualizadas individualmente (congeladas).

Estes aspectos nos levaram a implementar um algoritmo de compressão que, embora obedeça às linhas gerais do padrão MPEG-1, diferencia-se deste nos seguintes aspectos:

1. **Macrobloco**: No padrão MPEG, a predição de movimento é feita em unidades de 16x16 pixels, enquanto que na presente implementação é feita em blocos de 8x8 pixels. Este procedimento limita a extensão dos erros de previsão de movimento, diminuindo o efeito de "blocagem" da imagem. (O termo "blocagem" refere-se à visibilidade dos contornos dos blocos nos quais a imagem é subdividida, devida a erros de reconstrução.)
2. **Grupo de Imagens**: não havendo erros de transmissão, podemos dizer que não ocorre propagação de erros. Nesse caso podemos prescindir das imagens tipo **I**. Utilizamos apenas predição progressiva (Imagens **P**), baseada na imagem **P** anterior reconstruída. Apenas a primeira imagem da seqüência será tipo **I**.
3. **"*By-pass*" da DCT**: no caso de ocorrer *overflow* no cálculo dos coeficientes da DCT, o bloco correspondente é codificado na forma original (8x8 amostras de luminância). Este procedimento minimiza a ocorrência de artefatos na proximidade de detalhes de alto contraste na imagem.
4. **Detecção de Movimento**: considerando que as imagens de cineangiografia podem apresentar oscilações freqüentes de luminosidade (causadas por instabilidades na emissão de raios-X e pela atuação de controle automático de ganho, ativado pela luminosidade média, na captação da imagem), utilizamos um processo de medida de semelhança entre blocos que despreza o valor médio da luminância.

Além disso, o processo implementado de busca hierárquica para detecção de movimento diferencia-se dos métodos geralmente utilizados na codificação MPEG-1 em aplicações usuais.

As seqüências de imagens a serem processadas devem ser fornecidas na forma de arquivos independentes, em formato "Bitmap", numerados seqüencialmente. O processamento de uma seqüência de imagens pelo algoritmo de compressão implementado efetua-se em 3 etapas consecutivas:

1. Compensação de movimento, DCT e Quantização;
2. Análise estatística e montagem do Dicionário de símbolos;
3. Codificação estatística dos coeficientes.

3.2.1 Etapa 1: Detecção/Compensação de Movimento, DCT e Quantização:

Para executar esta etapa de processamento foi desenvolvido um programa (GMPEG.EXE), cuja estrutura obedece ao diagrama de blocos da fig. 3.2.1.

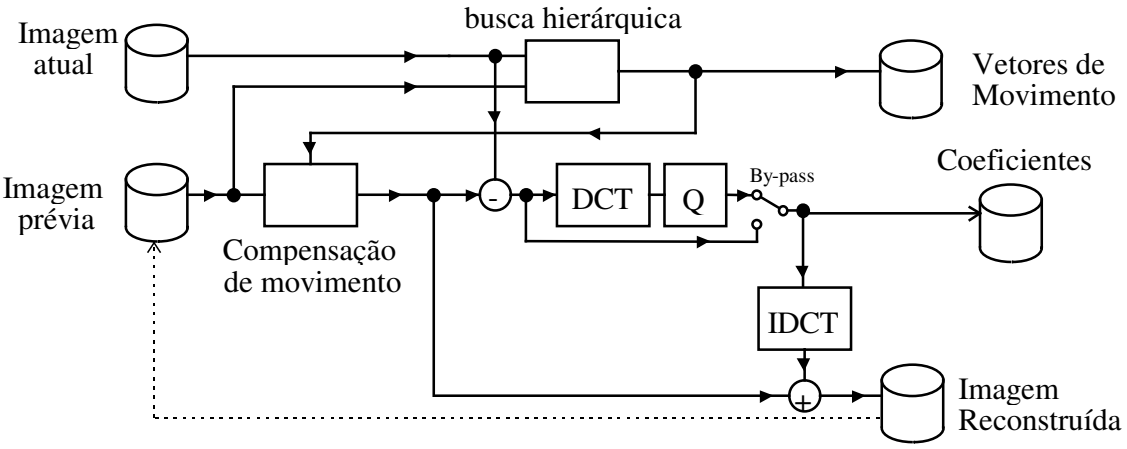


Fig. 3.2.1 - Diagrama do Algoritmo de Compressão Implementado (Etapa 1)

Na forma implementada, são gerados dois arquivos independentes de saída; um contendo os vetores de movimento, outro os coeficientes DCT quantizados, além de ser gerado um arquivo com a imagem reconstruída, o qual será usado como imagem de referência para compressão da próxima imagem da seqüência.

3.2.1.1 Algoritmo para Detecção de Movimento com Busca Hierárquica

Para a descrição a seguir, adotamos as seguintes denominações:

> **imagem atual:** é o fotograma da seqüência que está sendo codificado;
>
> **imagem prévia:** é a imagem reconstruída correspondente ao fotograma anterior, após sofrer o processo completo de compressão e descompressão;
>
> **bloco:** é um conjunto de 8×8 pixels da imagem atual.

Para cada bloco, procuramos dentro de uma região da imagem prévia (janela de busca) qual a posição na qual ocorre maior semelhança, pixel a pixel, entre o bloco da imagem atual e o respectivo trecho na imagem prévia. O deslocamento relativo entre as duas imagens, nas componentes horizontal e vertical, é denominado **vetor de movimento**.

Normalmente, a semelhança entre o bloco e a imagem prévia é determinada buscando-se o valor mínimo da somatória dos módulos das diferenças de intensidade, pixel a pixel. Para o caso de um bloco 8x8:

$$dif(x, y) = \sum_{i=0}^{7} \sum_{j=0}^{7} \left| f_a(i, j) - f_p(x + i, y + j) \right|$$

Podemos usar também a medida usual denominada DMA (Distorção Média Absoluta):

$$DMA(x, y) = \frac{1}{64} \sum_{i=0}^{7} \sum_{j=0}^{7} \left| f_a(i, j) - f_p(x + i, y + j) \right|$$

onde $f_a(i, j)$ é o bloco (8x8) da imagem atual e $f_p(x, y)$ é a imagem prévia. Os valores de (x,y) onde $dif(x,y)$ é mínimo são adotados como o **vetor de movimento** do bloco em questão.

A operação de busca acima, quando executada sobre todos os blocos da imagem ($60 \times 64 = 3840$ blocos), e considerando uma janela de busca de +/- 64 pixels no sentido vertical e horizontal, demanda um tempo de processamento

muito elevado : 3840 blocos × (128×128) posições × 64 pixels = 4×10^9 operações por imagem.

Para reduzir a complexidade computacional do algoritmo, decidimos fazer a busca utilizando uma imagem intermediária com resolução reduzida (busca hierárquica). Este processo está esquematizado na fig. 3.2.2.

Numa primeira fase, a partir da imagem prévia $f_p(x, y)$ de 480×512 pixels, geramos uma imagem com meia-resolução, $f_{ph}(u, v)$ com 240×256 pixels, onde cada pixel é calculado como a soma de 4 pixels adjacentes da imagem prévia. Para cada bloco a ser codificado da imagem atual, calculamos ainda um sub-bloco $f_{ah}(i, j)$ de 4×4 pixels, obtidos da mesma forma. Assim, a busca de movimento reduz-se para uma janela de +/- 32 pixels, executada sobre blocos de 4×4 pixels.

Na segunda fase, após obtido o vetor de movimento preliminar (u,v) fazemos uma nova busca, agora com o bloco e a imagem prévia nas resoluções originais, porém apenas dentro de uma janela reduzida de +/- 4 pixels, centrada nas posições correspondentes a (u,v). Deste modo, reduzimos a complexidade para 3840 × (64×64×16 + 8×8×64) = 267×10^6 operações por imagem.

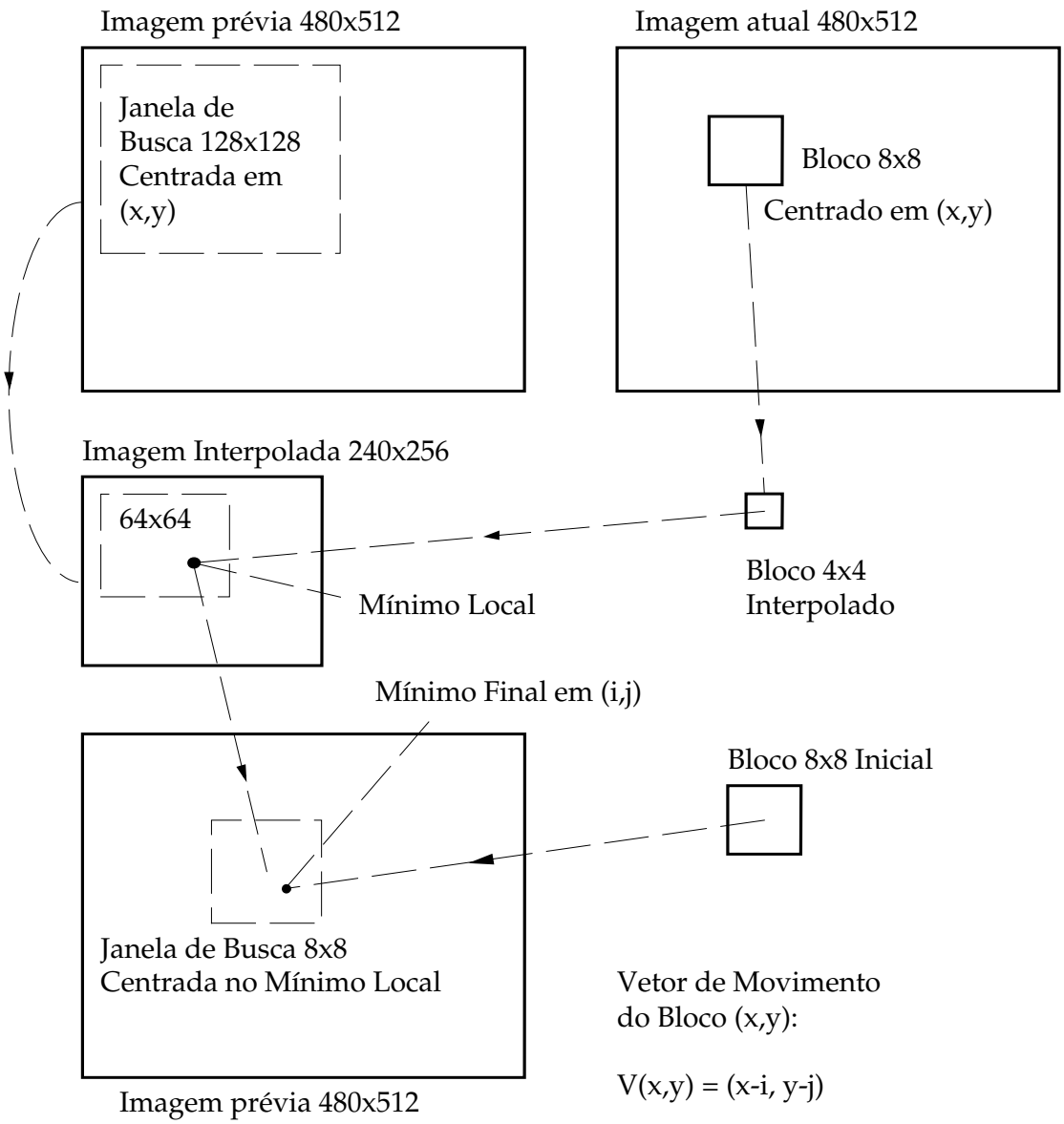


Fig. 3.2.2 - Diagrama do processo de Busca Hierárquica

A figura 3.2.3 (vide também Capítulo 8) apresenta uma situação típica para o erro de predição de movimento (exibido com contraste ampliado 16x para melhor visualização). O histograma correspondente de distribuição de amplitudes está apresentado na fig. 3.2.4. Podemos observar que a energia média do erro de predição é bastante reduzida (desvio padrão igual a 4,02 níveis de quantização para esta imagem em particular).

Para o caso da primeira imagem da seqüência, adotamos como "imagem prévia" um valor constante igual a 128 (cinza médio).

A imagem reconstruída é obtida a partir da imagem prévia, dos vetores de movimento e da aplicação da transformada DCT inversa sobre os coeficientes quantizados. Este processo é idêntico ao que seria realizado na descompressão da seqüência de imagens.

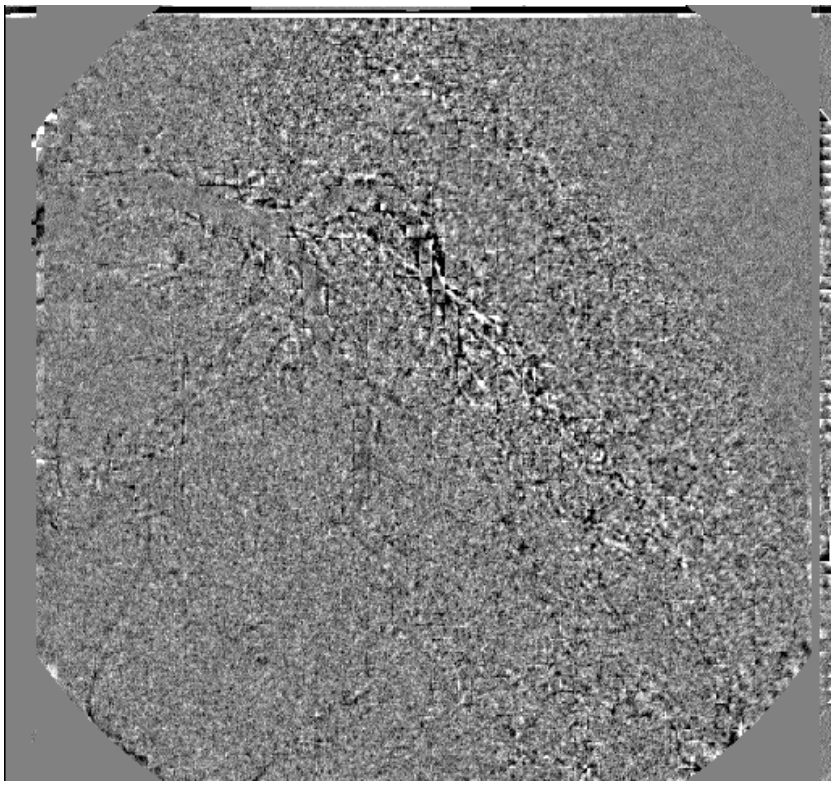


Fig. 3.2.3 - Erro de Predição de Movimento (Imagem No. 30)

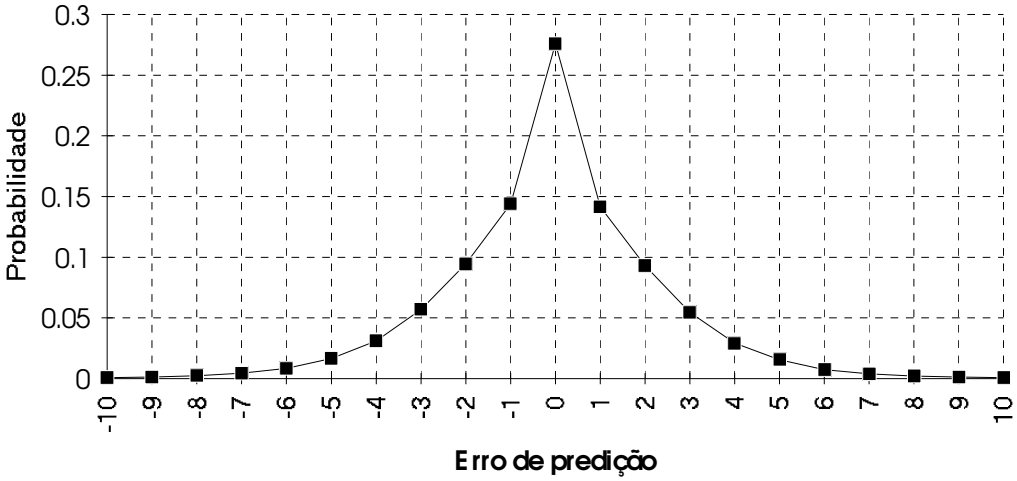


Fig. 3.2.4 - Histograma do Erro de Predição de Movimento (ref. XA030.BMP)

3.2.1.2 Algoritmo de Busca por Máxima Semelhança

No processo descrito acima, a detecção de movimento privilegia blocos, na imagem de referência, que possuem a mesma luminosidade média do que o bloco em análise. Este algoritmo é satisfatório em aplicações na área de Televisão, onde a iluminação de cena é constante, e objetos em movimento mantêm a mesma luminância. No caso em que ocorrem variações de intensidade entre 2 quadros consecutivos, o algoritmo irá determinar vetores de movimento espúrios, que não correspondem a movimentos reais dos objetos em relação à imagem. O codificador MPEG não possui mecanismos para acompanhar mudanças de luminosidade dos objetos (SYMES, 1998, p.241; WHITAKER, 1998, p.163).

A distorção média absoluta será minimizada; no entanto, como os blocos de referência obtidos não correspondem ao mesmo detalhe do objeto, o erro de predição de movimento residual terá amplitude considerável, resultando no surgimento de vários coeficientes não nulos após a transformada DCT.

Este fenômeno ocorre, em imagens de televisão codificadas por MPEG, quando é usado o recurso de "fade out" para corte gradual de uma cena. Neste caso, chega a ser claramente visível um aumento considerável do ruído de reconstrução.

Idealmente, a busca do "melhor" bloco para compensação de movimento envolveria a codificação por DCT e quantização do erro de predição para cada valor de deslocamento (x,y), sendo escolhido como vetor de movimento aquele que resultasse em um erro de reconstrução tolerável com a menor quantidade de coeficientes não nulos (MITCHELL, 1996, p.241). No entanto, este processo, além de ser extremamente dispendioso em termos de processamento, não garante que este bloco será codificado com a menor quantidade de bits, uma vez que as estatísticas do conjunto de imagens (e o conseqüente dicionário de codificação Huffman) só serão obtidas no final do processo.

Uma vez que a luminância média do bloco (correspondente ao coeficiente *F(0,0)* da transformada DCT) é codificada e quantizada de forma independente dos demais coeficientes, sugerimos então um processo de busca no qual esta luminosidade média dos blocos é cancelada, sendo então feita a determinação da Distorção Média Absoluta entre os blocos resultantes. Desta maneira, o erro de predição será separado em duas componentes: um erro de *intensidade média* do bloco (que será codificado com alta precisão) e um erro de *forma* do bloco, constituído das componentes de freqüência espacial diferente de zero (que será codificado com quantização variável).

Seja $f_a(i,j)$ um bloco (8x8) da imagem atual, a ser codificado; e seja $f_p(x,y)$ a imagem prévia. Determinamos inicialmente, para este bloco, a somatória de todos os seus elementos:

$$sum(f_a) = \sum_{i=0}^{7}\sum_{j=0}^{7} f_a(i,j)$$

Para um dado vetor de movimento **(*x*,*y*)** a ser pesquisado, determinamos agora a somatória de um bloco 8x8 da imagem prévia, correspondente ao vetor de movimento:

$$sum(f_p) = \sum_{i=0}^{7}\sum_{j=0}^{7} f_p(x+i, y+j)$$

A seguir, é calculada a diferença absoluta entre os elementos dos blocos, descontando para cada elemento os valores médios dos blocos:

$$dif(x,y) = \sum_{i=0}^{7}\sum_{j=0}^{7} \left| f_a(i,j) - \frac{sum(f_a)}{64} - f_p(x+i, y+j) + \frac{sum(f_p)}{64} \right|$$

Esta operação pode ser simplificada uma vez que os valores de $\boldsymbol{sum(f_p)}$ podem ser calculados para a imagem prévia inteira, uma única vez, no início do processo de codificação de uma nova imagem da seqüência:

$$sum(f_p(x,y)) = \sum_{i=0}^{7}\sum_{j=0}^{7} f_p(x+i, y+j)$$

Na implementação em linguagem "C" deste algoritmo (programa GMPEG.EXE), o tempo de execução é de cerca de 70 segundos por imagem, em um Pentium 266.

