UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE MATÉMATICA, ESTATÍSTICA E CIÊNCIA DA COMPUTAÇÃO
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

TÉCNICAS DE COMPRESSÃO DE SEQÜÊNCIAS DE IMAGENS VISANDO TRANSMISSÃO EM TEMPO REAL

por

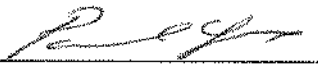
Carlos Antonio Reinaldo Costa

Maio de 1993

Técnicas de Compressão de Seqüências de Imagens Visando Transmissão em Tempo Real

Este exemplar corresponde a redação final da tese devidamente corrigida e defendida pelo Sr. Carlos Antonio Reinaldo Costa e aprovada pela Comissão Julgadora.

Campinas, 19 de Maio de 1993.



Prof. Dr. Paulo Lício de Geus

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação, UNICAMP, como requisito parcial para a obtenção do Título de MESTRE em Ciência da Computação.

Carlos Antonio Reinaldo Costa

Técnicas de Compressão de Seqüências de Imagens Visando Transmissão em Tempo Real

Dissertação apresentada em 19 de Maio de 1993.

Banca Examinadora:

Dr. Paulo Lício de Geus – DCC/UNICAMP – Orientador

Dr. Jorge Stolfi – DCC/UNICAMP

Dr. Arnaldo de Albuquerque Araujo – DCC/UFMG

Agradecimentos

" O homem magnânimo sabe como deve comportar-se quando é exaltado e quando é humilhado. Sabe mostrar temperança na sorte, seja boa ou má. Não prova nem exagera alegria num grande sucesso nem muita dor numa derrota. Não procura, mas também não evita o perigo e poucas são as coisas que o preocupam. Não é dado a falar, mas quando a ocasião o exige, diz franca e corajosamente o que sente. Não ambiciona ser louvado nem ver os outros censurados. Não se zanga por coisa de pouca monta e não conta com a ajuda de ninguém."

Aristóteles

Apesar de concordar com as palavras de Aristóteles, sou obrigado a reconhecer que não contando com ajuda de ninguém dificilmente teria concluido este trabalho. Durante a execução do mesmo contei com o apoio técnico, financeiro e emocional de várias pessoas e instituições para as quais eu gostaria de dirigir meus mais sinceros agradecimentos.

Ao meu orientador, Prof. Paulo Lício de Geus, que despertou meu interesse por esta área de pesquisa e cuja orientação foi de fundamental importância para o êxito do trabalho. Seus conhecimentos de processamento de imagens e computação em geral contibuiram para que a experiência fosse, para mim, bastante enriquecedora.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq - e à Fundação de Amparo à Pesquisa do Estado de São Paulo - FAPESP - que deram o suporte financeiro à execução do Programa de Mestrado e também à UNICAMP que através do FAEP viabilizou a parte final desta pesquisa.

Ao pessoal do projeto Vide: Marcos Aguilera, Francisco Hoyos e Fabrício de Souza pelas contribuições que deram ao desenvolvimento deste trabalho.

Também não posso deixar de lembrar dos vários colegas do mestrado que foram muito mais do que colegas: foram amigos. Meu muito obrigado a todo este pessoal com quem eu sempre soube que podia contar, seja para ajudar a resolver problemas do dia à dia, seja para uma conversa descontraída nas horas de folga. E um obrigado especial para o Visoli que por muitas vezes teve que aturar meus momentos de mal humor no período em que dividimos o mesmo apartamento.

Quero também dirigir um agradecimento muito especial, à Tina que esteve sempre ao meu lado e foi muito mais que amiga.

Minha familha, mesmo distante, também deu uma contribuição muito importante durante este período. Muito obrigado aos meus pais e a cada um dos meus irmãos pelo valioso apoio moral.

"Procurei minha instrução mais que o dinheiro,
preferi a ciência ao ouro fino,
por que a sabedoria vale mais que as pérolas
e tudo quanto há de apetecível não se lhe pode comparar."

Provérbios 8, 10-11

Aos meus pais, os principais responsáveis
por eu ter chegado até aqui.

Abstract

Advances in digital technology over the past decade have made possible the development of many telecommunication applications that employ digital video transmission. Teleconferencing and video telefony are examples of such applications. However, some specific problems appear with real-time transmission of digital images. Due to limitations imposed by the bandwidth of most local area networks, it is normaly impossible to transmit all the large amount of information needed to represent images. Therefore, it is necessary to devise a fast image compression method that is able to reach high compression rates, while still keeping a sufficiently good quality on reproduced images.

After a brief survey on the main image compression methods, attention is directed to two specific techniques: vector quantization and transform coding. These techniques are studied together with a scheme called MDPT, that combines movement detection and progressive transmission strategies. Initially, a method based on vector quantization is studied and modified in some aspects to improve results. Following that, a new method is introduced, based on the discrete cosine transform, that present several advantages over the previous scheme. For each method, a detailed description is given as well as an analysis of experimental results.

Sumário

Com os avanços na tecnologia digital durante a decada passada, tornou-se possível o desenvolvimento de inúmeras aplicações de telecomunicação que empregam a transmissão de imagens digitais de vídeo. Teleconferências e videofones são alguns exemplos deste tipo de aplicação. Entretanto, existem alguns problemas que surgem quando se trata de transmissão de imagens digitais em tempo real. Devido a limitações impostas pela largura de banda da maioria das redes locais, normalmente é impossível transmitir toda a grande quantidade de informação necessária para representar as imagens. Portanto, é necessário que se elabore um método rápido de compressão de imagens, que seja capaz de atingir altas taxas de compressão, mantendo uma qualidade suficientemente boa nas imagens reproduzidas.

Depois de uma rápida revisão sobre os principais métodos de compressão de imagens, duas técnicas específicas são abordadas neste trabalho: quantização vetorial e codificação por transformadas. Estas técnicas são estudadas juntamente com um esquema denominado MDPT que combina estratégias de detecção de movimento e transmissão progressiva. Inicialmente, um esquema baseado em quantização vetorial é examinado e modificado em alguns aspectos com o objetivo de melhorar os resultados. Em seguida, introduzimos um novo método, baseado na transformada do cosseno, que possui uma série de vantagens com respeito ao esquema anterior. Para cada método é feita uma descrição detalhada bem como uma análise dos resultados experimentais obtidos.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Quando se pensa em transmissão de seqüências de imagens digitais de vídeo, uma questão surge quase de imediato: como diminuir a grande quantidade de informação envolvida? Uma única imagem, com resolução de 480×640 pontos e 256 tons de cinza, ocupa aproximadamente 307 Kbytes de memória: para transmitir integralmente uma seqüência de vídeo com uma freqüência de 30 quadros por segundo, seria necessário manter um fluxo através de uma rede local de mais de 9 Mbytes por segundo, ou mais no caso das imagens serem coloridas e/ou de maior resolução. Isto dificulta bastante a transmissão de seqüências de vídeo em tempo real, pois a grande maioria das redes locais disponíveis não suporta um fluxo de informação desta magnitude, e sendo assim, torna-se imprescindível o uso de métodos eficientes de compressão de imagens.

Uma grande quantidade de técnicas de compressão de imagens foi desenvolvida nos últimos anos, algumas das quais, voltadas para imagens isoladas e outras para seqüências de vídeo. O objetivo central desta pesquisa é a elaboração de um esquema que combine algumas destas técnicas de modo a permitir que se atinja altas taxas de compressão com a possibilidade de codificar e decodificar as imagens em tempo real. Para elaborar tal esquema, é necessário combinar métodos de codificação que reduzam tanto a redundância espacial (dentro de um mesmo quadro) quanto a redundância temporal (entre quadros contíguos). Para redução de redundância espacial, nós voltamos nossa atenção principalmente para duas técnicas: quantização vetorial e codificação por transformadas, pois são as que melhor atendem aos nossos objetivos. Para redução de redundância temporal usamos estratégias de detecção de movimento e transmissão progressiva. A combinação destas estratégias foi inicialmente sugerida por Geus [Geu90] que elaborou um esquema baseado em quantização vetorial. Na primeira fase desta pesquisa, nós examinamos e modificamos este método em alguns aspectos com o objetivo de melhorar seus resultados. Em uma outra fase, elaboramos um novo método usando as mesmas idéias para exploração de redundância temporal, mas substituíndo a quantização vetorial por codificação por transformadas.

Nesta dissertação, antes de ser dada a descrição desses métodos, fazemos uma rápida revisão que tem o objetivo de dar ao leitor uma visão geral sobre nossa área de pesquisa, mas com um certo aprofundamento em alguns pontos mais relevantes. Esta revisão visa também introduzir todos os conceitos e notações que usamos ao longo de toda a dissertação. Não estamos supondo nenhum conhecimento anterior, por parte do leitor, quanto a codificação ou processamento de imagens; esperamos apenas um conhecimento básico de ciência da computação e noções de matemática de um nível não superior ao de um curso de cálculo diferencial e integral; conhecimentos de álgebra linear, variáveis complexas e teoria da probabilidade podem ser úteis, mas não são essenciais.