3.2.1.3 Transformada Discreta de Cosenos (DCT)

O erro resultante da compensação de movimento é codificado por uma Transformada Discreta de Cosenos. Nesta implementação, a DCT é implementada por um produto matricial da forma

$$\mathbf{DCT} = \mathbf{C} \times \mathbf{B} \times \mathbf{C}^{\mathrm{T}}$$

onde **B** é um bloco de 8 × 8 pixels da imagem original e **C** é uma matriz de transformação, definida por:

$$C_{i,j} = \begin{cases} \dfrac{1}{2\sqrt{2}} & \text{se } i = 0 \\ \dfrac{1}{2}\cos\left(\dfrac{(2j+1)i\pi}{16}\right) & \text{se } 0 \le i \le 7 \end{cases}$$

$\mathbf{C_{8x8}}$=

.354	.354	.354	.354	.354	.354	.354	.354
.490	.416	.278	.098	-.098	-.278	-.416	-.490
.462	.191	-.191	-.462	-.462	-.191	.191	.462
.416	-.098	-.490	-.278	.278	.490	.098	-.416
.354	-.354	-.354	.354	.354	-.354	-.354	.354
.278	-.490	.098	.416	-.416	-.098	.490	-.278
.191	-.462	.462	-.191	-.191	.462	-.462	.191
.098	-.278	.416	-.490	.490	-.416	.278	-.098

3.2.1.4 Quantização dos Coeficientes da DCT

Os coeficientes obtidos pela DCT são quantizados através da divisão por um *Fator de Quantização*, com arredondamento para o inteiro mais próximo. Pode ser usada ainda uma função não linear, como apresentada na fig. 3.2.5,

incorporando uma zona morta para os valores menores que +/-2. Este quantizador não linear é análogo ao disponível (opcional) na codificação padrão MPEG-2 (HASKELL, 1997, p.168).

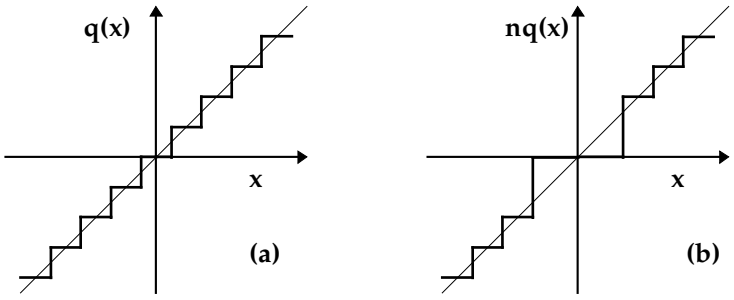


Fig. 3.2.5 - Quantizador Linear (a) e Não-linear (b)

Após a quantização, caso algum coeficiente ultrapasse o valor representável com 8 bits (-128 a +127), a codificação DCT será descartada e o bloco será armazenado na forma original ("*by-pass*" da DCT).

3.2.1.5 Reordenação dos Coeficientes Quantizados

É usada a reordenação padrão em "zig-zag" do processo MPEG para organizar os coeficientes quantizados (fig. 3.2.6). Para imagens típicas, com concentração de componentes de baixa freqüência espacial, espera-se que após este processo os coeficientes não nulos estejam concentrados no início da seqüência reordenada.

3.2.2 Etapa 2: Análise Estatística e Montagem do Dicionário dos Símbolos

Nesta etapa são analisados os arquivos de coeficientes DCT, sendo feita a contagem cumulativa de todos os símbolos possíveis (combinações de *run-lenght* e magnitude dos coeficientes).

Para esta análise foi desenvolvido o programa ANALISE.EXE. Nesta implementação, os símbolos são ordenados em uma matriz de 64 x 128 elementos, correspondentes às quantidades seguidas de zeros e aos valores possíveis das magnitudes dos coeficientes. Estamos supondo que coeficientes

positivos e negativos de mesma magnitude terão probabilidades de ocorrência iguais.

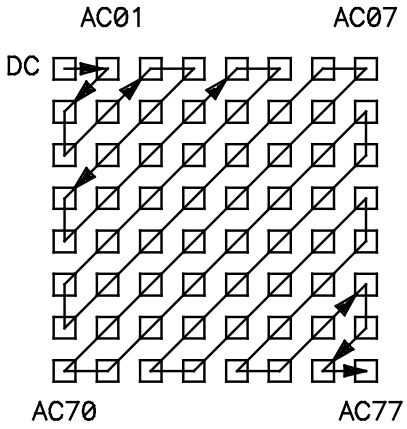


Fig. 3.2.6 - Seqüência de Codificação dos Coeficientes da DCT

Após contabilizados todos os símbolos presentes em todos os arquivos de coeficientes quantizados, é gerado um arquivo contendo os símbolos (*run-lenght* e valor), bem como sua probabilidade de ocorrência, desde que esta seja maior que zero. Estes símbolos são agrupados em uma matriz , contendo os valores de *run-lenght* e de magnitude do coeficiente, além da contagem de ocorrências dos mesmos.

3.2.2.1 Código de Huffman Modificado

Para reduzir o "*overhead*" resultante da possibilidade de ocorrência de longas seqüências de zeros, gerando códigos com baixa probabilidade de ocorrência, símbolos com *run-lenght* maior que 16 são desmembrados através da introdução de um código (16,0) que corresponde a uma seqüência de 16 zeros. Desta forma, o número máximo de códigos é reduzido para 16 x 128 = 2048 combinações de *run-lenght* e magnitude.

A seguir, é feita a atribuição de códigos de comprimento variável utilizando o algoritmo de Huffman.

3.2.2.2 Resultados da Análise Estatística

Como saída, o programa ANALISE.EXE apresenta um arquivo de dicionário, contendo os valores de run-lenght, magnitude e o respectivo código binário de comprimento variável. O sinal (polaridade) do coeficiente a ser codificado será anexado ao final do código correspondente no dicionário.

Como medida da eficiência que a compactação pode alcançar, o programa calcula ainda a entropia dos códigos, pela expressão

$$E(X) = -\sum_{i=1}^{n} P(s_i) \log_2 P(s_i)$$

onde $P(s_i)$ é a probabilidade de ocorrência do símbolo s_i e n é o número total de códigos gerados.

3.2.3 Etapa 3: Codificação Estatística dos Coeficientes

Nesta etapa, implementada pelo programa CODE.EXE, os arquivos de coeficientes quantizados e de vetores de movimento, gerados na Etapa 1, são combinados em um único arquivo de saída, sendo usada a codificação Huffman conforme o dicionário elaborado na Etapa 2. Os vetores de movimento são compactados, ocupando 6 bits cada um, e os coeficientes correspondentes ao valor médio de cada bloco são armazenados com 8 bits.

3.2.4 Análise do Erro de Reconstrução

Uma etapa adicional do processo, necessária para este trabalho, consiste da análise do erro de reconstrução, que é obtido efetuando-se a subtração entre a imagem original e a imagem reconstruída, gerada na Etapa 1. O programa ERROR.EXE, desenvolvido com esta finalidade, produz arquivos de saída no formato Bitmap, nos quais a magnitude do erro de reconstrução é ampliada 16 vezes, de modo a permitir melhor visualização comparativa. Um erro igual a zero é representado como um nível de cinza (valor de luminância igual a 128).

Este programa gera ainda um arquivo contendo o erro quadrático médio de reconstrução de cada imagem da seqüência, calculado na região útil da mesma (uma região de 440 linhas x 492 colunas centralizada na imagem).

4. Análise dos Resultados Práticos Obtidos

Serão apresentados aqui os resultados obtidos a partir da implementação descrita no capítulo anterior. Nesse sentido, o sistema de compressão de imagens desenvolvido foi aplicado no processamento de uma seqüência de imagens médicas, fornecida pelo InCor.

A seqüência estudada nos exemplos apresentados neste trabalho consiste de 38 fotogramas extraídos de uma seqüência de cineangiografia, digitalizados na forma de arquivos individuais no formato Bitmap, com resolução de 480 linhas por 512 colunas x 8 bits (seqüência de arquivos XA001.BMP a XA038.BMP).

Cada arquivo ocupa 246.839 bytes, sendo que os primeiros 1079 bytes correspondem ao "header" do formato Bitmap, contendo as dimensões da imagem e a paleta de cores utilizada. Os restantes 245.760 bytes trazem as luminosidades de cada elemento de imagem quantizadas com 8 bits.

Foram feitos ensaios utilizando tanto o método de busca por semelhança, descrito no item 3.2.1.2, como utilizando busca normal (minimizando a Distorção Média Absoluta). O efeito do Fator de Quantização (Q) sobre a taxa de compressão média e sobre o erro de reconstrução foi estudado para Q=2, Q=4, Q=8 e Q=16.

O desempenho dos processos de compressão foi avaliado pelos seguintes aspectos:

- **Taxa de compressão**: relação entre o tamanho médio dos arquivos Bitmap originais (*descontando o "header"*) e o tamanho médio dos arquivos codificados (incluindo vetores de movimento e coeficientes codificados por Huffman);
- **Erro de Reconstrução**: desvio padrão (R.M.S.) médio da diferença entre a imagem original e a imagem reconstruída, calculado pixel a pixel dentro de uma região de 440 linhas x 492 colunas, centralizada na imagem, medido em níveis de quantização, *não ponderado*.

A ponderação do erro de reconstrução por um filtro com resposta em freqüência espacial semelhante à curva de sensibilidade da visão humana (como é feito na medida de ruído em imagens de vídeo e TV) proporcionaria valores finais menores; no entanto, como as imagens médicas podem ser visualizadas em condições de ampliação (por proximidade do observador) variáveis, a curva de ponderação aplicável seria diferente para cada caso, resultando em valores inconsistentes. Preferimos assim avaliar o erro de reconstrução pelo valor R.M.S. não ponderado.

Foi desenvolvido ainda um programa de avaliação (FSHOW.BAS), escrito em Visual Basic, que permite a exibição de seqüências de vídeo para análise subjetiva. São apresentadas as seqüências original e reconstruída, além do erro de reconstrução, com possibilidade de seleção do fator de quantização e do processo de busca utilizado.

4.1 Compressão com Busca Normal

Para estes testes foi usada uma versão do programa GMPEG.EXE na qual as rotinas de busca por máxima semelhança foram desativadas, permanecendo os demais procedimentos inalterados. Para os fatores de quantização testados, o desempenho quanto à taxa de compressão pode ser avaliado pelo gráfico da Fig. 4.1.1, na qual são apresentados os tamanhos resultantes dos arquivos comprimidos, para fator de quantização Q=2, 4, 8 e 16. Podemos observar que o tamanho do arquivo sofre um pequeno aumento na imagem 16; é onde começa a ser injetado o contraste nas coronárias, aumentando a visibilidade das mesmas. Já a Fig. 4.1.2 apresenta o erro quadrático médio obtido para cada imagem da seqüência, para os fatores de quantização usados. As primeiras imagens desta seqüência em particular são mais escuras que as outras; isso explica porque apresentam erro menor.

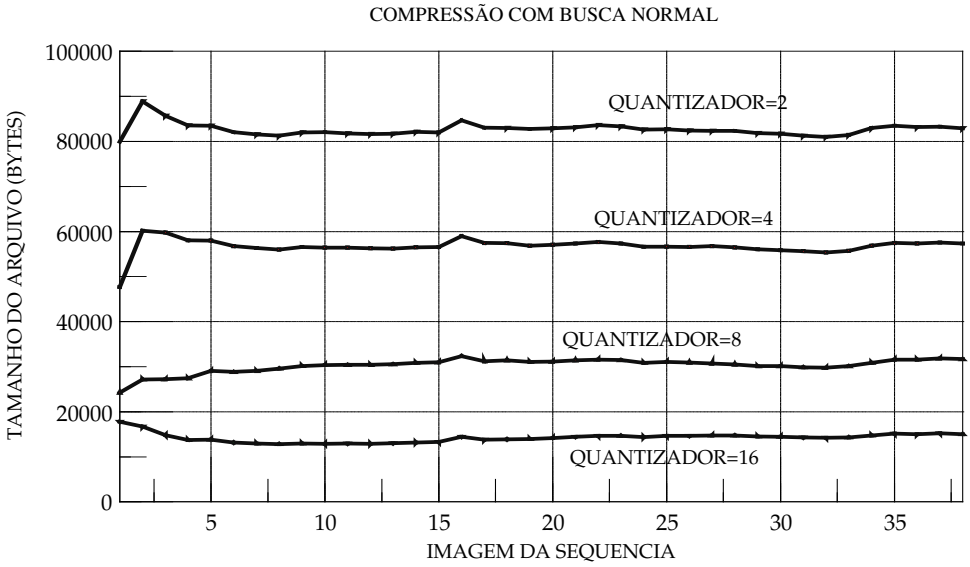


Fig. 4.1.1 - Tamanho dos Arquivos Comprimidos - Busca Normal

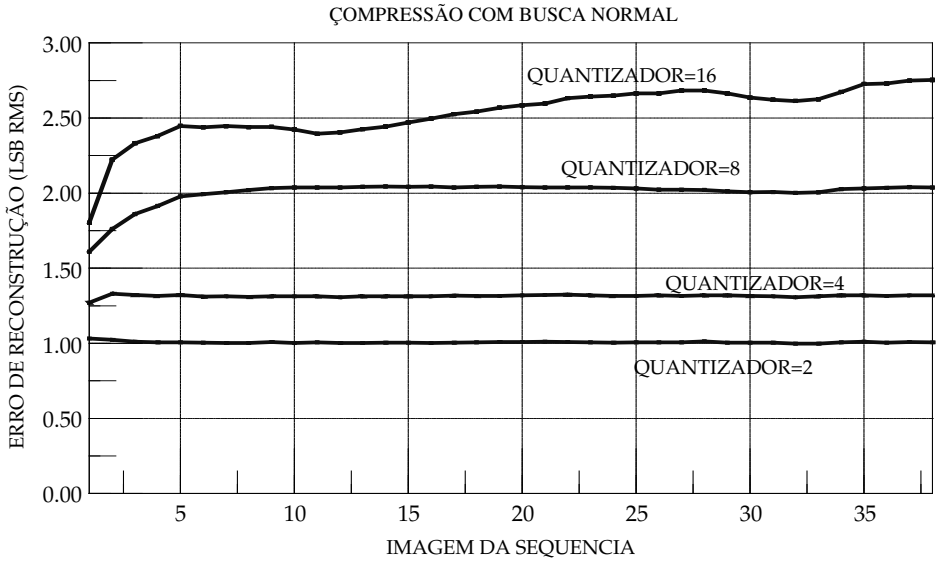
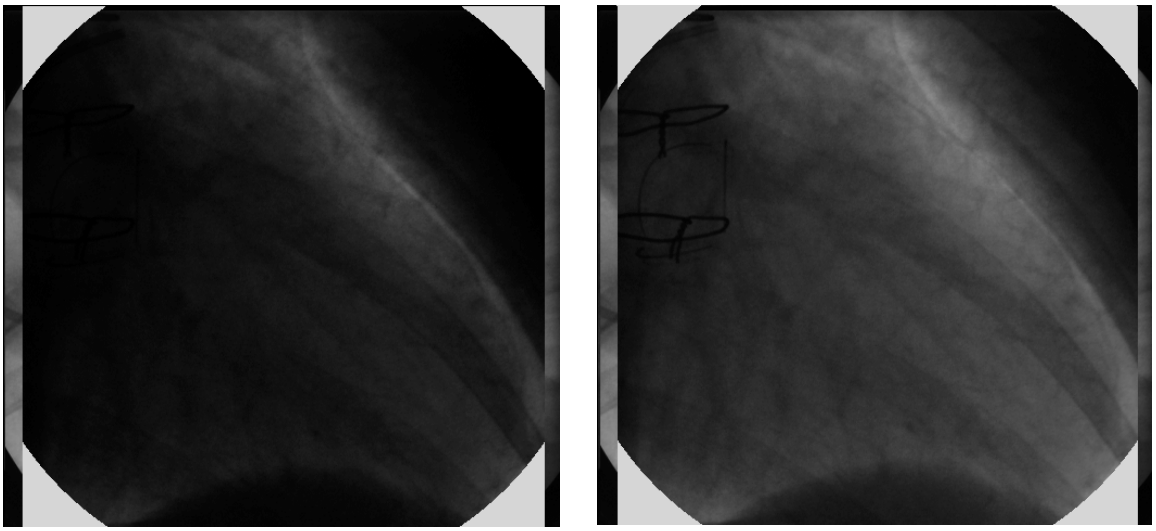


Fig. 4.1.2 - Erro de Reconstrução - Busca Normal

Nas primeiras imagens da seqüência há um aumento gradativo da luminosidade (Fig. 4.1.3), o que pode acarretar erros de predição de movimento pelo algoritmo normal de busca por mínima DMA. Para certificar-nos deste fato, examinamos o histograma de amplitudes dos vetores de movimento para a segunda imagem da seqüência (fig. 4.1.4).



XA001.BMP **XA002.BMP**

Fig. 4.1.3 - Variação de Luminosidade nas 2 primeiras imagens da seqüência

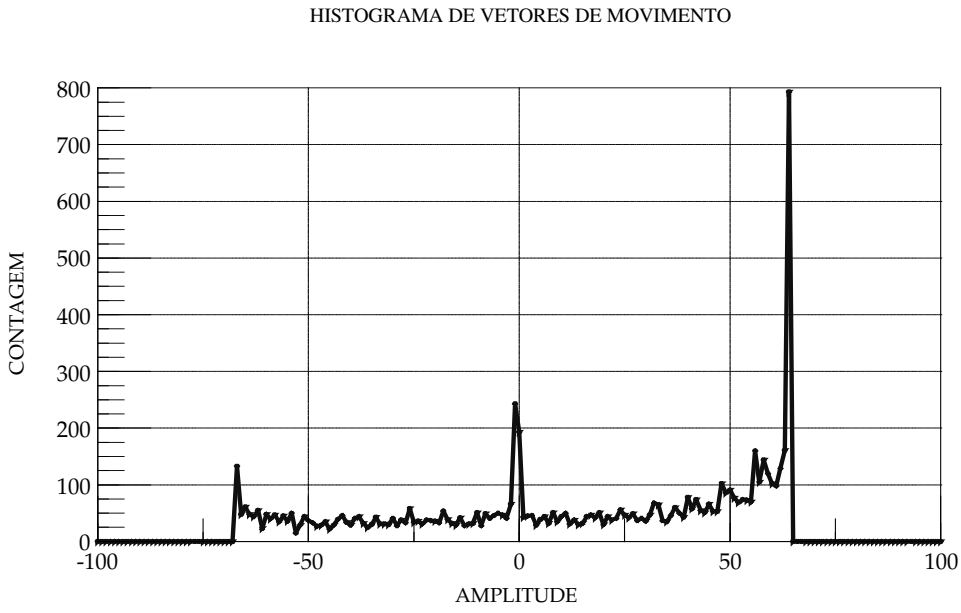


Fig. 4.1.4 - Histograma de Vetores de Movimento - Busca Normal - Imagem 2

Pelo histograma da figura 4.1.4, podemos ver que os vetores concentram-se no limite de extensão da área de busca (+64), indicando que a compensação de movimento está sendo efetuada com blocos que não correspondem às posições corretas, uma vez que não há movimento real dos objetos entre estas duas imagens.

4.2 Compressão com Busca por Máxima Semelhança

Os testes efetuados no item anterior foram repetidos usando o programa GMPEG.C, que utiliza detecção de movimento com busca por máxima semelhança, conforme descrito no item 3.2.1.2.

Para os fatores de quantização testados, o desempenho quanto à taxa de compressão pode ser avaliado pelo gráfico da Fig. 4.2.1, na qual são apresentados os tamanhos resultantes dos arquivos comprimidos, para Q=2, 4, 8 e 16. Já na Fig. 4.2.2 apresentamos o erro quadrático médio obtido para cada imagem da seqüência, para os fatores de quantização usados.