1.1 Transmissão Digital *versus* Transmissão Analógica

A informação digital tem uma série de vantagens sobre a informação analógica, entre elas, maior flexibilidade de processamento, facilidade de acesso aleatório em informação armazenada e possibilidade de transmissão livre de erro. A transmissão analógica consiste de um sinal contínuo cuja freqüência varia dentro de uma faixa normalmente chamada de largura de banda. Este tipo de transmissão é muito sujeito a ruídos e para grandes distâncias necessita de amplificadores que nunca conseguem reproduzir o sinal exatamente como o original, de modo que pode haver uma acumulação de erros que levam a uma distorção considerável. Na transmissão digital, são enviadas seqüências de 0's e 1's, e como estes são os únicos valores possíveis, a informação pode ser reproduzida exatamente como a original, o que evita a acumulação de erros.

Entretanto, há algumas limitações que são impostas à transmissão digital devido à lagura de banda dos canais de transmissão. Existe um teorema, devido a Nyquist, segundo o qual o número máximo de bits que pode ser transmitido através de um canal livre de erro com lagura de banda de b Hz é de $2b$ bits por segundo. Nyquist também provou que um sinal arbitrário que varie dentro de uma faixa de freqüência de b Hz pode ser totalmente reconstruído caso seja representado por $2b$ amostras por segundo. Com esta quantidade de amostras, se queremos transmitir um sinal de, por exemplo, 1 MHz de largura de banda, usando um esquema de modulação digital a 8 bits por amostra, então precisamos de um canal com 8 MHz de largura de banda. Este é o preço a se pagar para termos as vantagens da transmissão digital. No entanto, técnicas de compressão de dados podem ser usadas para minimizar este custo, e no caso de imagens, muitas vezes é possível reduzir a largura de banda necessária para a transmissão digital para níveis inferiores aos necessários para transmissão analógica.

1.2 Compressão de Dados e Compressão de Imagens

Quando se pensa inicialmente em compressão de imagens, pode-se imaginar que isto nada mais é do que um caso particular do que se costuma chamar de compressão, ou compactação, de dados. Entretanto existem algumas diferenças entre estes dois conceitos. Quando nos referimos a compactação de dados, a informação a ser comprimida é vista tão somente como uma seqüência de 0's e 1's que não precisam ter nenhum significado específico; normalmente esta seqüência é dividida em diferentes padrões que representam símbolos de um alfabeto finito. A grosso modo, o que a maioria destes métodos fazem é procurar repetições de tais símbolos (ou seqüências de símbolos), usando menos bits para representar os mais comuns e mais bits para representar os mais raros. Um aspecto importante destes métodos é que eles preservam informação, isto é, após a codificação e a decodificação os dados são reproduzidos exatamente como eram antes. Obviamente, estes métodos podem também ser aplicados a compressão de imagens, mas em geral as taxas de compressão tendem a ser baixas.

Para atingir resultados melhores foi criada uma outra classe de métodos voltada para imagens, onde a perda de informação é admissível. O que temos neste caso não é apenas uma seqüências de 0's e 1's mas sim uma matriz de pontos que são representados por um ou mais números que indicam uma determinada cor. Entre as inúmeras estratégias usadas por estes métodos, está a procura por padrões iguais ou semelhantes e a exploração da relação entre cada ponto e sua vizinhança. Como nestes casos pode haver perda de informação as taxas de compressão tendem a ser muito maiores.

Para determinar até que ponto a perda de informação é admissível, em geral são estabelecidos

alguns critérios de fidelidade que servem para avaliar a qualidade da imagem reproduzida. Estes critérios podem ser de dois tipos: subjetivos ou quantitativos. Critérios subjetivos são baseados na inspeção visual das imagens por parte de um grupo de observadores, que podem classificá-las em escalas globais, que vão de *insatisfatório* a *excelente*, ou escalas comparativas que vão de *pior* a *melhor*. Critérios quantitativos baseiam-se em medidas matemáticas de distorção (ver Apêndice A). Os critérios subjetivos e quantitativos nem sempre coincidem, ou seja, uma imagem que tem uma distorção maior pode ter uma qualidade visual melhor do que uma com menor distorção. Entretanto, na maioria das vezes estes critérios são bastante relacionados entre si.

1.3 Organização da Dissertação

A dissertação pode ser dividida basicamente em dois tipos de abordagem, uma teórica e outra experimental.

Na abordagem teórica, inicialmente é feita uma coletânea de diferentes métodos encontrados na literatura, e para os quais são feitas apenas descrições sucintas que dão uma visão geral de seu funcionamento. Dentre estes métodos, apenas quantização vetorial e codificação por transformadas foram descritos de maneira mais detalhada, e para estes dedicamos capítulos separados. Todos os capítulos teóricos são acompanhados no final por uma pequena seção denominada **Leituras Complementares** que contém sugestões de referências bibliográficas acompanhadas de rápidos comentários; estas seções destinam-se a leitores que desejem aprofundar seus conhecimentos sobre cada assunto.

Na abordagem experimental são feitas descrições dos métodos que foram implementados e dos resultados obtidos. Apesar dos métodos se destinarem à transmissão em tempo real, não chegamos a fazer implementações com geração de código específico para transmissão ou que fossem capazes de codificar e decodificar as imagens em tempo real, mesmo porque para isto certamente precisaríamos de implementação em *hardware*. Nossa atenção voltou-se simplesmente para as taxas de compressão e qualidade das imagens reproduzidas. Entretanto, sempre tivemos em mente a possibilidade dos métodos serem implementados para tempo real, e por isto evitamos utilizar esquemas com excessivo custo computacional. O Apêndice C dá algumas informações sobre como foram feitas as implementações e a visualização das imagens reproduzidas.

1.4 Descrição do Conteúdo

Além desta Introdução, a dissertação é dividida em mais seis capítulos, sendo que os de números 2, 3 e 5 fazem parte da abordagem teórica, os de números 4 e 6 fazem parte da abordagem experimental e o Capítulo 7 contém as conclusões gerais da dissertação.

O Capítulo 2 é uma revisão contendo explicações sucintas sobre o funcionamento dos principais métodos encontrados na literatura para compressão de imagens isoladas ou em seqüências. Neste capítulo também mostramos algumas propostas de classificação para tais métodos e dois esquemas que foram propostos como padrão para compressão de imagens para aplicações comerciais, um deles voltado para imagens isoladas e outro para seqüências de vídeo. Este capítulo tem o objetivo de dar uma visão geral sobre o que já foi realizado nesta área e de justificar a nossa preferência por quantização vetorial e codificação por transformadas na elaboração de esquemas de compressão de imagens.

O Capítulo 3 é totalmente dedicado à quantização vetorial. Inicialmente damos uma descrição formal desta técnica e mostramos um algoritmo para obtenção de quantizadores vetoriais ótimos; em seguida, enumeramos alguns dos problemas que surgem com o uso de um esquema básico de quantização vetorial e fazemos uma rápida revisão contendo diversas variações que visam minimizar estes problemas; algumas dessas variações foram usadas nos esquemas que estudamos.

O Capítulo 4 traz a descrição de esquemas baseados em quantização vetorial e a análise dos resultados obtidos. Inicialmente, fazemos alguns testes com um método para compressão de imagens isoladas; em seguida, baseando-se nos resultados obtidos com este método, modificamos alguns pontos do esquema desenvolvido por Geus para compressão de seqüências, no intuito de melhorar a qualidade das imagens reproduzidas pelo mesmo. Por fim, descrevemos um esquema de quantização vetorial classificada que foi desenvolvido com o objetivo de se adaptar ao esquema de Geus, na tentativa de atingir resultados melhores.

O Capítulo 5 é voltado para algumas transformadas matemáticas muito usadas em processamento de imagens e mostra como elas podem ser utilizadas em compressão de imagens. Iniciamos dando uma descrição geral sobre transformadas unitárias ortogonais e em seguida definimos e damos algumas propriedades importantes de três transformadas que estão entre as de maior interesse teórico ou prático para compressão de imagens. Também descrevemos um esquema básico de codificação por transformadas e citamos rapidamente algumas de suas variações. Por fim, mostramos alguns algoritmos rápidos que são de fundamental importância para que se possa utilizar transformadas em aplicações com necessidade de tempo real.

No Capítulo 6, apresentamos um novo esquema que é baseado na transformada do cosseno e em algumas das estratégias do esquema desenvolvido por Geus. O método é descrito detalhadamente e são mostrados os resultados de diferentes variações com o objetivo de demonstrar sua flexibilidade. Também é feita uma comparação deste esquema com o anterior, o que mostra uma significativa superioridade.

Por fim, o Capítulo 7 encerra com as conclusões gerais, indicando as contribuições da dissertação e dando uma lista contendo alguns possíveis temas para pesquisas futuras.