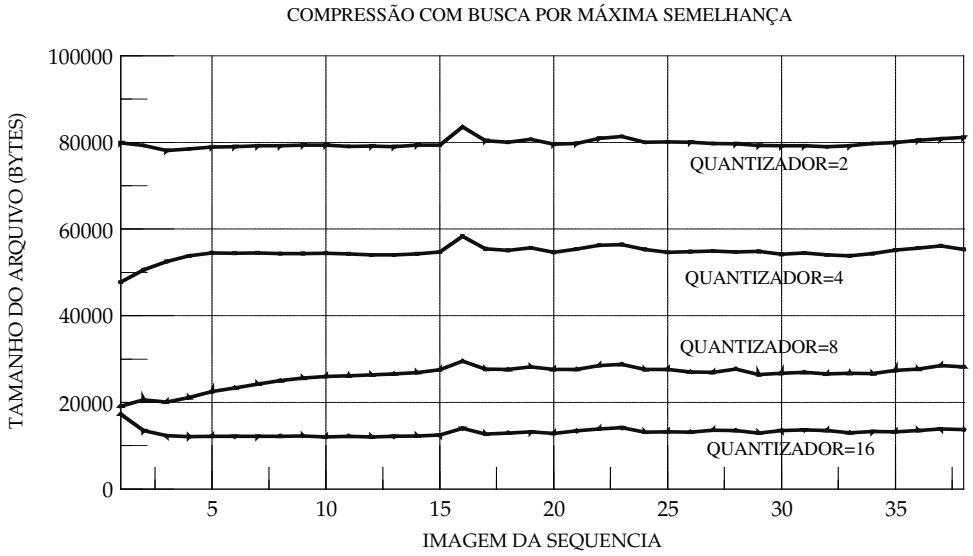


Fig. 4.2.1 - Tamanho dos Arquivos Comprimidos - Busca p/ Máxima Semelhança

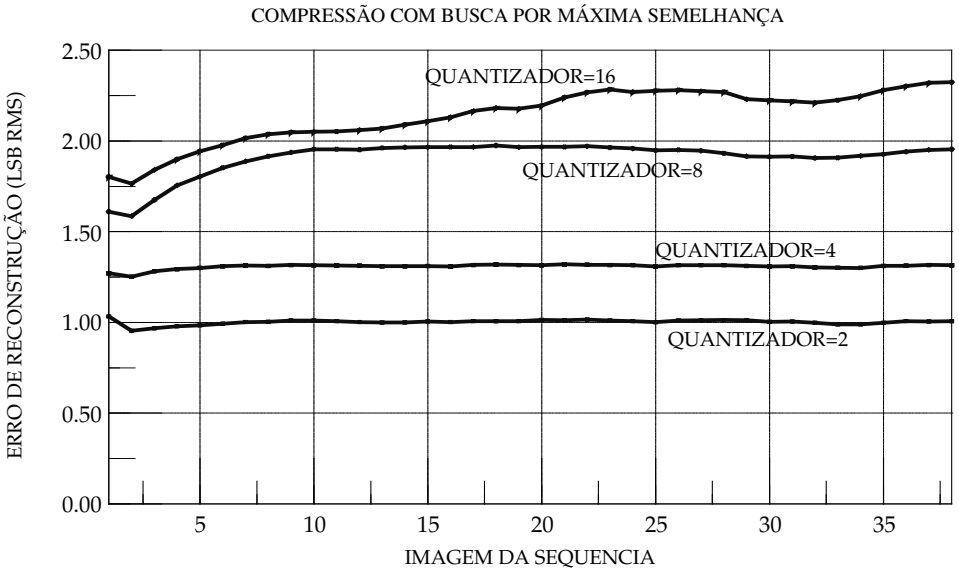


Fig. 4.2.2 - Erro de Reconstrução - Busca por Máxima Semelhança

Notamos que nesta implementação o erro de reconstrução diminui da primeira para a segunda imagem da seqüência; isto era de se esperar, uma vez que a primeira imagem, não tendo compensação de movimento, terá um erro de predição residual maior. No caso anterior (fig. 4.1.1), a segunda imagem possui tamanho maior devido ao erro na determinação dos vetores de movimento, o que acarreta um maior erro de predição.

A figura 4.2.3 apresenta o histograma dos vetores de movimento para a segunda imagem da seqüência. Pode-se observar que, utilizando a busca por máxima semelhança, os vetores agora concentram-se em torno do zero, possuindo uma distribuição semelhante às demais imagens da seqüência.

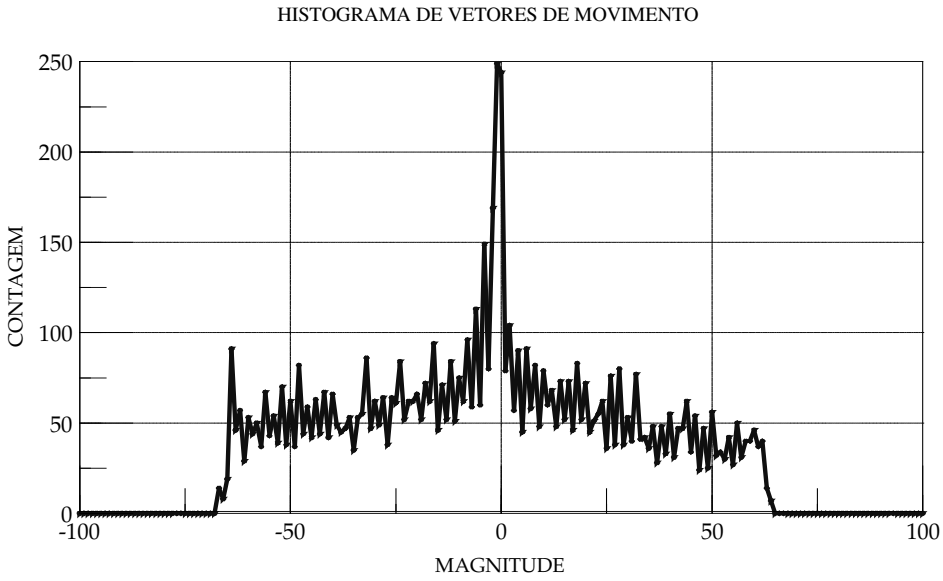


Fig. 4.2.3 - Vetores de Movimento - Busca p/ Max. Semelhança - Imagem 2

4.3 Comparação Entre os Métodos de Busca Para Compensação de Movimento

Para um dado fator de quantização, a utilização da busca por máxima semelhança traduz-se tanto por uma diminuição do tamanho final do arquivo compactado (ou aumento da taxa de compressão) quanto por uma melhora de qualidade (menor erro de reconstrução).

Nas figuras 4.3.1. e 4.3.2 temos uma comparação, quadro a quadro, dos tamanhos dos arquivos comprimidos pelos dois métodos, para fatores de quantização Q=2 e Q=8.

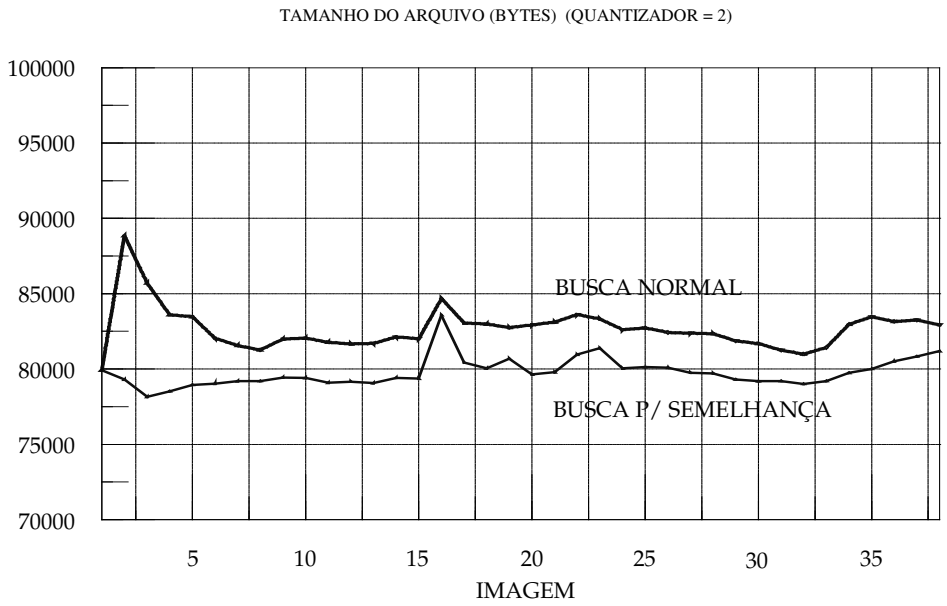


Fig. 4.3.1 - Tamanho dos Arquivos Comprimidos - Quantizador Q=2

Podemos constatar que o método de busca por máxima semelhança produz consistentemente arquivos com taxa de compressão maior. Mesmo no caso da primeira imagem da seqüência, que não utiliza predição de movimento, a redução de tamanho observada com fatores de quantização maiores (fig. 4.3.2) é conseqüência de um código de Huffman mais compacto.

As taxas de compressão médias efetivas, em função do valor do fator de quantização, são apresentadas na fig. 4.3.3. Da mesma forma, o erro de reconstrução dos dois processos é apresentado na fig. 4.3.4.

Observamos que a vantagem do método de busca por máxima semelhança acentua-se para fatores de quantização mais elevados, proporcionando portanto melhor qualidade com taxas de compressão mais elevadas.

Na Fig. 4.3.5, apresentamos o desempenho conjunto (tamanho médio do arquivo x erro de reconstrução) comparando os dois processos.

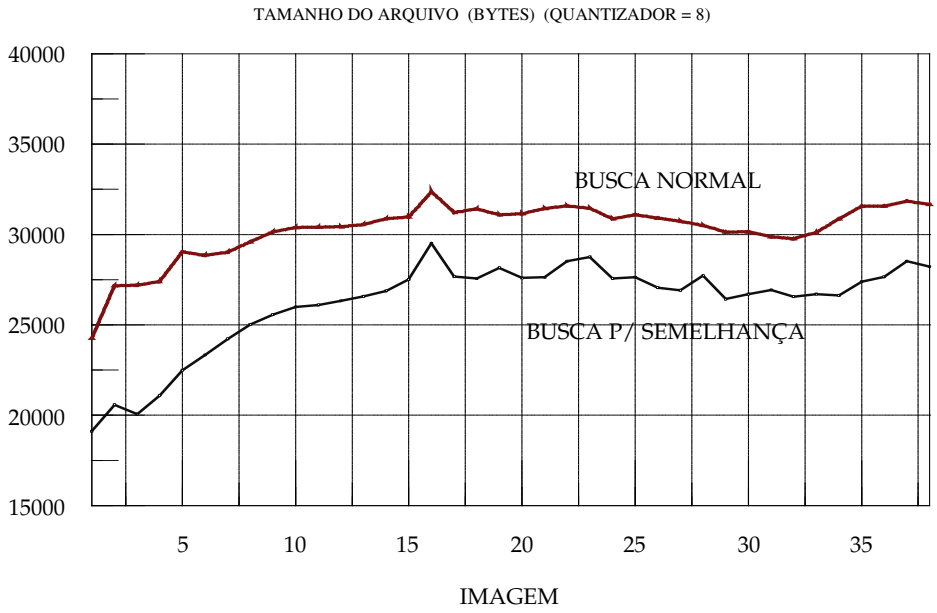


Fig. 4.3.2 - Tamanho dos Arquivos Comprimidos - Quantizador Q=8

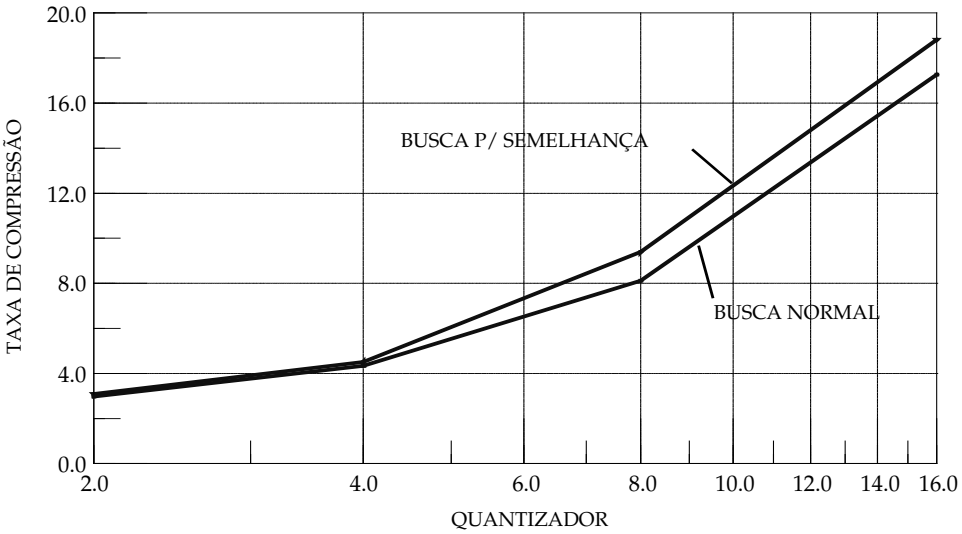


Fig. 4.3.3 - Taxas de Compressão médias em Função do Fator de Quantização

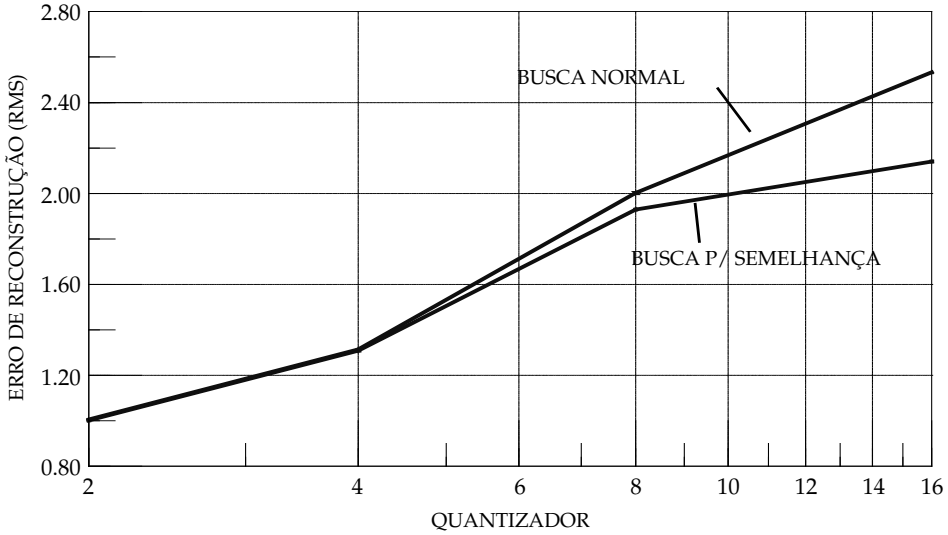


Fig. 4.3.4 - Erro de Reconstrução Médio em Função do Fator de Quantização

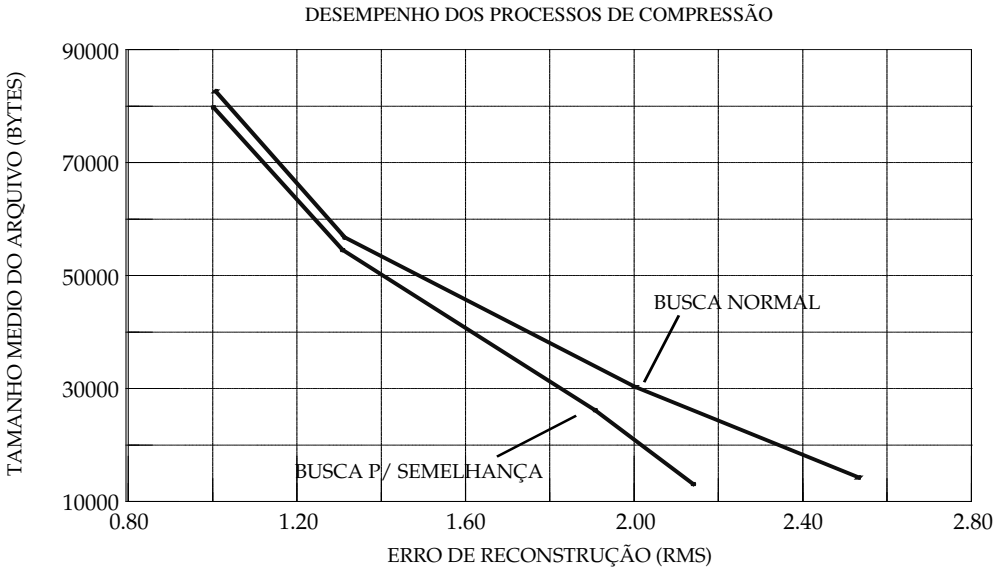


Fig. 4.3.5 - Desempenho Conjunto dos Processos de Compressão

4.4 Quantizador Não Linear

Foram feitos testes utilizando o quantizador não linear, descrito na seção 3.2.1.4, para fatores de quantização Q=1, Q=2, Q=4 e Q=8, em conjunto com o processo de busca por máxima semelhança. De um modo geral, o efeito deste quantizador equivale, em termos de taxa de compressão (fig. 4.4.1) e relação sinal/ruído (fig. 4.4.2), ao uso de um fator de quantização de duas a três vezes maior, em relação ao processamento com quantização linear.

Há um ganho discreto de desempenho conjunto, como pode ser evidenciado pela figura 4.4.3, para uma determinada faixa de fatores de quantização. No entanto, para valores extremos, o desempenho é menor que aquele proporcionado pelo quantizador linear.

Para uma taxa de compressão de 12:1, por exemplo, o método de busca por semelhança proporciona um ganho de 1,4 dB na relação Sinal/Ruído da imagem reconstruída. A utilização do quantizador não linear pode proporcionar um ganho adicional de 0,2 dB.

Analisando de outra forma, para um erro de reconstrução médio de 2.0 níveis RMS, a taxa de compressão efetiva obtida é:

	Taxa de Compressão
Predição por busca normal:	8.2 : 1
Busca por Máx. Semelhança:	11.2 : 1
Max. Sem. + Quantizador Não Linear:	14.0 : 1

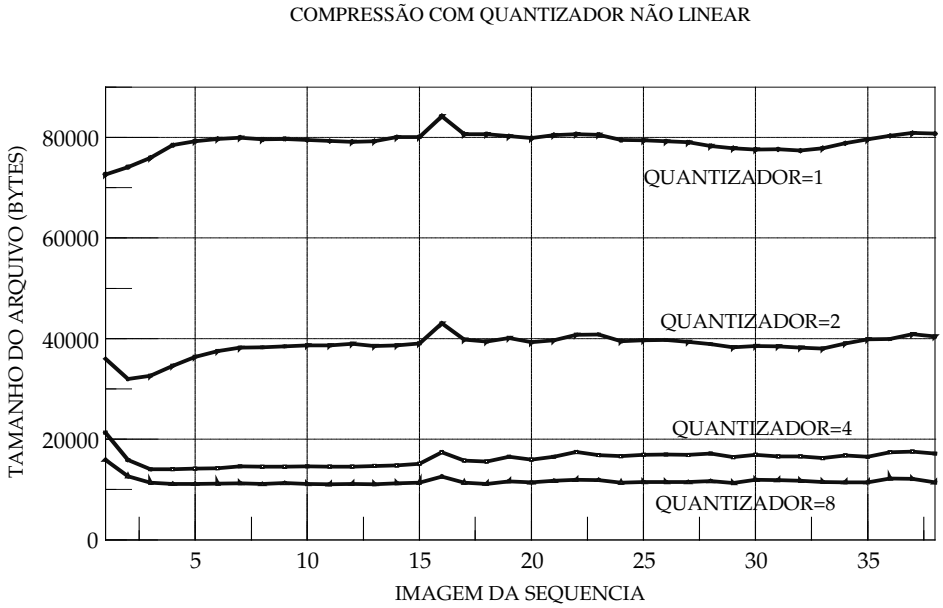


Fig. 4.4.1 - Tamanho dos Arquivos Comprimidos - Busca p/ Máxima Semelhança e Quantizador Não Linear

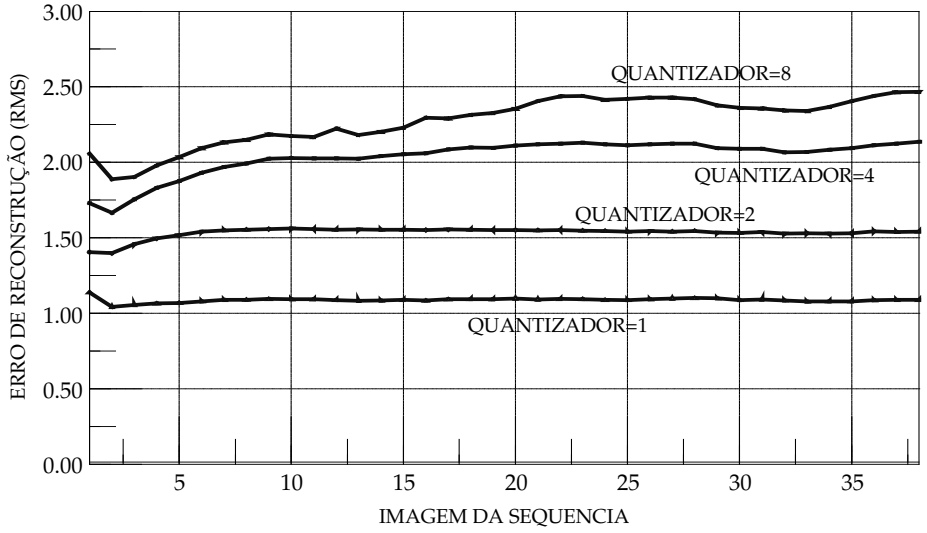


Fig. 4.4.2 - Erro de Reconstrução (R.M.S.)- Busca p/ Máxima Semelhança e Quantizador Não Linear

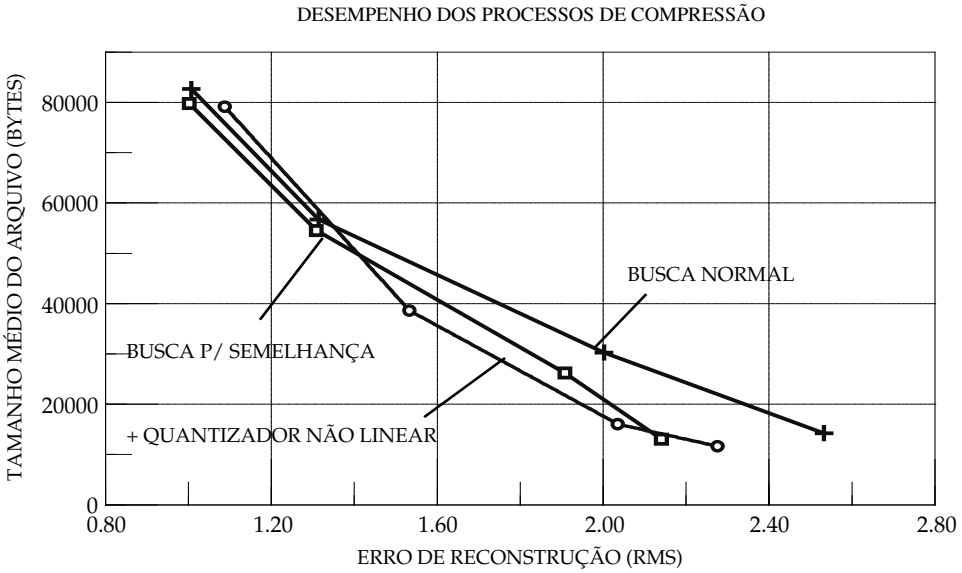


Fig. 4.4.3 - Desempenho Conjunto dos Processos de Compressão e Quantização

Modo	Parâmetro	Fator de Quantização			
		Q=2	Q=4	Q=8	Q=16
NL	Entropia (bits)	5.0630	4.9725	3.5596	
MS		3.83733	4.2353	5.0093	3.9964
N		3.9111	4.1952	4.9367	4.3752
NL	Número de Códigos	819	626	376	
MS		416	482	349	217
N		472	493	328	188
NL	Lmax (bits)	22	20	18	
MS		23	22	21	20
N		24	22	21	20

(MS) = Busca por Máxima Semelhança (N) = Busca Normal
(NL) = Quantização Não Linear

Tabela 4.1 - Parâmetros Estatísticos da Codificação Huffman

A tabela 4.1 apresenta alguns parâmetros obtidos pelo programa ANALISE.EXE, referentes à codificação Huffman. O **Número de Códigos** inclui o código EOB (End Of Block) e o código de extensão de zeros (usado quando

run-lenght >16). O comprimento máximo do código de Huffman (**Lmax**) inclui o sinal do coeficiente. Na fig. 4.4.4 temos um exemplo de distribuição de probabilidades de ocorrência de símbolos, ordenados decrescentemente, para o caso Q=2 com busca por máxima semelhança. Apesar da grande quantidade de símbolos (416), a **entropia** resulta baixa (3,83733 bits) devido à concentração de poucos símbolos com grande probabilidade.

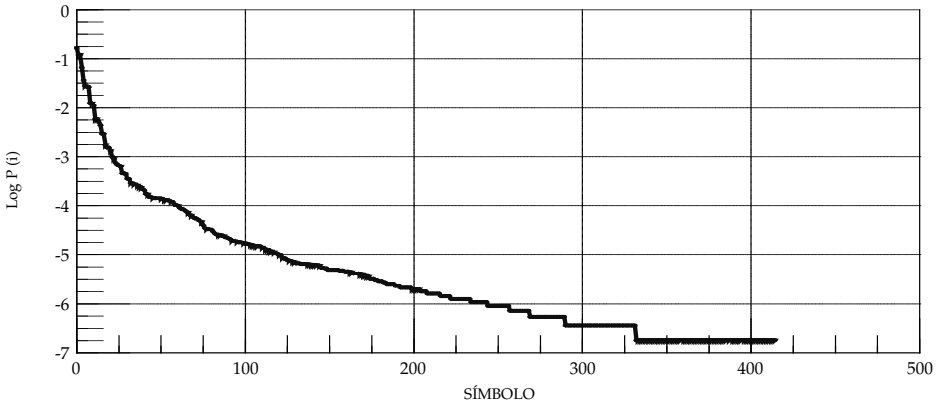


Fig. 4.4.4 - Distribuição de Probabilidades de Símbolos

4.5 Pós-Processamento das Imagens Reconstruídas:

Para taxas de compressão reduzidas (ex.: Fator de Quantização = 2) as imagens reconstruídas são praticamente idênticas às originais; já para taxas de compressão mais elevadas, torna-se perceptível a presença de um ruído superposto à imagem. Nos testes preliminares, por outro lado, chegou-se a observar que as imagens reconstruídas apresentavam menor granulação, pois o truncamento dos coeficientes da DCT na quantização eliminava componentes de baixa amplitude, resultando em uma imagem "arredondada", composta apenas pelas componentes de freqüência espacial preponderantes no bloco.

Com o aperfeiçoamento do processo de detecção de movimento, o erro de predição passou a ser preponderante; o truncamento dos coeficientes da DCT

implica que parte do erro de predição de movimento não será removida, resultando então em um ruído adicional relativamente uniforme na imagem reconstruída.

Neste item serão descritos estudos relativos ao desenvolvimento de um método de pós-processamento da imagem descomprimida, de modo a reduzir a visibilidade do erro de reconstrução.

4.5.1 Características Espectrais da Imagem e do Erro de Reconstrução

Inicialmente procuramos determinar o conteúdo espectral das imagens (Sinal + Ruído) utilizadas neste estudo, de forma a obtermos subsídios para identificação das características que podem influenciar a visibilidade do erro de reconstrução. O estudo a seguir concentrou-se apenas nas freqüências espaciais no sentido horizontal da imagem.

Para caracterizarmos o conteúdo espectral das imagens, aplicamos inicialmente às mesmas uma janela cosenoidal, de modo a eliminarmos os efeitos das bordas:

$$f_w(x, y) = f(x, y) \cdot \left(1 - \cos\left(\frac{\pi \cdot x}{256}\right)\right) \cdot \left(1 - \cos\left(\frac{\pi \cdot y}{240}\right)\right)$$

onde $f(x,y)$ é a imagem original e $f_w(x,y)$ é a imagem com a janela aplicada, admitindo dimensões da imagem de 512 x 480 elementos.

Para uma dada imagem janelada, foi aplicada a Transformada Discreta de Fourier para cada linha, obtendo-se o respectivo espectro, em função da freqüência espacial dada em ciclos/largura da imagem.

Obteve-se então a soma quadrática dos espectros de todas as linhas da imagem, resultando em um espectro médio eficaz (rms) da imagem.

A figura 4.5.1 apresenta o espectro médio de uma imagem original (XA030.BMP), com amplitude expressa em dB e freqüência espacial em ciclos/largura da imagem; já na figura 4.5.2 temos os espectros da mesma imagem, reconstruída, com fatores de quantização Q = 2, 4, 8 e 16.

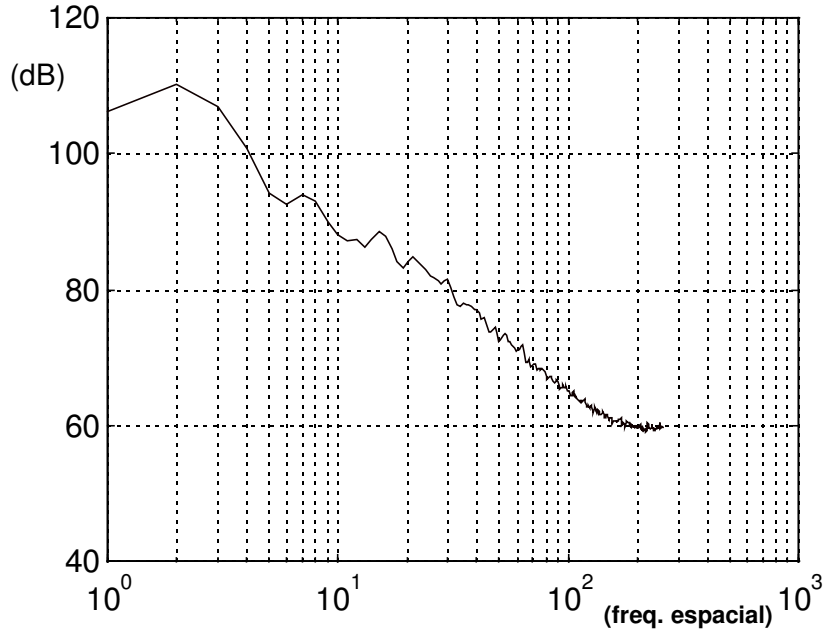


Fig. 4.5.1 - Espectro Médio (Freqüência Espacial Horizontal) - Imagem Original (XA030.BMP)

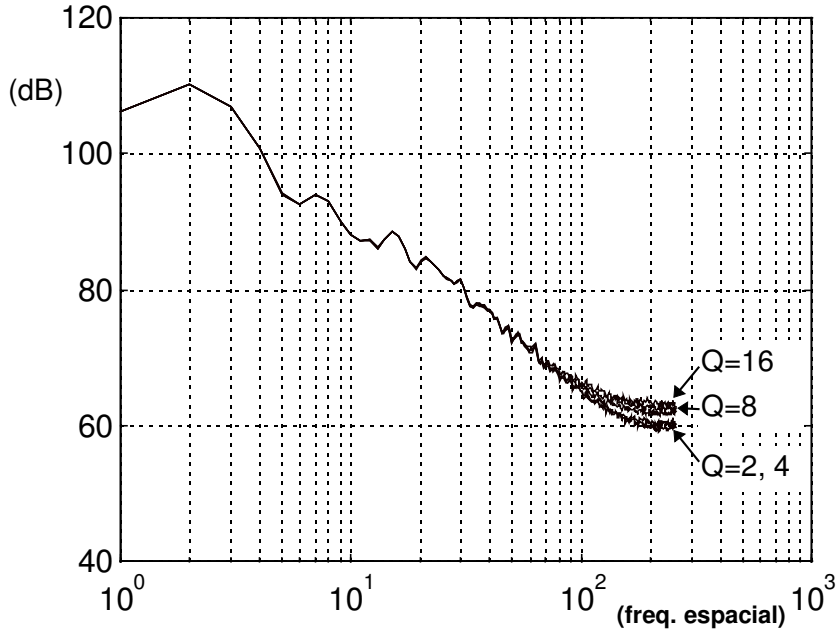


Fig. 4.5.2 - Espectro Médio - Imagens Reconstruídas

O mesmo procedimento foi aplicado ao erro de resconstrução (fig. 4.5.3 e 4.5.4); podemos observar que para os fatores de quantização 2, 4 e 8 o espectro do erro é praticamente uniforme. Conforme podemos verificar pela figura 4.5.4, além de uma diferença de amplitude de cerca de 1 dB, as características espectrais do

erro de reconstrução são similares para os dois processos de predição de movimento estudados.

Pelo gráfico da fig. 4.5.1 podemos estimar o ruído de fundo da imagem original, como sendo a assíntota horizontal da curva apresentada; podemos atribuir a esta imagem um ruído aditivo uniforme com amplitude de aproximadamente 60 dB na escala utilizada. Comparando com a figura 4.5.3, podemos dizer que com os fatores de quantização Q=2 e Q=4 o erro de reconstrução apresenta amplitude inferior ao ruído inerente da imagem. Verificamos, de fato, na figura 4.5.2, que o espectro da imagem reconstruída sofre aumento perceptível, em altas freqüências espaciais, apenas para Q=8 e Q=16.

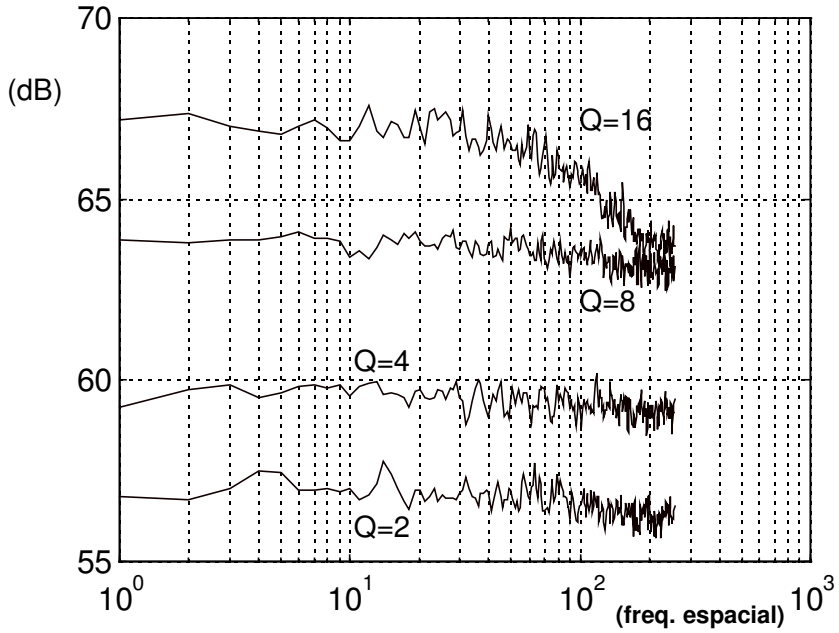


Fig. 4.5.3 - Espectro Médio (Freqüência Espacial Horizontal) do Erro de Reconstrução

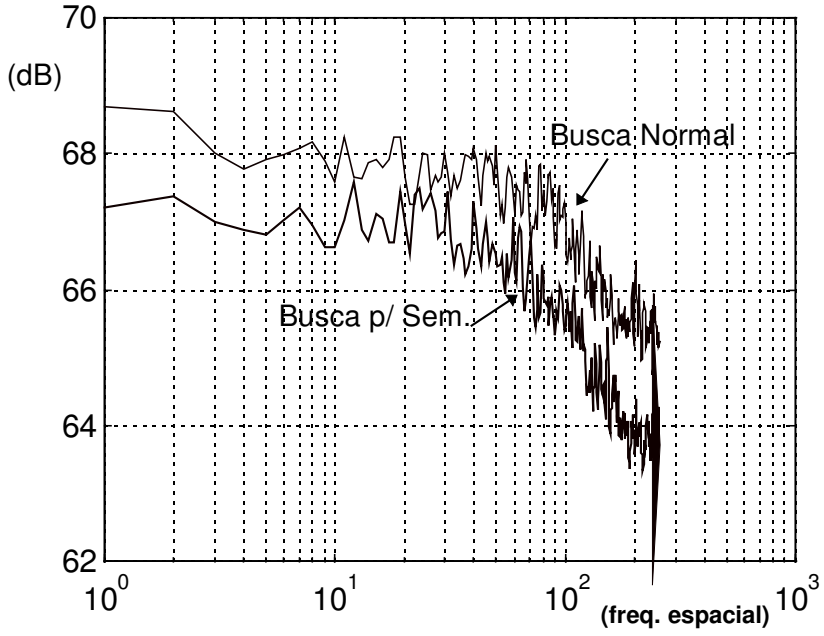


Fig. 4.5.4 - Espectro do Erro de Reconstrução (Fator de Quantização=16, Busca Normal e Busca Por Máxima Semelhança)

4.5.2 Diferenças Espectrais entre Imagem Original e Reconstruída

Embora o erro de reconstrução se apresente como um ruído aditivo com espectro praticamente uniforme, ele torna-se perceptível principalmente nas situações em que sua densidade espectral de potência supera a do próprio sinal original. Embora não seja possível cancelar o erro de reconstrução apenas a partir de um modelamento do mesmo, podemos equalizar o sinal reconstruído de modo que o espectro resultante se aproxime do espectro original.

A figura 4.5.5 apresenta as diferenças entre o espectro médio da imagem original e da imagem reconstruída, para os 4 fatores de quantização utilizados.

Podemos ver que, para Q=2, há uma pequena perda de resposta para freqüências na faixa de 30 a 100 ciclos/largura da imagem, que seria característica do truncamento de coeficientes da DCT; acima de 120 ciclos/largura o ruído aumenta, mantendo-se porém entre 0,5 e 1,0 dB .

Já para Q=4 e acima, o ruído na imagem reconstruída cresce a partir da freqüência de aproximadamente 100 ciclos/largura da imagem.