Capítulo 2

Técnicas de Compressão de Imagens

2.1 Introdução

Neste capítulo, pretendemos basicamente responder a seguinte questão: quais as principais técnicas de compressão de imagens e como elas se classificam? Isto é feito por meio de descrições simples e sucintas de cada método. Para as técnicas que são mais diretamente relacionadas com este projeto, são dadas visões mais ricas em detalhes nos capítulos seguintes. O objetivo aqui é apenas o de dar ao leitor uma visão geral sobre compressão de imagens, com abordagens introdutórias de um número significativo de métodos.

Em um sentido formal da expressão 'compressão de imagens' como sendo a redução da quantidade de informação necessária para representar imagens, o próprio processo de digitalização já é uma forma de compressão. Imagens do mundo real possuem infinitos pontos por unidade de área cada um dos quais podendo possuir uma entre infinitas cores e tonalidades, o que significa que seria preciso uma quantidade infinita de informação para representar tais imagens. Para possibilitar a representação das mesmas em computador, é preciso reduzi-las a uma quantidade finita de pontos por unidade de área, cada um dos quais possuindo uma entre um conjunto finito de cores. O processo mais comum utilizado para atingir este objetivo é conhecido como PCM (*Pulse Code Modulation*), no qual é dada uma representação discreta para a imagem, onde cada ponto, comumente chamado de *pixel*[1], é representado por um vetor binário de comprimento fixo.

O número de bits por pixel utilizado para representar as imagens resultantes deste processo é bastante variável. Normalmente, 8 bits por pixel são suficientes para produzir imagens de boa qualidade visual; entretanto, para algumas aplicações, pode ser necessário um número maior de bits como é o caso de imagens médicas que podem utilizar de 10 a 12 bits por pixel. Existem também as chamadas imagens de cores reais (*true color images*) que utilizam 24 bits por pixel, ou seja, 8 bits para cada um dos componentes de cor básicos vermelha, verde e azul. Todos os métodos que discutiremos neste trabalho referem-se, em princípio, a imagens de 8 bits por pixel em tons de cinza, mas a generalização para outros tipos de imagem como, por exemplo, imagens coloridas, não chega a ser uma tarefa difícil: bastaria separá-las em seus componentes vermelho, verde e azul ou nos componentes cor, brilho e saturação e processar cada um individualmente. Em geral, obtém-se melhores resultados com esta segunda decomposição.

[1] Palavra derivada do inglês *picture element*.

2.2 Classificação

Um dos principais problemas que surgem ao se tentar classificar os métodos de compressão de imagens é a propria definição do escopo de cada classe. Qualquer que seja a especificação das categorias, sempre haverá métodos com características em comum com mais de uma classe. Geus [Geu90] apresenta em uma abordagem bastante abrangente a seguinte classificação:

i) Técnicas de quadro e linha. São técnicas bastante simples que consistem apenas em desconsiderar partes das imagens de tal maneira que isto não cause grandes prejuizos sobre o efeito visual. Estas técnicas podem ser utilizadas mesmo antes que as imagens sejam digitalizadas. Os métodos nesta classe são:

- Entrelaçamento de linhas e pontos
- Congelamento de quadros (*frame freezing*)

ii) Técnicas básicas. São técnicas do aspecto mais geral, que não se encaixariam com perfeição em nenhuma das outras categorias. Nesta classe temos:

- PCM (*Pulse Code Modulation*)
- Codificação estatística
- *Run-length*
- *Bit-plane*
- *Quadtree*
- Subamostragem e interpolação
- Codificação por características visuais (contorno, arestas, textura)

iii) Codificação preditiva. São técnicas que levam em consideração o passado, ou seja, antes de codificar um pixel é feita uma estimativa de seu valor, baseando-se em um conjunto de pixels já codificados. A informação que é codificada neste caso, é a diferença entre a estimativa e o valor real do pixel. Nesta classe temos:

- Modulação Delta
- DPCM (*Differential Pulse Code Modulation*)
- Reconstrução condicional de quadros
- Compensação de movimento
- Quantização com predição

iv) Codificação por transformadas. São técnicas que se utilizam de transformadas matemáticas apropriadas, para dar uma nova representação aos pixels da imagem, de tal forma que possam ser codificados de maneira mais eficiente.

v) **Codificação por blocos.** O que caracteriza estas técnicas, além do fato da imagem ser dividida em blocos de tamanho fixo, é que as reconstrucões possíveis são predefinidas em um conjunto limitado de padrões, ao contrário do que acontece, por exemplo, na codificação por transformadas, que também é baseada na divisão em blocos, mas que não possui um conjunto predefinido de padrões de reconstrução. Os métodos nesta classe são:

- BTC (*Block Truncation Coding*)
- Quantização Vetorial

Existem outras abordagens que trazem classificacões diferentes. Netravali e Limb [NL80] fazem a seguinte divisão de categorias:

- PCM
- Codificação preditiva
- Codificação por transformadas
- Codificação estatística
- Outros métodos

Jain [Jai81, Jai89], por sua vez, preferiu definir as seguintes classes:

- Codificação por pixels
- Codificação preditiva
- Codificação por transformadas
- Outros métodos

Kunt *et al* [KIK85] fizeram um outro tipo de classificação, onde os métodos são divididos não por tipos, mas por taxas de compressão. Eles foram classificados em técnicas de primeira geração, que inclui metodos capazes de atingir taxas de compressão de até 10:1, e técnicas de segunda geração, que inclui métodos que podem atingir taxas superiores a 70:1.

Em nossa abordagem não seguimos, a rigor, nenhuma destas classificações, nem nos propomos a elaborar uma outra. Fazemos apenas uma divisão em dois grandes grupos que chamamos aqui de *codificação intraquadros*, para imagens isoladas, e *codificação interquadros* para seqüências de imagens.

2.3 Codificação Intraquadros

Nesta seção, tratamos de técnicas que levam em consideração apenas a redundância espacial, isto é, entre pixels de cada quadro isoladamente, não levando em consideração portanto, a redundância temporal, que é a semelhança entre quadros consecutivos de uma seqüência de imagens. Isto não significa, entretanto, que estas técnicas sejam incompatíveis com esquemas de compressão de seqüências; a maioria delas pode ser combinada com métodos que exploram redundância temporal como é visto mais adiante.

2.3.1 PCM

PCM, ou *Pulse Code Modulation*, é um processo de transformação de um sinal analógico em um sinal digital, e se divide em duas fases: amostragem e quantização.

A amostragem corresponde a dar uma representação discreta à imagem no sentido espacial, ou seja, é definida uma grade regular que divide uma imagem contínua em minúsculas regiões que dão origem ao conjunto finito de pontos, ou amostras, que representam a imagem digital. Uma vez distingüidos estes pontos, passamos para o processo de quantização, que corresponde a dar uma representação discreta no sentido de amplitude e isto consiste em nada mais do que uma função do tipo:

$$Q : R - S \tag{2.1}$$

onde R é o conjunto dos números reais e S é um subconjunto finito de R. Em outras palavras, é apenas um mapeamento de uma variável contínua numa variável discreta que é feito da seguinte maneira: são definidos n valores, chamados níveis de transição (ou decisão), que dão origem a $n+1$ intervalos disjuntos I_i, aos quais são associados outros $n+1$ valores r_i, chamados níveis de reconstrução (ver Figura 2.1). Desta forma, dado um valor $x \in R$,

$$Q(x) = r_i \text{ se } x \in I_i \ (i = 0, 1, ..., n) \tag{2.2}$$

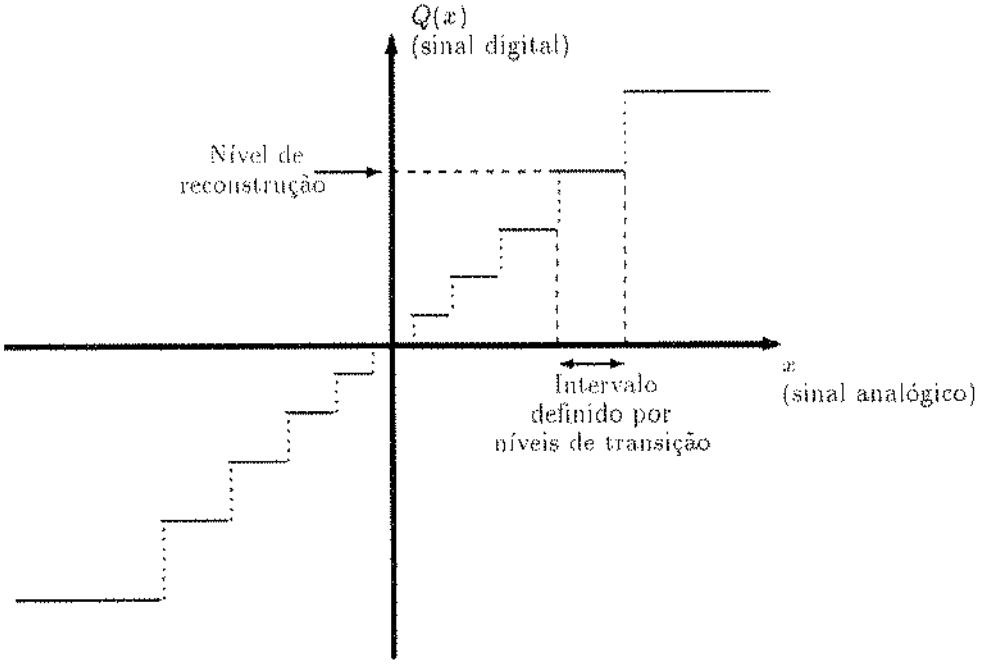


Figura 2.1: Quantização (função escada).