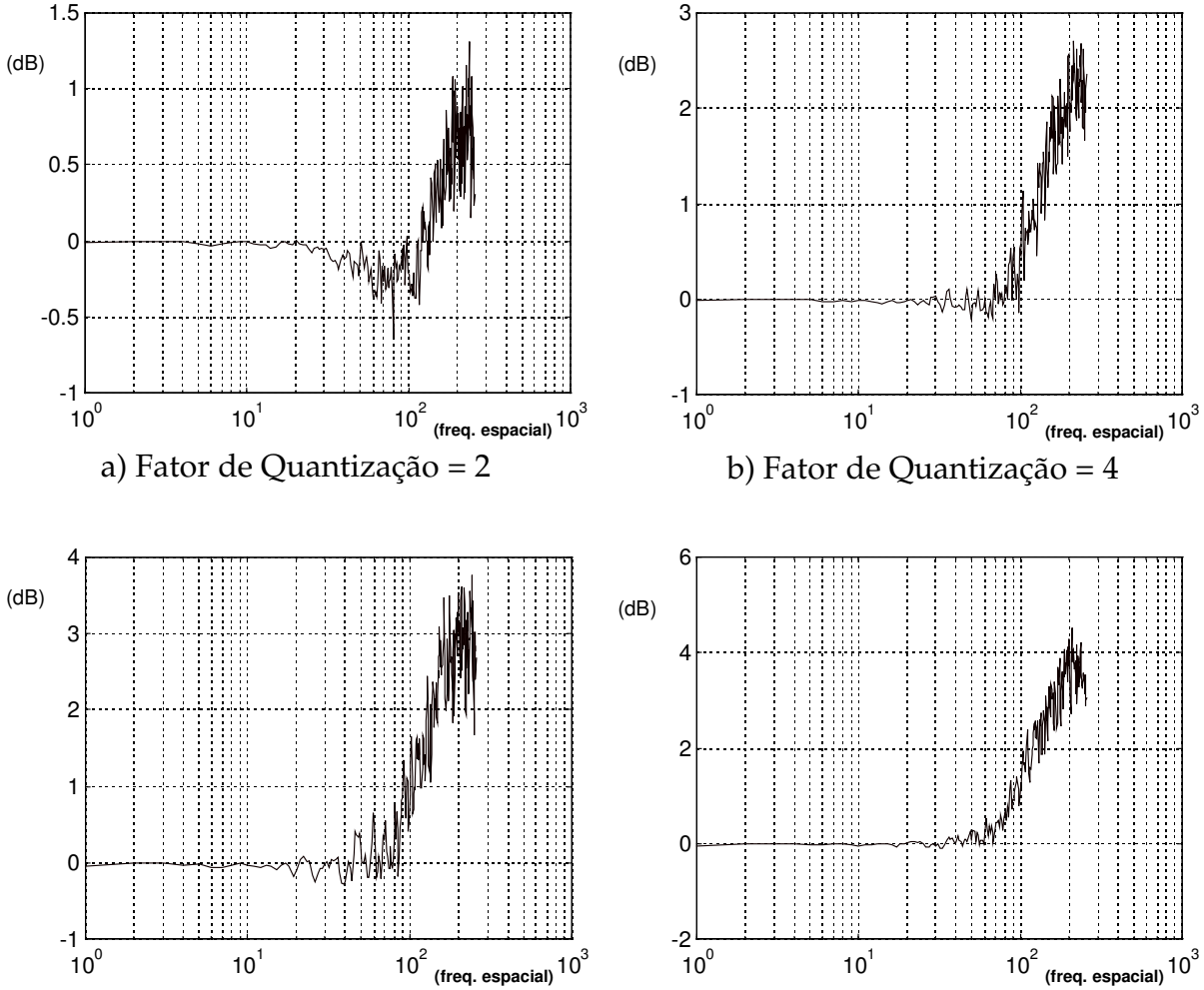


Fig. 4.5.5 - Diferenças Espectrais entre Imagem Original e Reconstruída

4.5.3 Pós-Processamento por Filtragem Passa-Baixas

Para reduzir a visibilidade do erro de reconstrução, construímos um filtro passa-baixas que equaliza a posteriori as imagens reconstruídas, procurando manter o espectro semelhante ao da imagem original. Este filtro foi implementado por um programa (LPF.EXE) que efetua a convolução da imagem reconstruída com uma matriz de resposta ao impulso bidimensional.

A implementação de um filtro ótimo, que equalize perfeitamente o sinal recuperado, é dificultada principalmente pelo fato de que estamos operando com imagens (tanto de entrada como de saída) quantizadas em 8 bits, e o ruído a ser filtrado possui amplitudes próximas ao próprio ruído de quantização. Em

um filtro FIR bidimensional, coeficientes muito pequenos não terão nenhum efeito.

Optamos então por um filtro simples, através da convolução com uma matriz de 3x3 coeficientes. A matriz de resposta ao impulso utilizada, com resultados satisfatórios, foi:

$$H(i,j) = \frac{1}{28}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 20 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

A figura 4.5.6 apresenta a diferença espectral média entre a imagem original e a imagem reconstruída e equalizada, através do filtro acima, para Q=16.

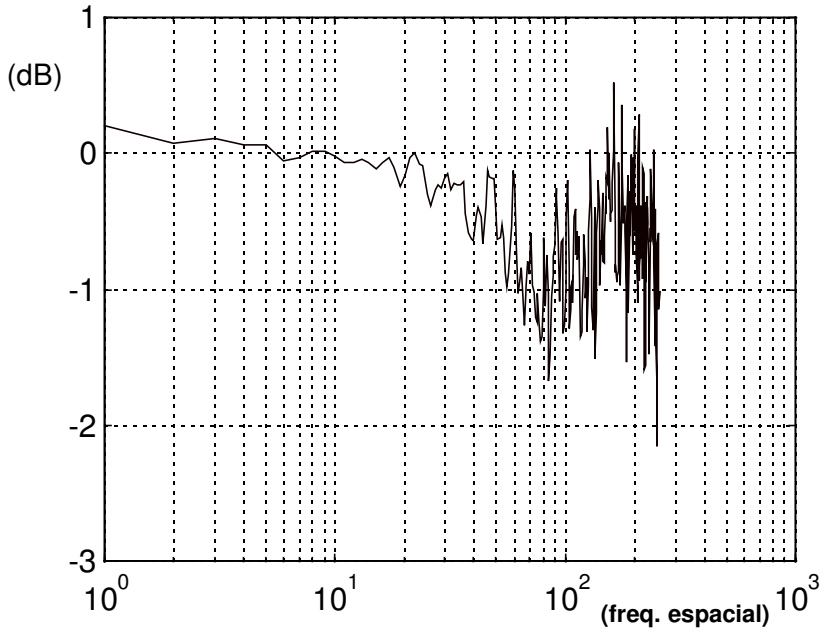


Fig. 4.5.6 - Diferença Espectral entre Imagem Original (XA030.BMP) e Reconstruída após Filtragem (XAP030N.BMP)

Lembramos que o sistema de visão humano tende a atribuir melhor definição a imagens ruidosas do que a imagens livres de ruído, especialmente se estas apresentam poucos detalhes (imagens "arredondadas" parecem fora de foco).

Este fenômeno justifica-se pela dificuldade que a visão humana encontra ao tentar focalizar uma imagem que não apresenta detalhes de alta freqüência

espacial, devido à ausência de um pico bem definido na correlação cruzada entre as imagens percebidas individualmente pelos olhos.

A aplicação do pós-processamento às imagens do presente estudo, na forma descrita, resultou em uma redução de visibilidade de artefatos da compressão, especialmente em áreas com texturas de baixo contraste. Nestas situações, a tendência da visão humana é de criar estruturas inexistentes, ligando entre si texturas espúrias introduzidas pelo processo de compressão. Com a filtragem implementada, essas componentes são atenuadas. Para fator de quantização Q=2, o pós-processamento não apresenta nenhuma melhora, como era de se esperar.

5. Conclusões

No âmbito do presente trabalho foi implementado um sistema de compressão de imagens em movimento, incorporando características direcionadas para aplicação na área de imagens médicas. O desempenho do sistema foi avaliado quando aplicado a uma seqüência de imagens de cine-angiografia, tendo sido caracterizado em termos de taxa de compressão e respectivo erro médio de reconstrução.

Foi desenvolvido um processo de compensação de movimento baseado em semelhança de forma, com a finalidade de otimizar a extração de vetores de movimento em presença de oscilações de intensidade da imagem. Este processo contribuiu consistentemente para uma melhora simultânea na taxa de compressão e na qualidade da imagem reconstruída, quando comparado ao método tradicional.

Testes efetuados com o quantizador não linear apresentaram melhora de desempenho apenas em algumas circunstâncias.

Às imagens reconstruídas foi aplicado um pós-processamento com a finalidade de reduzir a visibilidade do erro adicionado.

No desenvolvimento dos algoritmos citados, o resultado final foi avaliado através de um ambiente de visualização de imagens implementado com a finalidade de permitir comparações de imagens processadas por vários métodos.

Pelos processos implementados, foram obtidas taxas de compressão superiores a 10 : 1 com excelente qualidade de reprodução de imagem .

6. Referências bibliográficas:

FORAN, DAVID J. et al. " Compression Guidelines for Diagnostic Telepathology " , **IEEE Transactions on Information Technology in Biomedicine**, Vol. 1, 1997

GONZALEZ, RAFAEL C.; WOODS, R. E. " **Digital Image Processing**", Addison-Wesley, 1993

HASKELL, B.; PURI, A.; NETRAVALI, A. N. " **Digital Video: An Introduction to MPEG-2**", Chapman & Hall, 1997

HO, BRUCE K. T. et al. " Video Compression of Coronary Angiograms Based on Discrete Wavelet Transform With Block Classification " , **IEEE Transactions on Medical Imaging**, Vol. 15, 1996

LAFORE, ROBERT" **C programming using Turbo C++** ", Howard W. Sams, 1990

LATHI , B. P. " **Modern Digital & Analog Communications** " , Holt, Rinehart and Winston, 1983

MITCHELL, J. et al. " **MPEG Video Compression Standard** ", Chapman & Hall, 1996

MURAT TEKALP, A. " **Digital Video Processing** ", Prentice Hall, 1995

PROAKIS, JOHN G. " **Digital Communications** ", Mc Graw-Hill, 1995

RATIB, O. " **From PACS to the World Wide Web** " (http://expasy2.hcuge.ch/papers0995/ratib.html), Geneva University Hospital, 1995

SPIELMAN, JASON " Application Note AR361 - MPEG Standard ", **Multimedia Device Data Book** - Motorola Semiconductors, 1995

SYMES, PETER D. " **Video Compression** " , McGraw-Hill, 1998

VAN LYSEL, MICHAEL S.; BRONZINO, JOSEPH D. ed. " **The Biomedical Engineering Handbook** " , CRC Press, 1995

WALLACE, GREGORY " The JPEG Still Picture Compression Standard", **IEEE Transactions on Consumer Electronics**, Vol. 38, 1992

WHITAKER,, JERRY "**DTV: The Revolution In Electronic Imaging** ", Mc Graw-Hill, 1998

WILTON, RICHARD" **Programmer's Guide to PC & PS/2 Video Systems** ", Microsoft Press, 1987

7. Listagens dos Programas Desenvolvidos

7.1 - GMPEG.C - Deteção e compensação de movimento, Codificação por DCT, quantização e reordenação dos coeficientes;

7.2 - ANALISE.C - Estatística dos coeficientes da DCT reordenados, geração do dicionário de Codificação Huffman;

7.3 - CODE.C - Codificação por RLE e Huffman dos coeficientes DCT, formatação do arquivo compactado;

7.4 - ERROR.C - Analisa erro de reconstrução;

7.5 - LPF.C - Filtro Passa-Baixas para pós-processamento;

7.6 - FSHOW.TXT - Programa em Visual Basic para visualização de sequencias de imagens.