Na prática, um quantizador é constituído de dois mapeamentos separados, um codificador e um decodificador, que são funções do tipo:

$$C : R - M \text{ e } D : M - S \tag{2.3}$$

onde M é o conjunto das palavras de código que são associadas aos elementos de S. Desta forma Q fica definido pela composição entre D e C, isto é, $Q = D \circ C$.

Este processo evidentemente introduz alguma distorção com relação à imagem original, já que envolve perda de informação, mas existem quantizadores que tentam minimizar esta distorção, como é o caso do quantizador de Lloyd-Max, que é otimo no sentido de minimizar o erro médio quadrático (MSE) entre a imagem digitalizada e a original (ver Apêndice B).

2.3.2 Codificação Estatística

Codificação estatística (ou por entropia) representa uma coleção de métodos que são utilizados não apenas para imagens mas também em compressão de dados em geral. São métodos que não envolvem perda de informação. A idéia é tentar reduzir o máximo possível a redundância dos dados, de modo que a quantidade de informação utilizada para representá-los fique próximo de sua entropia, que é o limite teórico mínimo. Existem métodos que conseguem comprimir até bem próximo da entropia. A seguir damos uma breve descrição dos mais conhecidos:

- Código de Huffman: Neste método, que foi proposto por Huffman [Huf52], são listados todos os símbolos possíveis de um determinado tamanho, junto com as respectivas probabilidades de ocorrência. Então é construída uma árvore binária, a partir das folhas até a raíz, unindo-se sempre os dois nós de menor valor ainda não marcados, criando assim um novo nó cujo valor é a soma dos dois que lhe deram origem. Os valores iniciais das folhas são as probabilidades de ocorrência de cada símbolo. Para se construir as palavras de código que representam cada símbolo, basta descer a árvore, a partir da raiz, atribuindo-se 0's e 1's dependendo da direção que é tomada. Desta maneira, quanto maior for a probabilidade de ocorrência de um símbolo, menor será a palavra de código usada para representá-lo, resultando em um comprimento médio menor do que se fossem usadas palavras de tamanho fixo.

- Algoritmo de Lempel-Ziv: No método acima são usadas palavras de código de tamanho variável para representar símbolos de tamanho fixo; com o algoritmo de Lempel-Ziv [ZL77] ocorre o inverso: são usadas palavras de código de tamanho fixo para representar símbolos de tamanho variável que são selecionados de modo que tenham aproximadamente a mesma probabilidade de ocorrência. A forma mais usual deste algoritmo inclui uma modificação introduzida por Welch (algoritmo LZW [Wel84]), e sua idéia básica está na construção de uma tabela contendo as seqüências que já ocorreram. Durante a codificação os símbolos que vão aparecendo são concatenados um a um enquanto a seqüência resultante estiver na tabela; a maior seqüência encontrada é então codificada pelo seu endereço, e a resultante da concatenação com o símbolo seguinte é adicionada à tabela. Pela maneira como é elaborada, a tabela não precisa ser armazenada, pois pode perfeitamente ser reconstruída durante a decodificação.

- Codificação aritmética: Neste método, cada um dos símbolos que podem ocorrer é inicialmente representado por um intervalo no segmento de números reais entre 0 e 1. Estes intervalos devem ser todos disjuntos e o tamanho inicial de cada um deles é igual à probabilidade de ocorrência do símbolo a que se refere. Ao codificar-se uma seqüência, considera-se apenas o intervalo referente ao primeiro símbolo que aparece; este é então dividido de tal maneira que todos os outros intervalos sejam redefinidos em seu interior com tamanhos proporcionais aos que tinham antes; em seguida, considera-se apenas o novo intervalo referente

ao próximo símbolo que aparece e este é mais uma vez dividido proporcionalmente, e assim por diante. No final do processo, a seqüência será representada por qualquer número no interior do intervalo resultante, número tal que será descrito em forma de uma fração binária com precisão tão maior quanto menor for o intervalo. Para se decodificar, basta repetir o processo de divisão de intervalos, verificando-se em cada passo qual o símbolo referente ao intervalo onde o número chave se encontra (maiores detalhes sobre este método podem ser encontrados, por exemplo, em [RL79] ou [Lan84]).

Devido ao fato destes métodos não permitirem perda de informação, as taxas de compressão atingidas por eles dificilmente estão muito acima de 2:1 para imagens do mundo real, portanto, estes métodos não são os mais adequados quando se precisa atingir taxas mais elevadas de compressão. Isto não impede, contudo, que esses métodos estatísticos sejam utilizados em cascata para comprimir os dados resultantes de outros métodos, melhorando assim a eficiência global da compressão.

2.3.3 *Run-length* e *Bit-plane*

Run-length é uma técnica bastante simples que é mais freqüentemente utilizada na codificação de imagens de dois tons (1 bit por pixel) onde são esperadas longas seqüências de 0's e 1's consecutivos como é o caso de mapas, documentos, gráficos, entre outros. Neste caso, o método funciona da seguinte maneira: a imagem é percorrida linha a linha e são contadas as ocorrências consecutivas de 0's e 1's, codificando-se então os comprimentos das sucessões encontradas. Por exemplo, suponha que queiramos codificar uma linha da forma $< 0000000111100000 >$, o código resultante neste caso é $< 7,4,5 >$.

Este método pode facilmente ser generalizado para imagens de mais de um bit por pixel. Neste caso, junto com o comprimento das sucessões devem ser codificados também, os respectivos valores de pixel. Uma outra técnica existente para imagens de profundidade[2] maior do que 1, é fazer uma divisão da imagem em planos de um bit por pixel e codificar cada um deles separadamente. Uma imagem de 256 tons de cinza, por exemplo, pode ser dividida em 8 planos de profundidade 1 que são posteriormente codificados por *run-length* como descrito no parágrafo anterior. Esta técnica é conhecida como *bit-plane* [SB66].

As taxas de compressão atingidas por estes métodos geralmente estão entre 1,5:1 e 2:1, e podem ser bem inferiores se a imagem for muito ruidosa. Pode-se atingir índices maiores se admitirmos perda de informação e desprezarmos pequenas variações de um pixel para outro, mas mesmo assim, as taxas ainda não são muito altas, o que faz com que este método não seja o mais recomendável no caso de se pretender desenvolver um esquema que tenha o objetivo de atingir taxas elevadas de compressão.

2.3.4 *Quadtree*

Este método consiste em fazer sucessivas divisões na imagem até que cada região obedeça a uma determinada propriedade, gerando como resultado uma estrutura em forma de árvore onde cada folha representa uma região [Sam84]. Como ilustração, observe o esquema da Figura 2.2.

Em (a) temos uma região a ser codificada; esta região é dividida em quatro quadrantes que são divididos novamente se neles houver pixels de mais de uma cor. As divisões continuam até que todas