7.1 GMPEG.C

```
/*   GMPEG.C            */
/*   faz dct do erro de predicao  de movimento
de uma sequencia de arquivos usando busca por semelhanca de forma (r)
Gera arquivo de DCT's nnxxx.DCT ,
arquivo de desloc. vetoriais nnxxx.VET
e arquivo reconstruido nnRxxxG.BMP */


/*    GS 07/02/99  */
/*    Busca por semelhanca 15/11/99  */
/*    aritmetica integer   27/12/99  */
/*    revisao              30/12/99  */


#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "float.h"
#include "math.h"


int row,coluna;               /* indices y,x da imagem           */
int col,bank,i,x,y,mean,yp,xp;      /* indices diversos          */
int hx,hy,lx,ly,mx,my;        /* sub-indices e vetores de movimento */
char cmean;                   /* valor medio de um bloco 8x8     */
unsigned char flag;           /* Sinaliza overflow               */
int a,b;
int tecla;                    /* testa teclado p/ interromper    */

FILE *in,*inold;                /* Arquiivos de entrada (inagem e referencia)
*/
FILE *recout,*dctout,*vetout; /* Arquivos de saida               */

int corr;                     /* Correlacao entre blocos         */
long fator=2;                 /* fator de compressao             */
int corrm,dx,dy;              /* Correlacao minima, vetores      */
char pa[8][8];                /* bloco 8x8 imagem atual          */
int  cpa[4][4];               /* bloco 4x4 c/ meia resolucao     */
char huge pp[480][512];       /* imagem previa                   */
int  huge cpp[240][256];      /* imagem previa com meia resolucao */
int  numframes;               /* quantidade de frames na sequencia*/
char fname[20];               /* nome do arquivo                 */
char filetoken[20];           /* nome basico dos arquivos        */
char lastname[20];
char s1[20];                  /* Strings temporarias             */
char s2[20];
char s3[20];
char s4[20];
int huge pmp[236][252]; /* imagem media de 8x8 com meia resolucao */

char slice[8][520];           /* fatia horizontal de blocos      */
long dct[8][8];               /* matriz de coef. dct temporaria  */
int tab[8][8];                /* tabela de coef. p/ DCT          */
int coef[8][8];               /* coeficientes dct calculados     */
long quant[8][8];             /* tabela de quantizadores         */
long  max[8][8];              /* estat'istica de dct             */
long  min[8][8];
long  med[8][8];
long sum;                     /* acumulador p/ medias            */
char orden[64];               /* matriz reordenada               */
char indic[8][8]=             /* tab. p/ ordenacao               */
      {     { 0, 1, 5, 6,14,15,27,28},
            { 2, 4, 7,13,16,26,29,42},
            { 3, 8,12,17,25,30,41,43},
            { 9,11,18,24,31,40,44,53},
            {10,19,23,32,39,45,52,54},
            {20,22,33,38,46,51,55,60},
            {21,34,37,47,50,56,59,61},
```

```
            {35,36,48,49,57,58,62,63}     };

/* ============== Programa Principal ==========================  */

void main(int argc, const char *argv[])   /* Nome do arq. na  Command Line */
      {
      inittab();                          /* inicializa matriz p/ DCT   */
      strcpy (filetoken,argv[1]);         /* nome basico dos arquivos   */
      numframes = 39;                     /* quantidade de frames       */
      for (bank=1; bank<numframes; bank++)      /* loop para cada frame */
            {
            openfiles();                  /* Abre arquivos in e out     */
            if (bank>1) fseek(inold,(long)1079,SEEK_SET); /* pula header   */
            copyheader();                 /* Copia header do Bitmap     */
            printf("\n%s ",fname);        /* Mostra na tela             */
            for (y=0;y<480;y++)           /* enche matriz imagem previa */
                  {
                  for (x=0; x<512; x++)
                        {
                        if (bank>1) pp[y][x]=getc(inold)-128; /* desconta 128
*/
                        else pp[y][x]=0;   /* Primeiro frame nao tem anterior
*/
                        }
                  }

            halfres();        /* enche matriz imagem previa 1/2 res. */
            for(y=0; y<480; y=y+8)        /* loop para cada slice    */
                  {
                  printf (">");           /* mostra andamento        */
                  if (kbhit()) exit(-1);  /* para se apertar tecla   */
                  fillslice();            /* carrega fatia da imagem */

                  for (x=0; x < 512; x=x+8)     /* loop para cada bloco */
                        {
                        blockfind();       /* Efetua busca na imagem anterior
*/
                        putc((char)(mx-x),vetout);        /*  Arquivo  vetores
*/
                        putc((char)(my-y),vetout);
                        for (a=0; a<8; a++)     /* calcula diferenca    */
                              {
                              for (b=0; b<8; b++)
                                    {
                                    pa[a][b] = (pa[a][b]-pp[my+a][mx+b]);
                                    }
                              }     /* matriz pa[] contem erro de predicao */

                        dct8();     /* calcula dct do bloco em pa[][] */
                                    /* cmean tem o valor medio do bloco   */
                        putc (cmean,dctout); /* coef. medio de 64 amostras */
                        for (row=0; row<8; row++)
                              {
                              for (col=0; col<8; col++)
                                    {
                                    orden[(indic[row][col])]=pa[row][col];
                                    }    /* reordena coeficientes       */
                              }
                        for (col=0; col<64; col++)
                              { putc((char)orden[col],dctout); }
                                    /* arquivo de saida dos coefs.   */
                        idct8();          /* calcula inversa      */

                        for (a=0; a<8; a++)     /* calcula reconstruido */
                              {
                              for (b=0; b<8; b++)
                                    {
                                    slice[a][b+x] = (pa[a][b]+pp[my+a][mx+b]);
                                    }
                              }      /* matriz pa[] contem bloco reconstruido
*/
```

```
                        }           /* fim do loop de bloco       */
                  putslice();       /* salva fatia da imagem      */
                  }           /* fim do loop de fatia       */

            fclose (in);      /* fecha arquivos in e out    */
            fclose (recout);
            if (bank>1) fclose (inold);
            fclose (dctout);
            fclose (vetout);
            }                 /* fim do loop de frame       */
      }                       /* fim do programa main       */


/*======================  funcoes ==============================*/

halfres(void)     /* prepara matriz da imagem previa com 1/2 resolucao  */
      {           /* para busca hierarquica     */
      for (y=0;y<240;y++)           /* coordenadas na matriz saida      */
            {
            for (x=0;x<256;x++)
                  {
                  a=y+y; b=x+x;     /* coordenadas na matriz entrada    */
                  cpp[y][x] =(((int)pp[a][b] +(int)pp[a][b+1] +
                        (int)pp[a+1][b] + (int)pp[a+1][b+1])<<4);
                  }           /*  cpp[][] e' int  (somatoria) de 0 a 16320 */
            }
      for (y=0;y<236;y++)  /* enche matriz c/valores medios de regioes 4x4 */
            {
            for (x=0;x<252;x++)
                  {

a=(((int)cpp[y][x]+(int)cpp[y+1][x]+(int)cpp[y+2][x]+(int)cpp[y+3][x])>>4)

+(((int)cpp[y][x+1]+(int)cpp[y+1][x+1]+(int)cpp[y+2][x+1]+(int)cpp[y+3][x+1])>
>4)

+(((int)cpp[y][x+2]+(int)cpp[y+1][x+2]+(int)cpp[y+2][x+2]+(int)cpp[y+3][x+2])>
>4)

+(((int)cpp[y][x+3]+(int)cpp[y+1][x+3]+(int)cpp[y+2][x+3]+(int)cpp[y+3][x+3])>
>4);

                  pmp[y][x]=a; /* somatoria dos 64 valores  (0 a 16320)*/
                  }
            }
      }


fillslice()         /* Carrega uma fatia horizontal de blocos 8x8 da imagem
*/
      {
      for (row=0; row <8; row++)
            {
            for (col=0; col<512; col++)
                  {
                  slice[row][col] = getc(in)-128; /* subtrai 128        */
                  }
            }
      }


putslice()          /* Salva fatia horizontal de blocos 8x8 da imagem saida
*/
      {
      for (row=0; row <8; row++)
            {
            for (col=0; col<512; col++)
                  {
                  putc (slice[row][col]+128,recout); /* devolve(soma) 128 */
                  }
```

```
            }
      }


dct8()            /* Faz Transformada Discreta de Cossenos (DCT)        */
                  /* opera sobre matriz pa[8][8]           */
      {
      mean=32;               /* para arredondamento do valor final     */
      flag=0;                                 /* sinaliza overflow    */
      for (col=0; col<8; col++)
            {
            for (row=0; row<8; row++)
                  {
                  mean=mean+(int)pa[row][col];  /* Somatoria (valor DC) */
                  }
            }
      mean = mean/64;
      if (mean == 127) mean=126;              /* limita em 126        */
      for (col=0; col<8; col++)               /* produto matricial    */
            {
            for (row=0; row<8; row++)
                  {
                  sum = 0;
                  for (i=0; i<8; i++)
                        {
                        sum  =  sum  +(long)(  tab[col][i]*(((int)pa[row][i])-
mean));
                        }
                  dct[row][col]=qquant(sum,(long)256);
                  }
            }

      for (col=0; col<8; col++)               /* produto matricial    */
            {
            for (row=0; row<8; row++)
                  {
                  sum=0;
                  for (i=0; i<8; i++)
                        {
                        sum = sum + (long)tab[row][i] * dct[i][col];
                        }
                  coef[row][col]=(int)qquant(sum, quant[row][col]);
                  if ((coef[row][col]>127)||(coef[row][col]<(-127))) flag=1;
                  }                           /* houve overflow?      */
            }
      cmean=127;        /* se tem overflow, volta c/ imagem orig. */
      if (flag == 0)
            {           /* se nao tem overflow, sai os coef. DCT  */
            for (col=0; col<8; col++)
                  {
                  for (row=0; row<8; row++)
                        {
                        pa[row][col]=(char)coef[row][col];
                        }
                  }
            cmean = (char)mean;
            }
      }


inittab()         /* Inicializa matriz para calculo da DCT  */
                  /* e matriz de quantizacao conforme fator */
      {
      for (col=0; col<8; col++)
            {
            tab[0][col] = 91;
            max[0][col]=-1000;
            min[0][col]=1000;
            med[0][col]=0;
            for (row=0; row<8; row++)
                  {
```

```
                  quant[row][col]=256*fator;   /* era 512 */
                  if (row>0)
                        {
                  tab[row][col]=(int)(cos((2*col+1)*row*3.14159/16)*128+0.0);
                  max[row][col]=-1000;
                  min[row][col]=1000;
                  med[row][col]=0;
                        }
                  }
            }
      }


int diff8(void)     /* calcula Distorcao Media Absoluta em blocos 8x8  */
      {
      int a,b,c;
      long int sum,dif;
      sum=0;                  /* Somatoria elementos imagem atual       */
      dif=0;                  /* Somatoria Elementos Imagem previa      */
      for (a=0; a<8; a++)
            {
            for (b=0; b<8; b++)
                  {
                  sum=sum+pa[a][b];
                  dif=dif+pp[yp+a][xp+b];
                  }
            }
      dif=(sum-dif);       /*  somatoria  do  bloco  atual  menos  som.  do  bloco
anterior */
      sum = 32;         /* será dividido por 64 */
      for (a=0; a<8; a++)
            {
            for (b=0; b<8; b++)
                  {
                  c=((pa[a][b]-pp[yp+a][xp+b])<<6) - dif;
                              /* Desconta diferencas no valor medio */
                  if (c<0) c=-c;    /* modulo da diferenca  */
                  sum = sum+c;      /* somatoria das diferencas   */
                  }
            }
      return ((int)(sum>>6));       /* arredonda resultado  */
      }

int diff4(void)   /* calcula Distorcao Media Absoluta em blocos 4x4  */
      {
      int a,b;
      long int c;
      long int dif;     /* Somatoria bloco imagem anterior  */
      long int sum;     /* Somatoria DMA da imagem atual    */
      dif=((cpa[0][0]+cpa[0][1]+cpa[0][2]+cpa[0][3])>>4)+
          ((cpa[1][0]+cpa[1][1]+cpa[1][2]+cpa[1][3])>>4)+
          ((cpa[2][0]+cpa[2][1]+cpa[2][2]+cpa[2][3])>>4)+
          ((cpa[3][0]+cpa[3][1]+cpa[3][2]+cpa[3][3])>>4);
      dif=dif-pmp[yp][xp];          /* dif de 0 a 16320     */
      sum = 0;
                  c=cpa[0][0]-cpp[yp][xp]-dif;    /* literal p/ velocidade */
                  if (c<0) sum=sum-c;
                  else sum = sum+c;
                  c=cpa[0][1]-cpp[yp][xp+1]-dif;
                  if (c<0) sum=sum-c;
                  else sum = sum+c;
                  c=cpa[0][2]-cpp[yp][xp+2]-dif;
                  if (c<0) sum=sum-c;
                  else sum = sum+c;
                  c=cpa[0][3]-cpp[yp][xp+3]-dif;
                  if (c<0) sum=sum-c;
                  else sum = sum+c;
                  c=cpa[1][0]-cpp[yp+1][xp]-dif;
                  if (c<0) sum=sum-c;
                  else sum = sum+c;
                  c=cpa[1][1]-cpp[yp+1][xp+1]-dif;
```

```
                    if (c<0) sum=sum-c;
                    else sum = sum+c;
                    c=cpa[1][2]-cpp[yp+1][xp+2]-dif;
                    if (c<0) sum=sum-c;
                    else sum = sum+c;
                    c=cpa[1][3]-cpp[yp+1][xp+3]-dif;
                    if (c<0) sum=sum-c;
                    else sum = sum+c;
                    c=cpa[2][0]-cpp[yp+2][xp]-dif;
                    if (c<0) sum=sum-c;
                    else sum = sum+c;
                    c=cpa[2][1]-cpp[yp+2][xp+1]-dif;
                    if (c<0) sum=sum-c;
                    else sum = sum+c;
                    c=cpa[2][2]-cpp[yp+2][xp+2]-dif;
                    if (c<0) sum=sum-c;
                    else sum = sum+c;
                    c=cpa[2][3]-cpp[yp+2][xp+3]-dif;
                    if (c<0) sum=sum-c;
                    else sum = sum+c;
                    c=cpa[3][0]-cpp[yp+3][xp]-dif;
                    if (c<0) sum=sum-c;
                    else sum = sum+c;
                    c=cpa[3][1]-cpp[yp+3][xp+1]-dif;
                    if (c<0) sum=sum-c;
                    else sum = sum+c;
                    c=cpa[3][2]-cpp[yp+3][xp+2]-dif;
                    if (c<0) sum=sum-c;
                    else sum = sum+c;
                    c=cpa[3][3]-cpp[yp+3][xp+3]-dif;
                    if (c<0) sum=sum-c;
                    else sum = sum+c;
      return ((int)(sum>>4));
      }


openfiles(void)   /* Abre arquivos para cada  frame   */
      {
      strcpy (fname,filetoken);
      strcpy (s2,filetoken);
      strcat (s2,"P");              /* s2= arqP            */
      itoa (bank,s1,10);
      if ( strlen(s1) < 3 ) {strcat (fname,"0");strcat(s2,"0");}
      if ( strlen(s1) < 2 ) {strcat (fname,"0");strcat(s2,"0");}
      strcat (fname,s1);            /* fname = arq001      */
      strcat (s2,s1);               /* s2 = arqP001        */
      strcpy (s3,fname);            /* s3 = arq001         */
      strcpy (s4,fname);            /* s4 = arq001         */
      strcat (fname,".BMP");        /* fname = arq001.bmp  */
      strcat (s2,"G.BMP");          /* s2 = arqP00G1.BMP   */
      strcat (s3,"G.DCT");          /* s3 = arq001G.DCT    */
      strcat (s4,"G.VET");          /* s4 = arq001G.VET    */

      if ((in = fopen(fname,"rb"))==0) exit(-1); /* imagem original     */
      if ((recout= fopen(s2,"wb"))==0) exit(-1); /* imagem reconstruida */
      if (bank>1)             /* primeeira imagem nao tem anterior     */
         {
         if  ((inold=fopen(lastname,"rb"))==0)  exit(-1);  /*  imagem  reconst.
anterior*/
         }
      strcpy (lastname,s2);
      if  ((dctout=  fopen(s3,"wb"))==0)  exit(-1);         /*  arquivo  de  DCT's
*/
      if  ((vetout=  fopen(s4,"wb"))==0)  exit(-1);        /*  arquivo  de  vetores
*/
      }


copyheader(void)   /* Copia header de arquivo .BMP    */
      {
      int i;
```

```
      for (i=0; i<1079; i++)
            putc(getc(in),recout);
      }


blockfind(void)    /* Faz  busca (compensacao de movimento) 4x4 e 8x8   */
      {
      for (a=0; a<8; a++)     /* carrega bloco 8x8 da fatia corrente    */
            {
            for (b=0; b<8; b++)
                  {
                  pa[a][b] = slice[a][b+x];
                  }
            }
      for (a=0; a<4; a++)     /* prepara bloco 4x4 com meia resolucao   */
            {
            for (b=0; b<4; b++)
                  {
                  cpa[a][b] = (((int)pa[2*a][2*b] + (int)pa[2*a][2*b+1] +
                      (int)pa[2*a+1][2*b] + (int)pa[2*a+1][2*b+1]+2) <<4);
                  }
            }
      corrm=32767;            /* valor inicial (alto) */
      hx = (int)x/2+32; if (hx >252) hx=252;    /* estabelece limites   */
      lx = (int)x/2-32; if (lx < 0)  lx = 0;     /* para busca na img. previa
*/
      hy = (int)y/2+32; if (hy >236) hy=236;    /* (meia resolucao)     */
      ly = (int)y/2-32; if (ly < 0)  ly = 0;

      for (xp=lx; xp<hx; xp++)             /* varre janela na imagem anterior
*/
            {
            for (yp=ly; yp<hy; yp++)
                  {
                  corr = diff4();
                  if (corr < (corrm))
                        {
                        corrm=corr;       /* minima diferenca     */
                        mx=xp; my=yp;     /* vetores correspondentes    */
                        }
                   }
            }
                              /* repete para rfesolucao plena (8x8)     */
      corrm=32767;
      hx = 2*mx+3; if (hx >504) hx=504;   /* estabelece limites         */
      lx = 2*mx-3; if (lx < 0)  lx = 0;   /* para busca na img. previa  */
      hy = 2*my+3; if (hy >472) hy=472;   /* em torno do minimo local   */
      ly = 2*my-3; if (ly < 0)  ly = 0;
      for (xp=lx; xp<hx; xp++)
            {
            for (yp=ly; yp<hy; yp++)
                  {
                   corr = diff8();
                   if (corr < corrm)
                         {
                         corrm=corr;     /* minima diferenca     */
                         mx=xp; my=yp;   /* vetores finais       */
                         }
                   }
            }
      }


qquant(long x, long modul)    /* arredonda (quantiza) valores c/ sinal  */
   {
   if (x<0)
      {
      x=(modul/2)-x;
      x = -(x/modul);
      }
   else
```

```
      {
      x = (x+(modul/2))/modul;
      }
   return x;
   }


idct8()           /* Calcula DCT inversa sobre pa[][] se cmean <127   */
   {
      mean = (int)cmean;
      if (mean < 127)
      {
      for (col=0; col<8;col++)
            {
            for (row=0; row<8; row++)
                  {
                  sum=0;
                  for (i=0; i<8; i++)     /* Produto matricial    */
                        {
                        sum = sum +(long)tab[i][col]*((long)(pa[row][i]));
                        }
                  dct[row][col]=(long)qquant((sum*fator),(long)512);    /*  era
256 *//* NAO desconta 128 cada coef.*/
                  }
            }

      for (col=0; col<8; col++)
            {
            for (row=0; row<8; row++)
                  {
                  sum=64;
                  for (i=0; i<8; i++)
                        {
                        sum = sum + (long)tab[i][row] * dct[i][col];
                        }
                  sum = sum/128 + (long)mean;
                  if (sum>127) sum=127;
                  if (sum<(-127)) sum=(-127);
                  pa[row][col]=(char)sum;
                  }
            }
      }
   }
```

7.2 ANALISE.C

```
/*    ANALISE.C                     */
/*    Mede estatisticas de DCT      */
/*    GS 21/05/97                   */
/*    Huffman 02/01/00              */

#include "stdio.h"
#include "stdlib.h"
#include "dos.h"
#include "graph.h"
#include "string.h"
#include "float.h"
#include "math.h"


int bank,i,x;
int numbanks;           /* numero de frames     */
char kk;                /* entrada do arquivo   */
FILE *in;               /* arquivo IN (DCT)     */
FILE *out;              /* arquivo OUT (stats)  */
FILE *out2;             /* arquivo OUT (codificacao huffman)      */
char s1[20];            /* string de uso geral  */
char s2[30];
char sinp[30];

long huge codes[65][256];     /* contagem indexada por Run-lenght e Valor */
long prob[4096];              /* contagem de simbolos (ordenados)       */
signed char code[4096];       /* bit 0 ou 1 (-1 para fim de codigo)     */
int pointer[4096];            /* ponteiro de cadeia de codificacao      */
int mag[4096];                /* magnitude do simbolo codificado        */
int runl[4096];               /* run-lenght do simbolo codificado       */
long huffman[4096];           /* codigo huffman correspondente          */
int huflen[4096];             /* comprimento do codigo huffman          */

long minimo,secmin;           /* minimos da ordenacao huffman           */
int inext,imin,isec;          /* indexadores para ordenacao huffman     */
int maxcodes;                 /* quantidade de codigos nao nulos        */
int lencodes;                 /* comprimento maximo dos codigos         */

int i,j,k;              /* indexadores gerais   */
int count,a;            /* contagem de zeros    */
long h1,h2;
float totcodes;         /* total de codigos p/ calculo entropia   */
float entropia;         /* entropia dos codigos */
float probab;           /* probabilidade do codigo    */

/* ==================== Programa Principal ========================== */

void main(int argc, const char *argv[])
      {
      for (i=0; i<65; i++)    /* ate' 64 zeros  */
            {
            for (j=0; j<256; j++)
                  { codes[i][j]=0;} /* inicializa matriz de contagens   */
            }
      numbanks = 39;
      out  =  fopen  ("stats","w");                 /*  abre  arquivo  de  saida
*/
      out2=  fopen  ("huffman.txt","w");       /*  abre  arquivo  de  codificacao
*/
      for (bank=1; bank<numbanks; bank++)
            {                                 /* abre arquivos de DCT       */
            strcpy(s2,"0");
            strcpy(sinp,argv[1]);
            itoa (bank,s1,10);
            if (strlen(s1)<2) strcat(s2,"0");
            strcat(s2,s1);
            strcat(s2,"g.dct");                   /* s2= 0xxg.dct     */
            strcat(sinp,s2);
```

```
            if((in = fopen(sinp,"rb"))==0)
                  {
                  fclose(out);
                  exit(-1);
                  }
            printf("header ok  %d\n",bank);  /* sinaliza andamento   */

            for (i=0; i<3840; i++)           /* contador de blocos   */
                  {
                  a=getc(in);                   /* componente DC        */
                  count=0;
                  if (a==127)                   /* e' by-pass?          */
                        {
                        for (j=0; j<64; j++) x=getc(in);    /* descarta */
                        codes[64][1]++;         /* (64,1) = bypass      */
                        }
                  else                          /* coeficientes DCT     */
                        {
                        for (j=0; j<64; j++)
                              {
                              kk=(signed char)(getc(in));
                              if (kk<0) kk=-kk;    /* contabiliza modulo   */
                              if (kk==0) count++;      /* conta zeros    */
                              else
                                    {           /* valor nao nulo       */
                                    if (count > 16)
                                          {
                                          count = count-16;
                                          codes [16][0]++;
                                          }     /* (16.0) = 16 zeros    */
                                    if (count > 16)
                                          {
                                          count = count-16;
                                          codes [16][0]++;
                                          }
                                    if (count > 16)
                                          {
                                          count = count-16;
                                          codes [16][0]++;
                                          }        /*  ate'  48  zeros  seguidos
*/
                                    codes[count][kk]++;
                                    count=0;        /*  contabiliza  (count,kk)
*/
                                    }
                              }
                        codes[64][0]++;             /*  (64,0)  =  End  Of  Block
*/
                        }
                  }
            fclose(in);                      /* fecha IN             */
            }                                /* termina loop p/ cada frame */

        totcodes=0;
        for (i=0; i<65; i++)             /* gera arquivo de saida      */
            {                            /* varre todos os run-lenght  */
            for (j=0; j<256; j++)        /* e todos os valores de DCT  */
                  {
                  if (codes[i][j] != 0)   /* imprime codigos nao nulos  */
                        {
                        fprintf  (out,  "%ld;  %d;  %d  \n",codes[i][j],i,(signed
char)j);
                        totcodes = totcodes+codes[i][j]; /* conta simbolos*/
                        }
                  }
            }
        fclose(out);                     /* fecha OUT            */

        printf ("\n total codigos= %f",totcodes);
        entropia = 0;
        for (i=0; i<65; i++)
```

```
            {                                /* varre todos os run-lenght  */
            for (j=0; j<256; j++)            /* e todos os valores de DCT  */
                  {
                  if (codes[i][j] != 0)      /* so' p/ valoes nao nulos    */
                        {
                        probab = ((float)codes[i][j])/(float)totcodes;

                        entropia = entropia-(probab*(log(probab/2))/log(2));
                        }
                  }
            }
        printf ("\n entropia= %f\n",entropia);

/*======== codificacao de huffman sobre simbolos com contagem nao nula  */

      for (i=0; i<4096; i++)
            {
            code[i]=-1;       /* inicializa matrizes p/ codificacao     */
            pointer[i]=0;     /* ponteiro p/ lista ligada (arvore)      */
            prob[i]=0;        /* contagem do simbolo (probabilidade)    */
            runl[i]=0;        /* run lenght do simbolo                  */
            mag[i]=0;         /* magnitude do simbolo (sem sinal)       */
            huffman[i]=0;     /* codigo Huffman do simbolo(binario)     */
            huflen[i]=0;      /* comprimento do codigo Huffman          */
            }

      k=1;
      for (i=0; i<65; i++)
            {                                /* varre todos os run-lenght  */
            for (j=0; j<256; j++)            /* e todos os valores de DCT  */
                  {
                  if (codes[i][j] != 0)      /* separa  codigos nao nulos  */
                        {
                        mag[k]=j;
                        runl[k]=i;
                        prob[k]=codes[i][j];
                        k++;                 /* coloca apenas nao nulos    */
                        }
                  }
            }

      maxcodes=k;                   /* final da lista de simbolos       */
      inext=k;                      /* inicio de simbolos conjuntos     */

      printf ("maxcodes= %d\n",maxcodes);
      imin=1;
      isec=1;
      while ((imin > 0) && (isec > 0))
            {
            findmin();                 /* dois simbolos de menor probabilidade
*/
            if ((imin>0) && (isec>0))     /* enquanto existirem...      */
                  {
                  prob[inext] = prob[imin]+prob[isec];
                  prob[imin] = 0;          /* cria simbolo com prob. conjunta
*/
                  prob[isec] = 0;
                  code[imin] = 0;         /* atribui codigos 0 e 1      */
                  code[isec] = 1;
                  pointer[imin] = inext;   /* lista ligada para novo simbolo
*/
                  pointer[isec] = inext;
                  inext++;
                  }
            }                       /* repete ate' acabar composicao    */

      printf ("inext final= %d\n",inext);

      lencodes = 0;
      for(i=0; i<maxcodes; i++)     /* recupera codigo de cada simbolo  */
            {
```

```
            k=i;                    /* percorre lista ligada           */
            while (code[k] >= 0)    /* ate' acabar concatenacao        */
                  {                 /* combina codigos em cada ligacao */
                  huffman[i]=(huffman[i]+huffman[i])+code[k];
                  huflen[i]++;      /* cresce comprimento do codigo    */
                  k=pointer[k];     /* aponta p/ proxima ligacao       */
                  }
            if (huflen[i] > 0)
                  {                    /* cria arquivo de saida (huffman.txt)
*/

                  if (huflen[i] > lencodes) lencodes = huflen[i];
                  fprintf(out2, "%d \n", runl[i]);    /* run lenght     */
                  fprintf(out2, "%d \n", mag[i]);     /* magnitude      */
                  h1 = huffman[i];                    /* codigo(ASCII)  */
                  for (j=0; j<huflen[i]; j++)
                        {
                        h2 = h1/2;
                        if (h1 == h2+h2)
                              {
                              fprintf (out2, "0");
                              }
                        else
                              {
                              fprintf (out2, "1");
                              }
                        h1 = h2;
                        }
                  fprintf (out2,"\n\n");  /* pila linha entre codigos   */
                  }
            }
      printf ("comprimento maximo = %d bits\n",lencodes);
      fclose (out2);                      /* fecha arquivo saida        */
      }

/* ============== funcoes  =================*/

findmin()               /* acha 2 codigos com menor probabilidade */
      {
      long minimo = 10000000;
      int   i;
      imin = 0;               /* ponteiro para minimo             */
      isec = 0;               /* ponteiro p/ vice-minimo          */
      for (i=0; i<4096; i++)
            {
            if ((minimo > prob[i]) && (prob[i] > 0))
                  {
                  minimo=prob[i];
                  imin=i;     /* aponta simbolo de menor probab.  */
                  }           /* apenas para simbolos nao nulos   */
            }
      minimo=10000000;
      for (i=0; i<4096; i++)
            {
            if (i != imin)    /* exceto simbolo ja' encontrado    */
                  {
                  if ((minimo > prob[i]) && (prob[i] > 0))
                        {
                        minimo=prob[i];
                        isec=i;     /* simbolo seguinte           */
                        }
                  }
            }
      }
```