[2] Profundidade aqui significa o número de bits por pixel

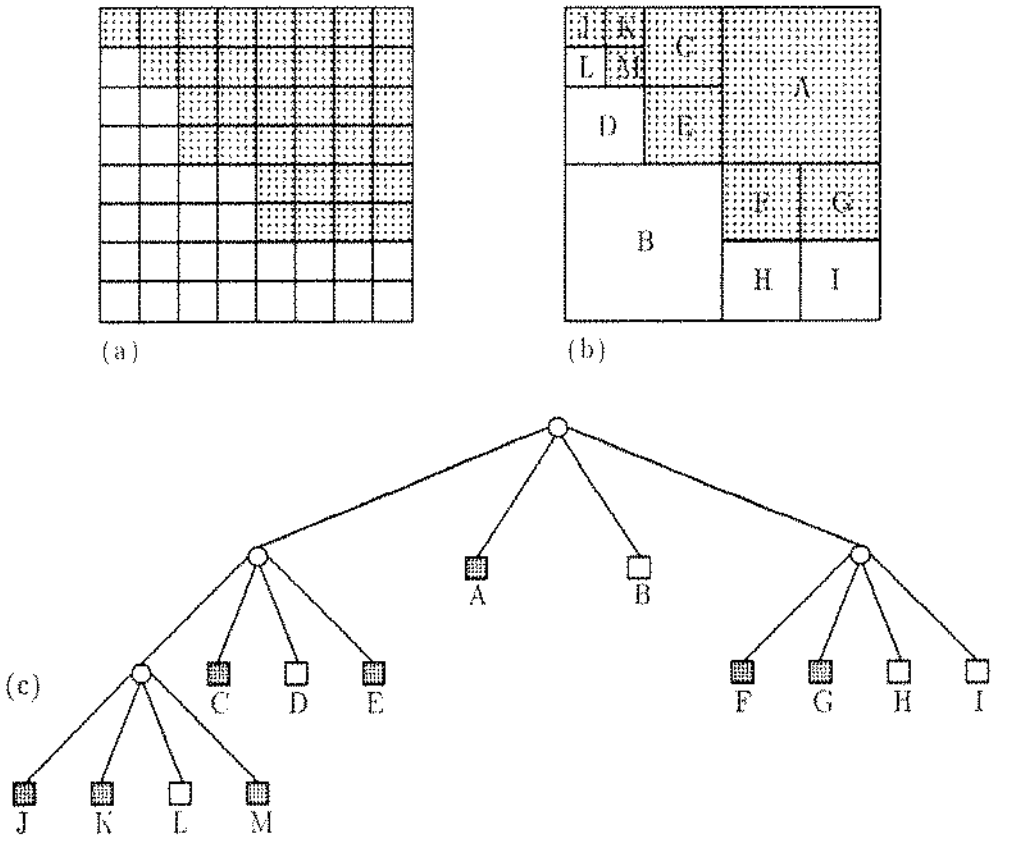


Figura 2.2: *Quadtree*.

as regiões contenham pixels de uma única cor. Como resultado deste processo temos as regiões que aparecem em (b) e que são representadas pelas folhas da árvore (c). Os nós intermediários (círculos) representam regiões mistas, isto é, que possuem pixels de mais uma cor.

Este método, como o *run-length*, também é mais freqüentemente utilizado em imagens de 2 tons, mas podemos perfeitamente utilizá-lo em imagens com profundidade maior que 1, conseguindo resultados melhores que os obtidos com *run-length*, apesar de ainda não ser este o método mais recomendável para se atingir altas taxas de compressão.

2.3.5 Subamostragem e Interpolação

Este método consiste em escolher um subconjunto de pixels para representar a imagem e fazer uma aproximação dos demais por interpolação. O subconjunto de pixels pode ser escolhido de várias maneiras, como por exemplo: um pixel a cada bloco de quatro, linhas alternadas, quadros alternados (no caso de seqüências), um pixel em cada dois, etc. A taxa de compressão atingida e a qualidade da imagem resultante dependem desta escolha e do método de interpolação utilizado. O tipo de interpolação que é usado com maior freqüência é a interpolação polinomial, que como o próprio nome já diz, faz uma aproximação utilizando curvas resultantes de polinômios. É de se esperar que quanto maior seja o grau destes polinômios, maior seja a qualidade da imagem

reproduzida, entretanto, o custo de processamento cresce consideravelmente com aumento do grau, e muitas vezes o ganho em qualidade não é tão significativo para justificar a escolha de polinômios de ordem superior.

Um tipo de interpolação muito utilizado é o linear, que é baseada em polinômios de grau 1 (retas). Vejamos, por exemplo, a Figura 2.3: para cada bloco de 2×2 em (a) é escolhido um único pixel, dando origem ao padrão mostrado em (b); em seguida é feita a interpolação linear, primeiro nas linhas ((b)→(c)) e depois nas colunas ((c)→(d)).

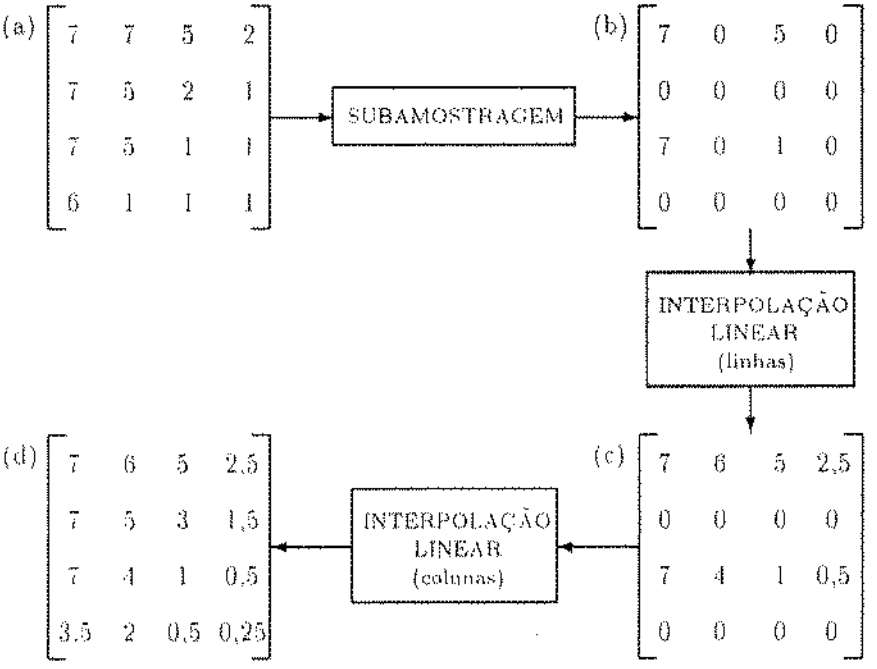


Figura 2.3: Subamostragem e interpolação.

Esta é uma das variações mais simples de subamostragem e interpolação e atinge uma taxa de compressão de 4:1, mantendo muitas vezes uma qualidade aceitável na imagem reproduzida. Métodos mais sofisticados, que utilizem interpolações de ordem superior, podem atingir resultados melhores, mas dificilmente se consegue manter uma boa qualidade na imagem reproduzida com taxas de compressão superiores a 4:1.

2.3.6 Codificação por Características Visuais

A idéia deste tipo de método[3] é a de tentar fazer uma descrição baseada em aspectos visuais da imagem, em outras palavras, representar no computador aquilo que o olho humano é capaz de perceber. Estes métodos se dividem em três classes: codificação por contornos de luminância, codificação por arestas e codificação por textura.

[3] Chamado em inglês de *feature coding*

Codificação por contornos de luminância. Basicamente, consiste em descrever a imagem como uma espécie de mapa topográfico onde as curvas de nível, ao invés de representar uma determinada altura, representam um certo valor de brilho. Estas curvas seriam as fronteiras entre os quais teríamos regiões de brilho constante. O maior problema com este tipo de método, é que em imagens de mundo real os contornos de luminância quase sempre não são bem definidos, além do fato de sempre haver algum ruído na aquisição das mesmas, que destrói estes contornos. Portanto, para possibilitar a utilização desta técnica, é imprescindível a utilização de um filtro para extração de ruído. Um exemplo de filtro eficiente é o filtro sigma [Lee83], que remove o ruído sem afetar as arestas da imagem. As taxas de compressão atingidas com este método estão em torno de 2:1, que são relativamente baixas para um método desta complexidade.

Codificação por arestas. Neste método, a imagem é separada em duas partes: as arestas e todo o resto. As arestas compõem os chamados componentes de alta freqüência espacial da imagem e o restante são os componentes de baixa freqüência. A idéia por traz disto, é que se for possível reproduzir bem as arestas, toda a imagem poderá ser reproduzida com qualidade. As arestas geralmente são representadas por um código binário, o que permite que se utilizem algoritmos destinados a imagens de dois tons. Para os componentes de baixa freqüência, podem ser utilizados alguns métodos que têm melhor desempenho na ausência de arestas como é o caso da subamostragem e, como é visto mais adiante, codificação por transformadas. É possível atingir altas taxas de compressão por este método, algumas variações conseguem atingir até 16:1 ou mais. Entretanto o seu custo computacional torna difícil o seu uso em esquemas que têm necessidade de tempo real.

Codificação por textura. Como no método anterior, aqui também as arestas são detectadas e codificadas separadamente, desta forma a imagem fica dividida em regiões com texturas suaves, i.e. com baixa freqüência espacial. Estas texturas são então modeladas de alguma maneira, em geral por funções polinomiais bidimensionais que dão uma descrição aproximada da forma da distribuição de brilho em cada região. As taxas de compressão atingidas por este método também podem ser relativamente altas, mas com um elevado custo computacional, assim como nos demais métodos baseados em características visuais da imagem, e isto dificulta sua utilização em aplicações de tempo real.

2.3.7 DPCM

O princípio básico do DPCM (*Differential Pulse Code Modulation*), como dos demais métodos que utilizam predição, é fazer uma estimativa do valor de cada pixel, baseada em outros já processados e codificar apenas o erro de predição. O método conta fundamentalmente com dois elementos: uma regra de predição e um quantizador. A regra de predição é uma função de um ou mais pixels já codificados, que podem estar todos na mesma linha (DPCM unidimensional), em linhas anteriores (DPCM bidimensional) ou até mesmo em quadros anteriores, se estivermos lidando com seqüências de imagens. Uma vez feita a estimativa calcula-se seu erro, que serve de entrada para o quantizador; o decodificador, por sua vez, soma o valor quantizado de erro à estimativa para obter o valor do pixel reconstruído. Matematicamente falando, o método funciona da seguinte maneira: Para cada amostra $u(m,n)$ é associado um conjunto $S_{m,n}$ que define janelas de pixels já codificados; então definimos uma regra de predição Ψ e um quantizador Q, de forma que a estimativa do pixel na

posição (m, n) é dada por:

$$\tilde{u}(m,n) = \Psi(S_{m,n}) \tag{2.4}$$

e o erro de predição:

$$e(m,n) = u(m,n) - \tilde{u}(m,n) \tag{2.5}$$

$e(m,n)$ é então quantizado e armazenado ou transmitido. O decodificador então utiliza Ψ para fazer a mesma estimativa $\tilde{u}(m,n)$ de $u(m,n)$; em seguida, determina-se o valor do pixel reconstruído como sendo:

$$u^*(m,n) = \tilde{u}(m,n) + e^*(m,n) \tag{2.6}$$

onde $e^*(m,n) = Q(e(m,n))$ é o valor quantizado do erro de predição.

Geralmente são usadas funções lineares como regras de predição devido a sua simplicidade; neste caso, os conjuntos S referidos acima normalmente têm entre um e quatro pixels . Experimentos mostram que há uma melhora significativa quando o número de pixels utilizado para predição é aumentado de um para dois ou de dois para três, mas a partir daí o ganho em qualidade tende a ser pequeno [Hab71].

Para facilitar o entendimento, vamos analisar um exemplo de um caso bem simples de DPCM unidimensional. Considere a seqüência 100, 102, 100, 120, 118, 118, 120, 120. Vamos utilizar como regra de predição a função identidade sobre o pixel anterior; desta maneira a expressão 2.4 fica:

$$\tilde{u}(n) = u^*(n-1) \tag{2.7}$$

e o erro de predição é portanto:

$$e(n) = u(n) - u^*(n-1) \tag{2.8}$$

utilizamos um quantizador de quatro níveis (2 bits) definido da seguinte maneira:

$$Q(x) = \begin{cases} 5 & \text{se} \quad 2 < x \\ 1 & \text{se} \quad 0 < x \leq 2 \\ -1 & \text{se} \quad -2 < x \leq 0 \\ -5 & \text{se} \quad x \leq -2 \end{cases} \tag{2.9}$$

Os resultados são mostrados na Tabela 2.1. Observe que existe uma aresta abrupta entre as posições 2 e 3 de $u(n)$ que aparece suavizada ao longo dos pixels 2 a 6 de $u^*(n)$. Isto é um problema que pode acontecer quando utilizamos quantizadores de poucos níveis. Em regiões constantes, há um outro efeito que pode ocorrer: o leitor que tiver a curiosidade de continuar o processo da Tabela 2.1 acrescentando uma seqüência constante de 120 a partir da oitava posição de $u(n)$, observará que os valores de $u^*(n)$ irão se alternar entre 120 e 119. Este efeito é chamado de granularidade.

Problemas como estes podem ser minimizados se usarmos uma regra de predição mais sofisticada como no caso bidimensional. Um exemplo de escolha para os conjuntos $S_{m,n}$ é mostrado na Figura 2.4.

Com este conjunto podemos elaborar uma regra de predição da forma:

$$\begin{aligned} \Psi(S_{m,n}) = {} & a_1 u^*(m-1,n) + a_2 u^*(m,n-1) + \\ & a_3 u^*(m-1,n-1) + a_4 u^*(m+1,n-1) \end{aligned} \tag{2.10}$$

n	$u(n)$	$\ddot{u}(n)$	$e(n)$	$e^*(n)$	$u^*(n)$
0	100	-	-	-	100
1	102	100	2	1	101
2	100	101	−1	−1	100
3	120	100	20	5	105
4	118	105	13	5	110
5	118	110	8	5	115
6	120	115	3	5	120
7	120	120	0	−1	119

Tabela 2.1: Exemplo de DPCM unidimensional.

$m-1, n-1$	$m, n-1$	$m+1, n-1$
$m-1, n$	X	-

Figura 2.4: Conjunto de pixels usado para predição em DPCM bi-dimensional

onde a_1, a_2, a_3 e a_4 são constantes convenientemente calculadas [Jai81]. Uma escolha adequada para os valores destas constantes seria por exemplo:

$$a_1 = a_2 = 0,95 \quad a_3 = -a_1 a_2 \quad a_4 = 0 \tag{2.11}$$

A qualidade da imagem reproduzida depende de quão bons são a regra de predição e o quantizador utilizados. A vantagem deste método sobre o PCM é que pode-se utilizar um quantizador com uma menor quantidade de níveis mantendo o mesmo padrão de qualidade, resultando em uma menor quantidade de bits por pixel. Observe por exemplo a Tabela 2.1, utilizando PCM os valores a serem quantizados seriam os da coluna $u(n)$, enquanto que por DPCM seriam quantizados os valores da coluna $e(n)$ que são significativamente menores. As taxas de compressão atingidas por este método não são muito altas mas a importância do DPCM reside no fato de que, por sua simplicidade, ele requer um pequeno poder de computação e pode facilmente ser utilizado em conjunto com outros métodos mais complexos.

2.3.8 Modulação Delta

Modulação Delta é o mais simples entre os métodos que utilizam predição, e é muito semelhante ao exemplo de DPCM visto na Seção 2.3.7. A estimativa e seu erro são dados pelas expressões 2.7 e 2.8, a diferença é que o quantizador aqui, tem apenas dois níveis (1 bit por pixel); se o erro de predição for positivo é gerado um pulso positivo, caso contrario, é gerado um pulso negativo. Desta maneira a imagem é representada apenas por uma seqüência binária, ou seja, com uma compressão de 8:1. Entretanto a qualidade da imagem reproduzida é um tanto pobre, os problemas citados na Seção 2.3.7, borramento de arestas e granularidade, são ainda mais evidentes aqui. A Figura 2.5 ilustra como isto acontece.

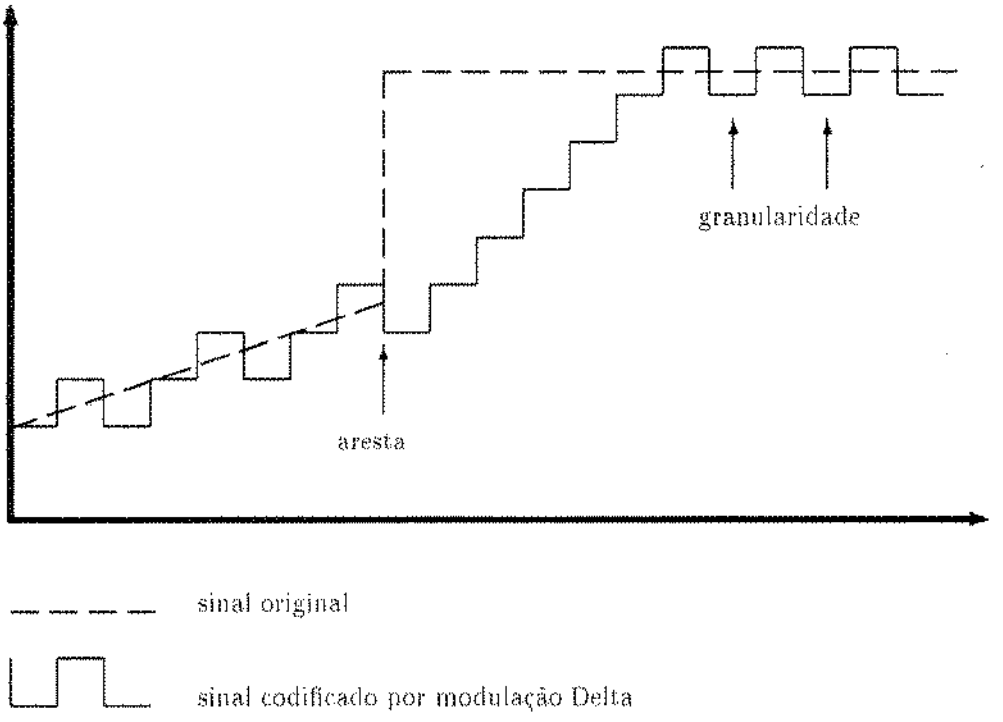


Figura 2.5: Modulação Delta.

O uso de um filtro passa-baixas[4] pode reduzir o efeito de granularidade. Existem também variações, como uma chamada *tri-state deltamodulator*, que além dos estados positivo e negativo, tem também um estado zero para regiões constantes, o que evita a granularidade. Entretanto, a qualidade da imagem reproduzida ainda é baixa.