7.3 CODE.C

```
/*    CODE.C                              */
/*    Compacta arquivo DCT com Huffman    */
/*    utiliza codebook em HUFFMAN.TXT     */
/*    Insere vetores de movimento         */
/*    GS 28/05/97                         */
/*    Revisao 02/01/2000                  */


#include "stdio.h"
#include "stdlib.h"
#include "dos.h"
#include "string.h"
#include "float.h"
#include "math.h"


int bank,i,x,y;   /* contador de frames etc.    */
int nul1;
int numbanks=39;  /* numero de frames           */
int nul2;
char k,sig;
FILE *in;         /* arquivo IN (DCT e Huffman) */
FILE *invet;      /* arquivo IN (vetores mov.)  */
FILE *out;        /* arquivo OUT (estatisticas) */
FILE *outc;       /* arquivo OUT (.CMP)         */
char s1[30];      /* string de uso geral  */
char s2[30];
char sinp[30];
char srec[30];
long huge codes[65][256];     /* simbolo indexado por Run-lenght e Valor */
int huge len[65][256];        /* tamanho do simbolo em bits              */
int runl,val;                 /* run lenght e valor               */
int bitcount,saida;           /* contagem de bits colocados       */
int esc,byp,i,j,a;            /* escape, bypass etc.              */
long count;                   /* conta bits colocados na saida    */

/*======================= Programa Principal ====================*/

void main(int argc, const char *argv[])
      {
                              /*    inicializa matrizes           */
      for (runl=0; runl<65; runl++)
            {
            for (val=0; val<256; val++)
                  {
                  codes[runl][val] = 0;   /* codigo Huffman             */
                  len[runl][val] = 0;     /* comprimento do codigo      */
                  }
            }

      in = fopen("huffman.txt","r");      /* le arquivo com codebook    */
      for (i=0; i<2048; i++)              /* ate' 2048 codigos          */
            {
            if((fgets(s2,30,in))==NULL) break;  /* ate' o final         */
            runl=atoi(s2);                /* le run lenght              */
            fgets(s2,30,in);
            val=atoi(s2);                 /* le valor coeficiente       */
            if (val<0) val = val+256;
            fgets(s2,30,in);              /* le codigo binario corresp. */
            x=strlen(s2)-1;
            fgets(s1,30,in);              /* le linha em branco         */
            y=0;
            for (j=0; j<x; j++)           /* converte string p/numerico */
                  {
                  y=y+y+*(s2+x-1-j)-48;   /* ASCII para numerico        */
                  }                        /* codigo guardado de tras p/ diante
*/
            codes[runl][val]=(long)y;
```

```
            len[runl][val]=x;             /* comprimento do codigo      */
            }                 /* terminou de carregar matriz de codigos */
      fclose(in);
      printf("%d codigos\n",i);
      out = fopen ("counts","w");         /* saida (estatisticas)       */
      for (bank=1; bank<numbanks; bank++)
                                          /* loop para cada frame       */
            {
            strcpy(s2,"0");
            strcpy(sinp,argv[1]);
            strcpy(srec,argv[1]);
            itoa (bank,s1,10);
            if (strlen(s1)<2) strcat(s2,"0");
            strcat(s2,s1);              /* s2 = "0xx"                 */
            strcat(srec,s2);            /* srec = "XA0xx"             */
            strcpy(s1,srec);
            strcat(srec,".cmp");        /* srec = "XA0xx.CMP"         */
            strcat(s2,"g.dct");         /* s2 = "0xxG.DCT"            */
            strcat(sinp,s2);            /* sinp = ex.: "XA001G.DCT"   */
            strcat(s1,"g.vet");         /* s1 = "XA0xxG.VET"          */

            if((in = fopen(sinp,"rb"))==0) exit(-1);
            outc = fopen(srec,"wb");   /* abre arquivos IN e OUT     */
            if ((invet = fopen(s1,"rb"))==0) exit(-1);

            printf("\n header ok  %d",bank);       /* sinaliza       */
            fprintf(out,"\n header ok  %d",bank);

            count=0;esc=0;byp=0;        /* inicializa contagens       */
            saida=0; bitcount=0;
            for (i=0; i<3840; i++)      /* contador de blocos         */
                  {
                  a=getc(invet);           /* vetor de movimento         */
                  putdir(6,a);
                  a=getc(invet);
                  putdir(6,a);             /* coloca vetores na saida    */
                  a=getc(in);              /* coeficiente DC da DCT      */
                  runl=0;
                  putdir(8,a);             /* coloca valor na saida      */
                  if (a==127)              /* bypass da DCT              */
                        {
                        byp++;
                        for (j=0; j<64; j++)
                              {
                              x=getc(in);
                              putdir(8,x);         /*  copia  bytes  by-passados
*/
                              }
                        }
                  else
                        {                  /* codigo normal              */
                        for (j=0; j<64; j++)
                              {
                              k=(signed  char)(getc(in));        /*  coef.  DCT
*/
                              sig = 0;                           /* sinal coef.
*/
                              if (k<0)
                                    {
                                    k=256-k;          /* elimina sinal  */
                                    sig=1;
                                    }
                              if (k==0)
                                     {
                                     runl++;       /* conta zeros    */
                                     }
                              else
                                    {
                                    if (runl > 16)
                                          {
                                          runl = runl-16;
```

```
                                        putcode  (16,0);      /*  sai  (16,0)
*/
                                        }
                                  if (runl > 16)
                                        {
                                        runl = runl-16;
                                        putcode (16,0);
                                        }
                                  if (runl > 16)
                                        {
                                        runl = runl-16;
                                        putcode (16,0);
                                        }                    /* ate' 48 zeros
*/
                                  if (len[runl][k]==0)
                                        {
                                        esc++;               /* nao tabelado
*/
                                        putcode(runl,k);
                                        putbits(1,sig);
                                        }
                                  else
                                        {
                                        putcode(runl,k);
                                        putbits(1,sig);    /* sinal do coef.
*/
                                        }
                                  runl=0;
                                  }
                            }
                      }
                puteob();         /* Fim de Bloco (EOB)         */
                }
                      printf(".");

            if  (bitcount  !=  0)  putbits((8-bitcount),0);  /*  esgota  bits
pendentes */
            fclose(in);               /* fecha frame IN e OUT       */
            fclose(invet);
            fclose(outc);
            printf (" Bytes: %ld  Esc: %d  Bypass: %d",count>>3,esc,byp);
            fprintf(out," Bytes: %ld  Esc: %d  Bypass: %d",count/8,esc,byp);
            }                         /* fim do loop de frames      */
        fclose(out);                  /* fecha estatisticas         */
        }

/*======================  funcoes ==============================*/

putcode(int runl, int  val)         /* coloca codigo de huffman         */
      {
      if (len[runl][val] == 0)
            {                       /*     escape           */
            putdir (8,0xBE);        /* codigo de escape     */
            putdir (4,len);         /* run lenght           */
            putdir (8,val);         /* valor explicito      */
            }
      else                          /* codigo tabelado      */
            {
            putbits(len[runl][val],codes[runl][val]);
            }                       /* comprimento e codigo */
      }


puteob(void)                        /* coloca codigo de fim de bloco    */
      {
      putcode (64,0);
      }


putflush(void)                      /* esgota bits restantes      */
      {
```

```
      if (bitcount != 0) putbits((8-bitcount),0);
      }


putbits(int len, long val)           /* coloca codigo de comprimento variavel
*/
      {
      int i,bit;
      long byte;
      count=count+(long)len;        /* contabiliza bits saida           */
      for (i=0; i<len; i++)         /* varre comprimento do codigo      */
            {
            bit=0;
            byte=val>>1;            /* bit mais significativo           */
            if ( val != (byte+byte)) bit=1;     /* bit mais significativo */
            val=byte;
            bitcount++;
            saida=(saida+saida)+bit;
            if (bitcount == 8 )     /* juntou 8 bits?       */
                  {
                  bitcount=0;
                  putc (saida,outc); /* sai 1 byte no arquivo de saida   */
                  saida=0;
                  }
            }
      }

putdir(int len, int  val)            /* coloca codigo de comprimento variavel
*/
      {                                    /* sem inversao de padrao de bits
*/
      int byte,i;
      long outp;
      outp=0;
      for (i=0; i<len; i++)         /* varre comprimento do codigo      */
            {
            outp=(outp+outp);       /* inverte palavra (LSb com MSb)    */
            byte=val>>1;            /* bit menos  significativo         */
            if ( val != (byte+byte)) outp++;
            val=byte;
            }
      putbits(len,outp);
      }
```

7.4 ERROR.C

```
/*   ERROR.C      */
/*   Calcula erro entre original e dct \TEMP\XA014.BMP  */
/*    GS 30/04/97   */

#include "stdio.h"
#include "stdlib.h"
#include "dos.h"
#include "string.h"
#include "float.h"
#include "math.h"

int row,coluna;
int col,bank,i,x,y,mean;
unsigned char r,g,b;
FILE *in, *in1;
FILE *out,*out2;
char s1[20]; /* string de uso geral */
char s2[30];
char sinp[30];
char srec[30];
char serr[30];
float noise;
float tot;


void main(int argc, const char *argv[])
      {
        out2 = fopen ("errlog","w");      /* arquivo de log dos erros */
        for (bank=1; bank<39; bank++)
              {
              strcpy(s2,"0");
              strcpy(sinp,argv[1]);
              strcpy(srec,argv[1]);
              strcat(srec,"p");
              strcpy(serr,argv[1]);
              strcat(serr,"e");
              itoa (bank,s1,10);
              if (strlen(s1)<2) strcat(s2,"0");
              strcat(s2,s1);
              strcat(srec,s2);
              strcat(srec,"g.bmp");

              strcat(s2,".bmp");                  /* s2= 0xx.bmp */

              strcat(sinp,s2);

              strcat(serr,s2);
              in = fopen(sinp,"rb"); if (in==0) exit(-1);
              in1= fopen(srec,"rb");
              out= fopen(serr,"wb");
              for (i=0; i<1079; i++)
                 {
                 r=getc(in1);            /* copia header .BMP */
                 r=getc(in);
                 putc(r,out);
                 }
              printf("header ok  %d\n",bank);
              noise=0;
              tot=0;
              for (row=0; row<480; row++) /* contador de linhas */
                 {
                 for (col=0; col<512; col++)
                       {
                       x=(getc(in)-getc(in1));
                       if((col>20)&&(col<492)&&(row>20)&&(row<460))
                             {
                             noise=noise+(float)x*x;
                             tot++;
```

```
                        }
                    x=16*x+128;
                    if (x>255) x=255;
                    if (x<0) x=0;
                    putc((char)x,out);
                    }
                }
            fclose(in);
            fclose(in1);
            fclose(out);
            fprintf(out2,"%d; %f \n",bank,sqrt(noise/tot));
            }
    fclose(out2);
    }
```

7.5 LPF.C

```
/*   LPF.C            */
/*   Filtro para simulação de resposta em frequencia */
/*    GS 01/09/99   */

#include "stdio.h"
#include "stdlib.h"
#include "dos.h"
#include "string.h"
#include "float.h"
#include "math.h"

int row,coluna;
int col,bank,i,j,x;

union REGS regs;
FILE *in;
FILE *out;

char s1[30]; /* string de uso geral */
char s2[30];
char sinp[30];
char srec[30];
char serr[30];
int huge buffer[480][512];

void main(int argc, const char *argv[])
        {
        for (bank=1; bank<39; bank++)
               {
               strcpy(s2,"0");
               strcpy(srec,argv[1]);
               strcat(srec,"p");
               strcpy(serr,argv[1]);
               strcat(serr,"p");
               itoa (bank,s1,10);
               if (strlen(s1)<2) strcat(s2,"0");
               strcat(s2,s1);
               strcat(srec,s2);
               strcat(serr,s2);
               strcat(srec,"g.bmp");                  /* s2= p0xxg.bmp */
               strcat(serr,"n.bmp");

               in = fopen(srec,"rb"); if (in==0) exit(-1);
               out= fopen(serr,"wb");           /* abre arquivos  */
               for (i=0; i<1079; i++)
                  {
                  r=getc(in);             /* copia header .BMP    */
                  putc(r,out);
                  }
               printf("header ok \n");
               for (row=0; row<480; row++)      /* contador de linhas */
                  {
                  for (col=0; col<512; col++)
                        {                         /* le arquivo de entrada */
                        x=(getc(in));
                        buffer[row][col] = x;
                        }
                  }
               fclose(in);
               for (row=0; row<478; row++)      /* contador de linhas */
                  {
                  for (col=0; col<510; col++)
                        {                         /* filtro passa baixas */
                    x=buffer[row][col]+buffer[row][col+1]+buffer[row][col+2];

       x=x+buffer[row+1][col]+20*buffer[row+1][col+1]+buffer[row+1][col+2];

       x=x+buffer[row+2][col]+buffer[row+2][col+1]+buffer[row+2][col+2];
```

```
                buffer[row][col]=x;
                    }
                }
            for (row=0; row<480; row++)      /* contador de linhas */
                {
                for (col=0; col<512; col++)
                    {                        /* le arquivo de entrada */

                    x=buffer[row][col]+14;
                    putc((char)(x/28),out);
                    }
                }


            fclose(out);
            }
    }
```

7.6 FSHOW.TXT (Visual Basic)

```
Option Explicit
Dim drawpoints
Dim cont
Dim enable
Dim versel
Dim typesel



Sub Check3D1_Click (Value As Integer)
    If Check3d1.Value = True Then
        typesel = 0
    Else
        typesel = 4
        End If
Loadframe

End Sub

Sub Command1_Click ()
    drawpoints = 1
    Loadframe
End Sub

Sub Command2_Click ()
drawpoints = 1
versel = 2
Loadframe
End Sub

Sub Command3_Click ()
drawpoints = 1
versel = 3
Loadframe
End Sub

Sub Command3D1_Click ()
        Timer1.Enabled = False
        enable = 0
        cont = cont - 1
        If cont < 1 Then cont = 38
        Loadframe

End Sub

Sub Command3D2_Click ()
    Timer1.Enabled = True
    enable = 1

End Sub

Sub Command3D3_Click ()
        Timer1.Enabled = False
        enable = 0
        cont = cont + 1
        If cont > 38 Then cont = 1
        Loadframe

End Sub

Sub Command3D4_Click ()
        Timer1.Enabled = False
        enable = 0
        cont = 1
        Loadframe
```

```
End Sub

Sub Command3D5_Click ()
    Timer1.Enabled = False
    enable = 0

End Sub

Sub Command4_Click ()
drawpoints = 0
Loadframe

End Sub

Sub Command5_Click ()
    drawpoints = 2
    Loadframe
End Sub

Sub Command6_Click ()
    drawpoints = 3
    Loadframe

End Sub

Sub Command7_Click ()
drawpoints = 1
versel = 4
Loadframe

End Sub

Sub Form_Load ()
    drawpoints = 0
    currentx = frmpoints.ScaleWidth / 2
    currenty = frmpoints.ScaleHeight / 2
    cont = 0
    enable = 0
    versel = 1
    typesel = 0


End Sub

Sub Loadframe ()
        Dim s As String
        Dim pat As String
        pat = "c:\cp\working\"

        s = Format$(cont, "000")
        Select Case drawpoints
        Case 1          'Reconstruído
            s = "xap" + s
            s = s + "g.bmp"
            s = "\" + s
            s = Format$((versel + typesel), "0") + s
            s = "ver" + s
            s = pat + s
        Case 0          'Original
            s = "xa" + s
            s = pat + s
            s = s + ".bmp"
        Case 2        'Erro
            s = "xae" + s
            s = s + ".bmp"
            s = "\" + s
            s = Format$((versel + typesel), "0") + s
            s = "ver" + s
            s = pat + s
        Case 3        'Simulado
            s = "xan" + s
```

```
            s = s + ".bmp"
            s = "\" + s
            s = Format$((versel + typesel), "0") + s
            s = "ver" + s
            s = pat + s
        End Select

        Text1.Text = s
        Picture1.Picture = LoadPicture(s)

End Sub

Sub mnuarcs_Click ()
    drawpoints = 3
End Sub

Sub mnuClear_Click ()
    drawpoints = 0
    frmpoints.Cls

End Sub

Sub mnuDrawPoints_Click ()
    drawpoints = 1

End Sub

Sub mnuErro_Click ()
    drawpoints = 2
    Loadframe
End Sub

Sub mnuexit_Click ()
    End

End Sub

Sub mnuFiguras_Clic ()
End Sub

Sub mnuFiguras_Click ()
    Timer1.Enabled = True
    enable = 1
End Sub

Sub mnuFrmminus_Click ()
        Timer1.Enabled = False
        enable = 0
        cont = cont - 1
        If cont < 1 Then cont = 38
        Loadframe

End Sub

Sub mnuFrmplus_Click ()
        Timer1.Enabled = False
        enable = 0
        cont = cont + 1
        If cont > 38 Then cont = 1
        Loadframe

End Sub

Sub mnulines_Click ()
    drawpoints = 2
    currentx = frmpoints.ScaleWidth / 2
    currenty = frmpoints.ScaleHeight / 2

End Sub

Sub mnuOriginal_Click ()
```

```
    drawpoints = 0
    Loadframe
End Sub

Sub mnuPausa_Click ()
    Timer1.Enabled = False
    enable = 0
End Sub

Sub mnuReconst_Click ()
    drawpoints = 1
    Loadframe
End Sub

Sub mnusim_Click ()
    drawpoints = 3
    Loadframe

End Sub

Sub Option3D1_Click (Value As Integer)
    versel = 1
    'drawpoints = 1
    Loadframe

End Sub

Sub Option3D2_Click (Value As Integer)
    versel = 2
    'drawpoints = 1
    Loadframe

End Sub

Sub Option3D3_Click (Value As Integer)
    versel = 3
    'drawpoints = 1
    Loadframe

End Sub

Sub Option3D4_Click (Value As Integer)
    versel = 4
    'drawpoints = 1
    Loadframe

End Sub

Sub Timer1_Timer ()
        cont = cont + 1
        If cont > 38 Then cont = 1
        Loadframe
End Sub
```

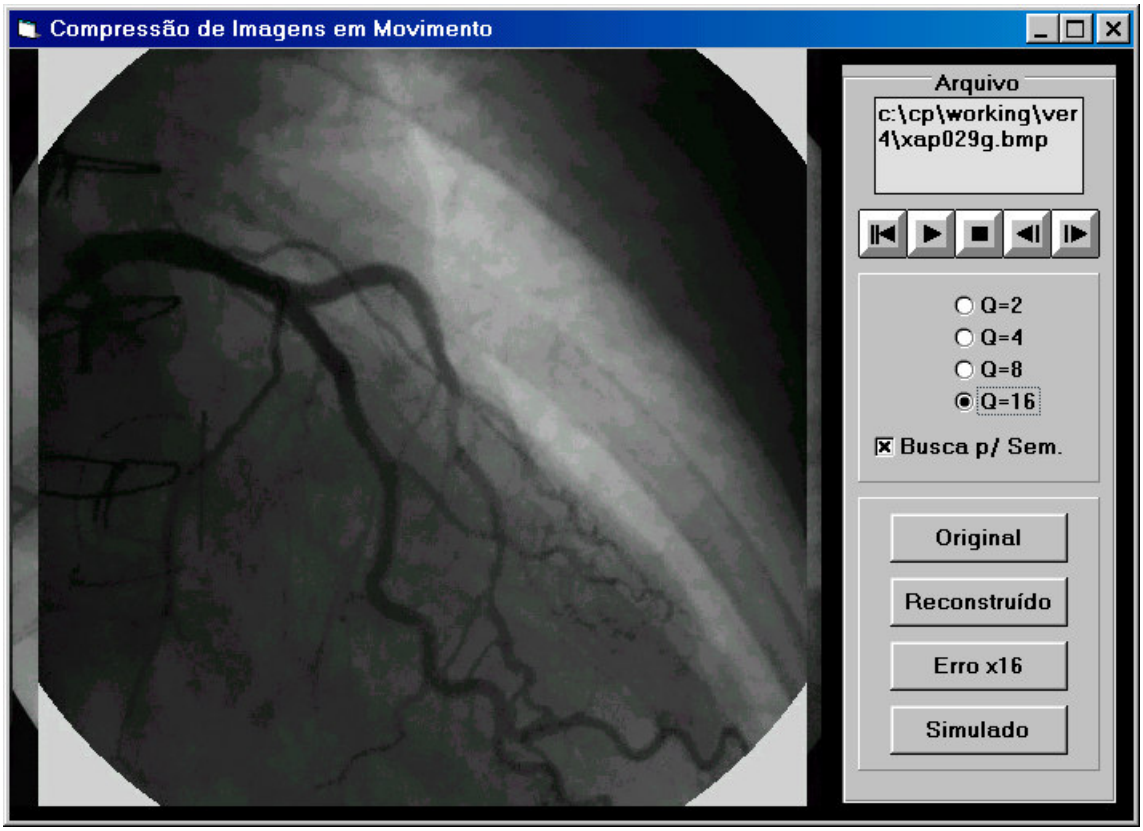


Fig. 7.6.1 - Tela do Programa FSHOW

8. Exemplos de Imagens Processadas

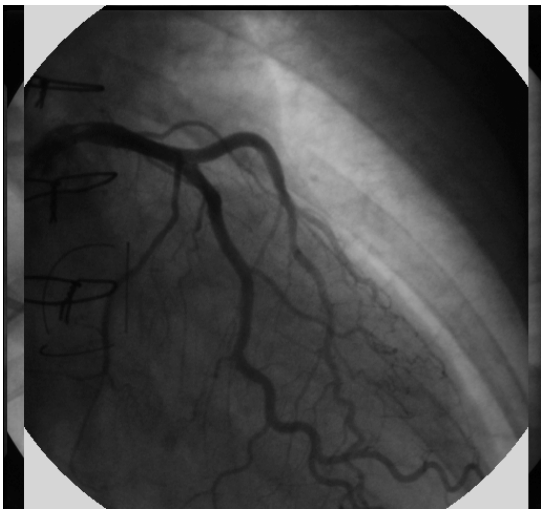


Fig. 8.1 - Imagem Original (XA030.BMP)

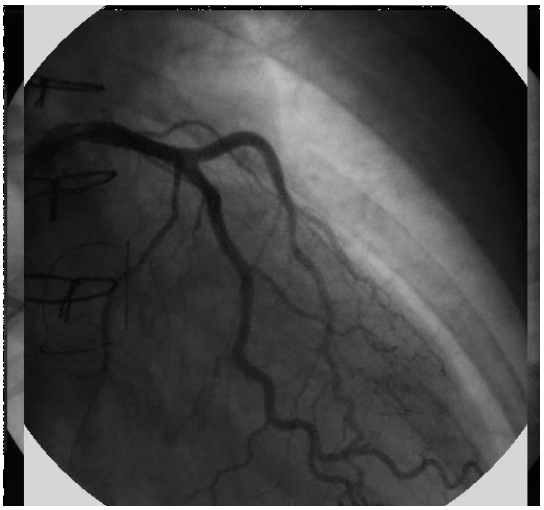


Fig. 8.2 - Imagem Reconstruída com Q=16

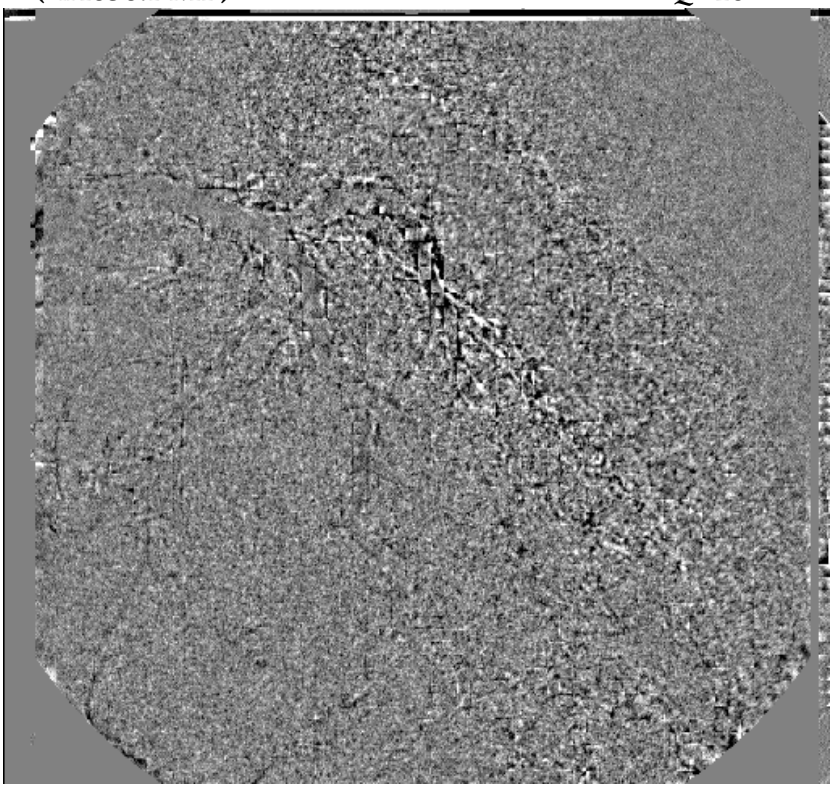
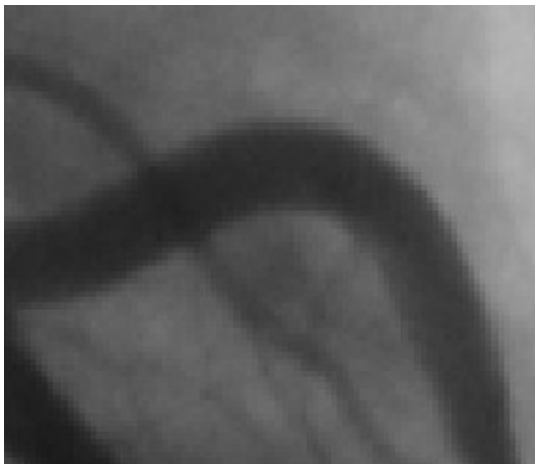
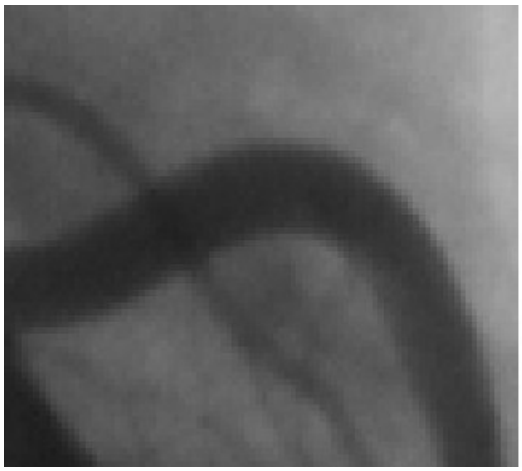


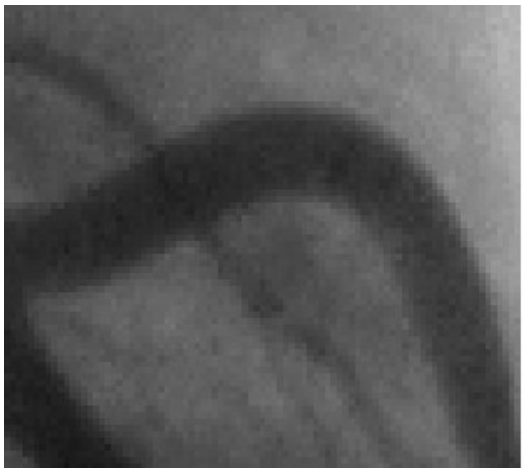
Fig. 8.3 - Erro de Predição de Movimento (ref. XA030.BMP)



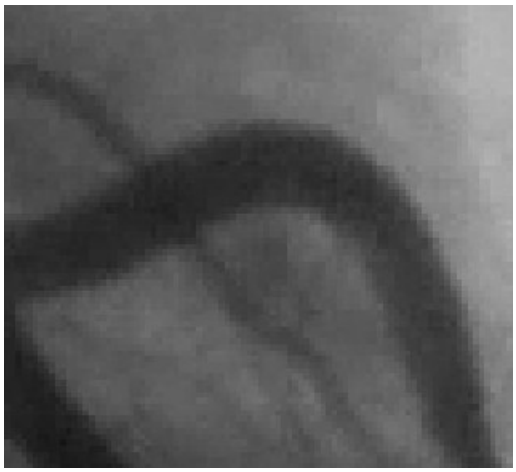
Detalhe Original



Detalhe c/ Quantizador = 2



Detalhe Q=16, Busca Normal



Detalhe Q=16, Busca p/ Semelhança

Fig. 8.4 - Detalhes Ampliados de Imagens Processadas

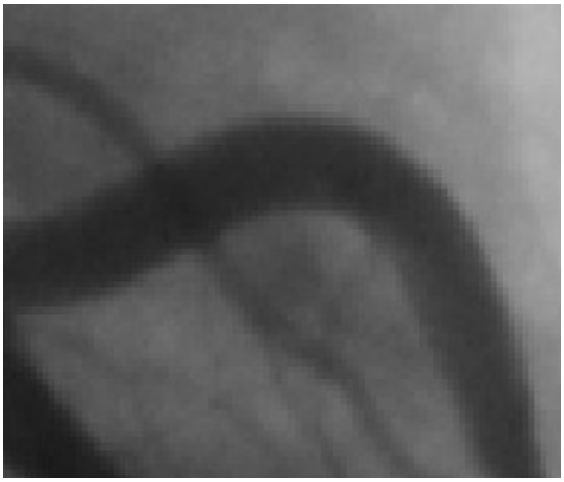


Imagem Original

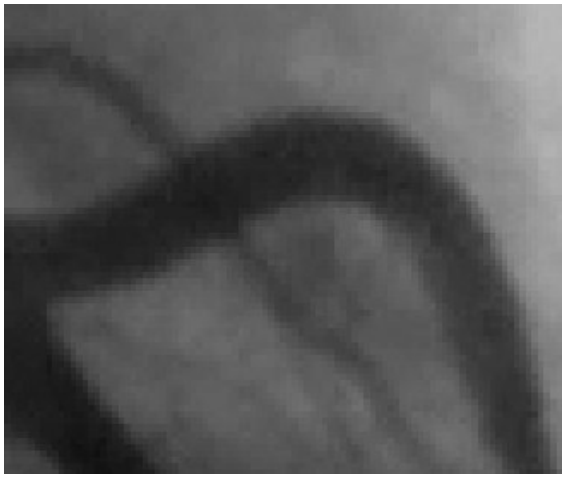
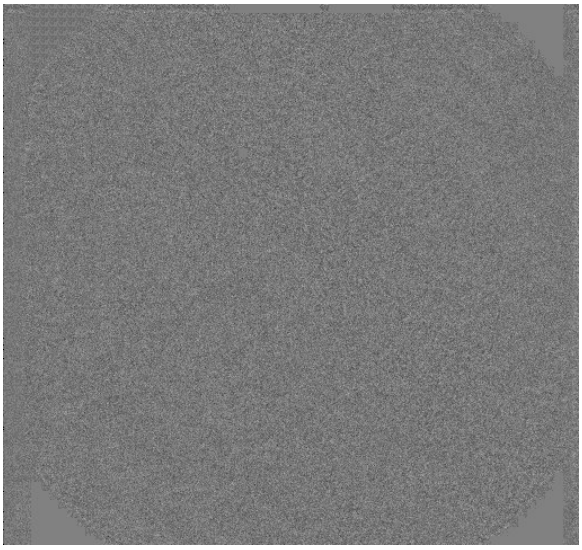
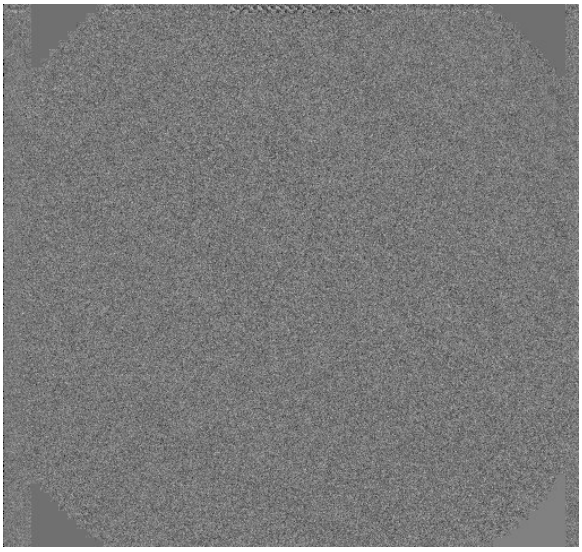


Imagem Reconstruída Equalizada

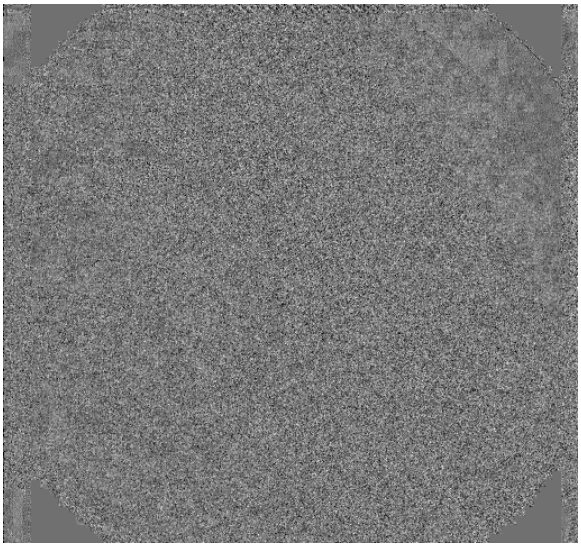
Fig. 8.5 - Detalhes de Imagem com Pós-Processamento (Q=16)



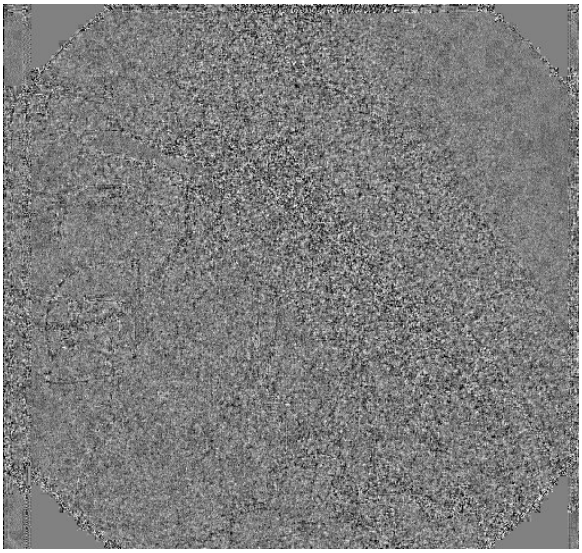
Quantizador = 2



Quantizador = 4

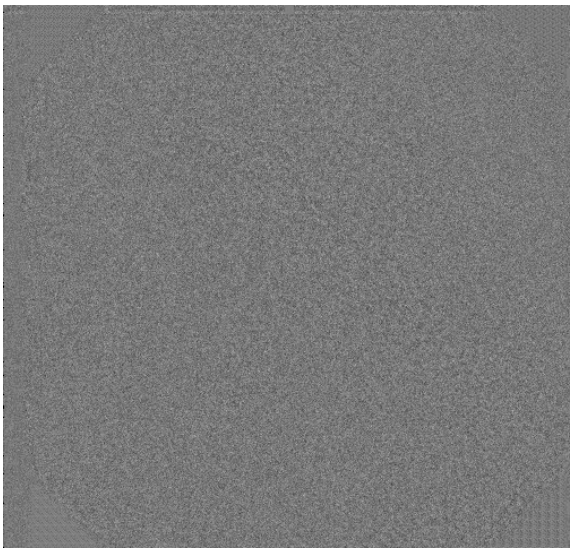


Quantizador = 8

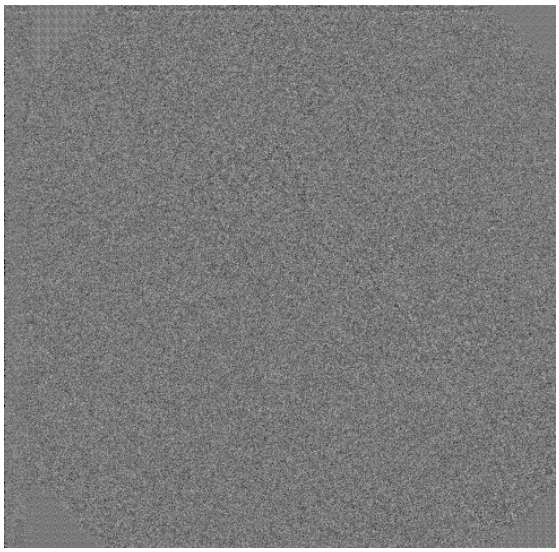


Quantizador = 16

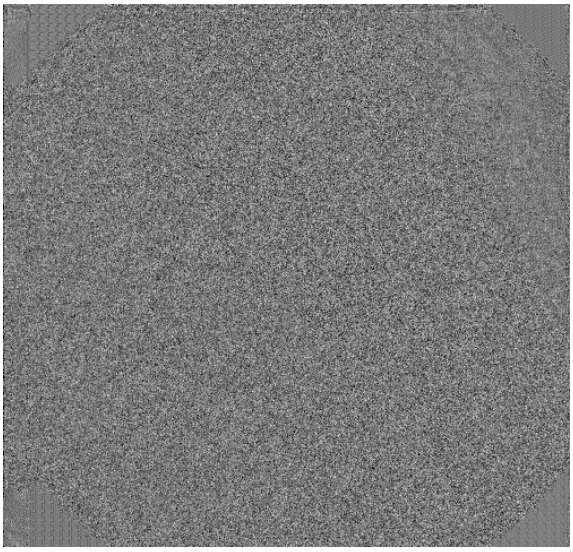
Fig. 8.6 - Erro de Reconstrução (contraste ampliado x16) com Quantizador Linear e Busca por Máxima Semelhança



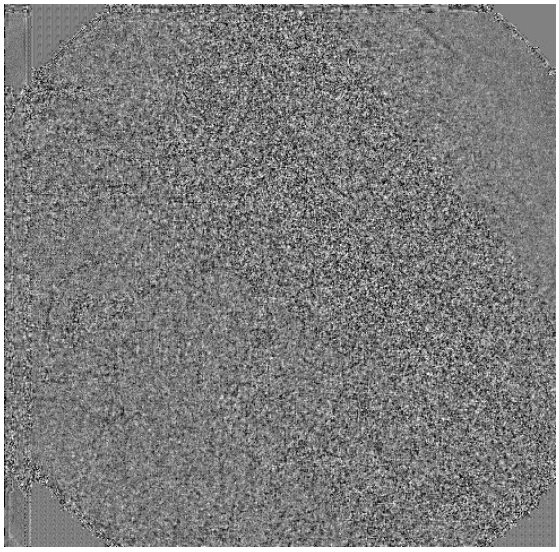
Quantizador = 2



Quantizador = 4



Quantizador = 8



Quantizador = 16

Fig. 8.7 - Erro de Reconstrução (contraste ampliado x16) com Busca Normal (Imagem No. 30 - XAE030.BMP)