O leitor já deve ter percebido que modulação Delta nada mais é do que um caso particular de DPCM, e tem como sua maior vantagem uma extrema simplicidade, no entanto, este método não é muito recomendável se tivermos uma maior exigência quanto à qualidade visual.

[4] Em inglês *low-pass filter*, tipo de filtro que extrai componentes de alta freqüência espacial.

2.3.9 Codificação por Transformadas

Existem algumas transformadas matemáticas que são muito freqüentemente utilizadas em processamento de imagens, devido à propriedade que possuem de separar a imagem em componentes de freqüência espacial. Esta propriedade facilita, por exemplo, a construção de filtros para detecção de arestas (passa-altas) e remoção de ruído (passa-baixas). Mas existe uma outra característica bastante interessante nestas transformadas: devido às características das imagens do mundo real, elas tendem a concentrar a maior parte da energia das imagens em uma pequena quantidade de coeficientes. Daí vem a sua aplicação em compressão de imagens. Fazendo-se uma alocação seletiva de bits, isto é, tanto menos bits quanto menor for a concentração de energia em cada coeficiente, pode-se obter boas taxas de compressão mantendo uma boa qualidade da imagem reproduzida.

A transformada mais usada em compressão de imagens é a do cosseno, ou DCT (*Discrete Cosine Transform*), pois, entre as que possuem algoritmo rápido, é a que, teoricamente, tem o melhor desempenho quanto à concentração de energia.

Este é um dos mais importantes métodos de compressão de imagens e voltaremos a falar sobre ele mais adiante. O Capítulo 5 é totalmente dedicado a transformadas, e o Capítulo 6 traz um esquema de compressão de seqüências de imagens baseado em DCT e alguns resultados experimentais.

2.3.10 BTC

BTC (*Block Truncation Coding*) é um método relativamente simples: consiste basicamente em dividir a imagem em pequenos blocos retangulares e codificar cada um utilizando um quantizador de 1 bit (2 níveis), que seja adaptativo de modo a preservar determinadas propriedades de cada bloco. A princípio qualquer critério poderia ser adotado quanto a estas propriedades; uma escolha que tem mostrado bons resultados é manter algumas medidas estatísticas dos blocos. Vejamos por exemplo uma abordagem que preserva o primeiro e segundo momentos de cada amostra.

Consideremos que a imagem seja dividida em blocos de $n \times n$, e sejam $x_1, x_2, ..., x_m$, onde $m = n^2$, os valores de pixel de um certo bloco. O primeiro e segundo momentos da amostra são dados por:

$$\overline{X} = \frac{1}{m} \sum_{i=1}^{m} x_i \tag{2.12}$$

$$\overline{X^2} = \frac{1}{m} \sum_{i=1}^{m} x_i^2 \tag{2.13}$$

e a variância:

$$\sigma^2 = \overline{X^2} - \overline{X}^2 \tag{2.14}$$

É preciso adaptar o nível de decisão X_d e os dois níveis de recontrução a e b do quantizador, de tal maneira que $\overline{X}$ e $\overline{X^2}$ sejam preservados. Um valor conveniente (mas não ótimo) para X_d seria o próprio $\overline{X}$. Podemos determinar a e b resolvendo o seguinte sistema de equações:

$$\begin{cases} m\overline{X} = (m-q)a + qb \\ m\overline{X^2} = (m-q)a^2 + qb^2 \end{cases} \tag{2.15}$$

onde q é o número de pixels no bloco com valor maior que X_d. A solução do sistema é dada por:

$$a = \overline{X} - \sigma\sqrt{\frac{q}{m-q}} \qquad b = \overline{X} + \sigma\sqrt{\frac{m-q}{q}} \tag{2.16}$$

Cada bloco é descrito por $\overline{X}$ e σ (média e desvio padrão) e um bitmap onde os 0's e 1's indicam quais pixels tem seus valores abaixo ou acima de X_q. A Figura 2.6 mostra um exemplo deste tipo de abordagem.

bloco original: $X = \begin{bmatrix} 121 & 114 & 56 & 47 \\ 37 & 200 & 247 & 255 \\ 16 & 0 & 12 & 169 \\ 43 & 5 & 7 & 251 \end{bmatrix}$

código gerado: $\overline{X} = 94,75$, $\sigma = 92,95$ $\quad \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

bloco reconstruído: $X^* = \begin{bmatrix} 204 & 204 & 17 & 17 \\ 17 & 204 & 204 & 204 \\ 17 & 17 & 17 & 204 \\ 17 & 17 & 17 & 204 \end{bmatrix}$

Figura 2.6: Exemplo de BTC

Utilizando 8 bits para cada um dos valores de média e desvio padrão e blocos de 4×4, temos uma representação da imagem a 2 bits/pixel, ou seja, uma compressão de 4:1. Para aumentar a taxa de compressão seria preciso aumentar o tamanho dos blocos, mas a qualidade da imagem reproduzida cai sensivelmente com o uso de blocos maiores. Delp e Mitchell [DM79] apresentam algumas variações que produzem resultados melhores, mas sempre usando blocos pequenos, com taxas de compressão entre 4:1 e 5:1. Um melhor desempenho pode ser conseguido ao se combinar BTC com outros métodos, mas não chegamos a discutir estas variações aqui.

2.3.11 Quantização Vetorial

O tipo de quantização a que nos referimos quando falamos de técnicas como PCM e DPCM, pode ser chamado de escalar, pois faz mapeamentos em apenas uma dimensão. Quantização vetorial é uma espécie de generalização deste processo para mais de uma dimensão, isto é, são mapeados vetores ao invés de números. Na quantização escalar, para cada valor de entrada é associado um valor de saída escolhido em conjunto finito e predefinido de números. Na quantização vetorial, a imagem é dividida em pequenos blocos de tamanho fixo, e cada um deles é comparado com um conjunto predefinido de padrões. Os blocos são codificados pelo endereço do padrão com o qual houve o melhor casamento.

Damos uma descrição mais detalhada deste método no Capítulo 3 e mostramos alguns resultados experimentais obtidos com esquemas que o utilizam no Capítulo 4.

2.4 Codificação Interquadros

Na Seção 2.3 estudamos métodos que exploram apenas a redundância espacial. Nesta seção, vamos examinar algumas técnicas que levam em consideração um outro tipo de redundância, que surge quando estamos lidando com seqüências de imagens: a redundância temporal, ou seja, a semelhança que existe entre quadros contíguos. Combinando estas técnicas com as que já vimos, podemos aumentar consideravelmente as taxas de compressão atingidas.

2.4.1 Congelamento de Quadros e Entrelaçamento de Linhas

Talvez a primeira idéia que venha à mente quando se pensa em comprimir uma seqüência de imagens de vídeo seja reduzir o número de quadros por segundo, simplesmente descartando alguns deles. Entretanto existe um limite, abaixo do qual começa a surgir um efeito de tremulação[5] na imagem que é bastante incômodo para o observador. Este limite depende da persistência da luminosidade no fósforo das telas de televisão ou monitores, bem como na retina do olho humano, e está em torno de 50 a 60 quadros por segundo [Jai89]. Pode-se, no entanto, atingir uma compressão de 2:1 transmitindo (ou armazenando) apenas 30 quadros por segundo, e repetindo duas vezes cada uma de tal forma que sejam mostrados 60 quadros por segundo. Esta técnica, que é conhecida como congelamento ou repetição de quadros, evita a tremulação, mas há um outro problema que pode ocorrer: a falta de continuidade no movimento. É como se fosse realizado por meio de pequenos saltos. Para evitar este efeito é comum se utilizar o entrelaçamento de linhas, isto é, cada quadro é dividido em dois campos que são transmitidos alternadamente, um com as linhas nas posições pares e outro com as linhas nas posições ímpares. Com esta técnica se mantém a sensação de continuidade no movimento, mas pode haver alguma degradação de arestas, já que as imagens estão sendo subamostradas.

A idéia de entrelaçamento pode ser utilizada não apenas entre linhas mas também entre pontos de uma mesma linha, chamado neste caso de entrelaçamento de pontos. Aplicando-se as duas estratégias simultaneamente é possível criar um esquema que mantem a continuidade do movimento com uma menor degradação das arestas [Geu90].

2.4.2 Reconstrução Condicional

Esta técnica é baseada na detecção e codificação de áreas em movimento, que são reconstruídas de quadro para quadro. Áreas onde não há detecção de movimento não são processadas, são simplesmente copiadas do quadro anterior. O método básico se desenvolve da seguinte maneira: seja $u(m, n, i)$ o valor de brilho do pixel na posição (m, n) do i-ésimo quadro. Calcula-se a diferença de sinal entre quadros consecutivos, dada por:

$$\epsilon(m, n, i) = u(m, n, i) - u^*(m, n, i-1) \tag{2.17}$$

onde u^* representa a imagem reconstruída; define-se um limiar η conveniente de modo que o valor de brilho do pixel (m, n) do i-ésimo quadro reconstruído é dado por:

$$u^*(m, n, i) = \begin{cases} u^*(m, n, i-1) + \epsilon^*(m, n, i) & \text{se } |\epsilon(m, n, i)| > \eta \\ u^*(m, n, i-1) & \text{caso contrário} \end{cases} \tag{2.18}$$

[5] Em inglês *flickering*.

onde e^* representa o valor quantizado de e.

Existe uma variação desta técnica, chamada reconstrução condicional de blocos, que é muito similar à idéia original, a diferença é que ao invés de manipular cada pixel separadamente, tanto o mecanismo de detecção de movimento quanto o de atualização, lidam com blocos de tamanho e forma predefinidos. A forma mais prática é a quadrada, com tamanhos que variam entre 4×4 e 16×16 pixels.

Usando-se unicamente este método, reconstruindo cada bloco diretamente por PCM, é possível atingir taxas de compressão entre 2:1 e 5:1 [Geu90], que já é bastante significativo, levando-se em conta a simplicidade do método. Uma das maiores vantagens desta técnica é que ela pode ser facilmente combinada com outros métodos mais sofisticados, permitindo atingir taxas bastante elevadas de compressão.

2.4.3 Compensação de Movimento

A idéia por traz deste método é que se a trajetória de movimento de cada pixel puder ser medida, então apenas o quadro inicial e a informação de deslocamento de um quadro para outro devem ser codificados. Para reproduzir as imagens temos simplesmente que propagar cada pixel ao longo de sua trajetória. O movimento de objetos em uma cena pode ser aproximado por vetores de deslocamento de um quadro para outro. O mecanismo básico desta técnica é o seguinte: dada uma região da imagem no quadro corrente procura-se uma região de configuração similar no quadro anterior; uma vez encontrada a região com que houve o melhor casamento, ela é subtraída pixel a pixel da região original no quadro corrente; o sinal de erro é então codificado juntamente com o vetor de deslocamento.

O sucesso deste método depende principalmente do algoritmo de estimativa de movimento. Em uma revisão de Musmann *et al* [MPG85] estão presentes diversos algoritmos classificados em duas categorias:

1. Algoritmos recursivos: A idéia é fazer uma primeira estimativa do deslocamento e melhorá-la recursivamente baseando-se no gradiente da função de erro de deslocamento.

2. Algoritmos de casamento de blocos[6]: Usa-se um certo critério de comparação para buscar o melhor casamento, no quadro seguinte, de um bloco do quadro corrente. A estimativa é feita verificando-se a diferença de posição entre os blocos.

Em uma outra abordagem feita por Jain [Jai89], são definidas três diferentes classes de algoritmos:

1. Técnicas de busca: Semelhante aos algoritmos de casamento de blocos citados acima. Diferentes métodos de busca podem ser utilizados para encontrar o vetor de deslocamento que minimiza a distorção entre o quadro de referência e o corrente.

2. Técnicas recursivas: Os mesmos algoritmos recursivos citados por Musmann *et al.*

3. Técnicas de diferenciação: A seqüência de imagens é considerada como uma função contínua da forma $u(x, y, t)$, que representa os valores de pixel da imagem em um tempo t. São

[6] Em inglês *block-matching algorithms*

utilizados mecanismos da análise diferencial para fazer a estimativa de deslocamento, levando-se em conta que os valores de pixel permanecem constantes ao longo de suas trajetórias, isto é, se $x(t), y(t)$ representa a trajetória de um determinado pixel, então $u(x(t), y(t), t)$ permanece constante para todo t.

Uma vez estimado o movimento, a compressão é atingida descartando-se alguns quadros e reproduzindo-os, através de repetição ou interpolação, ao longo da trajetória de movimento. Por exemplo, se são descartados os quadros alternados $u(m, n, 2i)$ $(i = 1, 2, \ldots)$, eles podem ser reconstruídos, utilizando repetição de quadros, da seguinte maneira:

$$u^*(m, n, 2i) = u^*(m - p, n - q, 2i) \tag{2.19}$$

ou usando interpolação:

$$u^*(m, n, 2i) = \frac{1}{2}(u^*(m - p, n - q, 2i - 1) + u^*(m - p', n - q', 2i + 1)) \tag{2.20}$$

onde (p, q) e (p', q') são os vetores de deslocamento relativos aos quadros anterior e posterior respectivamente. Sem compensação de movimento, teria-se $p = q = p' = q' = 0$, o que daria origem a uma imagem reproduzida com qualidade significativamente inferior [Jai89].

2.4.4 Troca de Resolução

O sistema visual humano não é capaz de distinguir com precisão cenas que contêm, simultaneamente, alta freqüência espacial e temporal. Portanto, áreas que mudam rapidamente em uma cena podem ser representados com uma resolução espacial menor do que o restante da imagem. Por outro lado áreas estacionárias podem ser representadas com menor resolução temporal. Desta maneira, pode haver uma troca entre resolução espacial e temporal, da qual pode resultar imagens de boa qualidade usando por volta de 2 a 2,5 bits por pixel.

Em um exemplo deste tipo de método, a imagem é segmentada em áreas estacionárias e em movimento, isto é feito verificando se as diferenças entre os pixels estão abaixo ou acima de um certo limiar. Nas regiões estacionárias, as diferenças entre pixels são transmitidas e todos os demais pixels são copiados do quadro anterior. Nas regiões em movimento, é feita uma subamostragem horizontal de 2:1, de tal maneira que os elementos descartados são reconstruídos por interpolação ao longo de cada linha. Se forem utilizados, por exemplo, 5 bits por pixel para codificar as diferenças, pode-se conseguir uma representação das imagens com uma média de 2.5 bits/pixel. A principal distorção que pode ocorrer é em arestas acentuadas movendo-se com velocidade moderada.

2.5 Padrões em Compressão de Imagens

Nas seções anteriores vimos que existe uma grande variedade de técnicas de compressão de imagens que podem ser utilizadas tanto na codificação de imagens isoladas quanto para seqüências de vídeo. Estas técnicas, quando bem combinadas, permitem atingir altas taxas de compressão sem afetar de maneira significativa a qualidade visual das imagens. Tal proposta tem seu lado positivo, no sentido de que para cada aplicação pode ser adotado o método, ou combinação de métodos, que for mais conveniente ou atingir os melhores resultados. Entretanto, para tornar aplicações que envolvem armazenagem ou transmissão de imagens amplamente difundidas no mercado internacional,

surge a necessidade de se criar um método padrão de compressão que permita a interoperabilidade entre equipamentos de diferentes fabricantes. Foi neste sentido que, em um esforço conjunto entre CCITT[7] e ISO[8], foi criado o comitê conhecido por JPEG (*Joint Photographic Experts Group*) [Wal91] que tem trabalhado para estabelecer o primeiro padrão de compressão de imagens isoladas. No caso de seqüências de imagens, a ISO mantém um outro comitê denominado MPEG (*Moving Picture Experts Group*) [Le 91], que tem como objetivo a padronização dos algoritmos de compressão de seqüências de vídeo para aplicações de multimídia. As seções a seguir dão uma breve descrição dos métodos desenvolvidos por estes comitês.

2.5.1 O Padrão JPEG

O JPEG foi incumbido da ambiciosa tarefa de elaborar um padrão de compressão que reunisse as necessidades de quase todas aplicações envolvendo imagens isoladas de tonalidade contínua[9]. Para tanto, foram estabelecidos os seguintes requisitos a serem obedecidos no desenvolvimento do método:

a) Permitir que a aplicação (ou usuário) possa escolher uma entre várias taxas de compressão possíveis, de modo que possa ser encontrada a relação compressão/qualidade mais adequada a cada caso.

b) Ser aplicável a praticamente todo tipo de imagem digital de tonalidade contínua, sem restrições quanto à complexidade, variedade de cores ou propriedades estatísticas.

c) Ter uma complexidade computacional aceitável, para tornar possível implementações em *software* com desempenho e ocupação de CPU viáveis, assim como implementações em *hardware* com custo praticável para aplicações que requerem alto desempenho.

d) Ter os seguintes modos de operação:

- Codificação seqüencial: cada componente da imagem (pixels ou blocos) é codificado em uma única varredura, da esquerda para a direita e de cima para baixo.
- Codificação progressiva: a imagem é codificada em múltiplas varreduras, para aplicações onde o tempo de transmissão é longo e o observador prefere apreciar a construção da imagem em vários passos com uma melhora progressiva da qualidade.
- Codificação sem perda: a imagem é codificada de modo a garantir a exata recuperação de toda a informação da imagem original, naturalmente, neste tipo de codificação as taxas de compressão serão inferiores àquelas obtidas nos modos que permitem perda de informação.

[7] *Comité Consultatif International de Téléphonie et Télégrafie.*

[8] Acrônimo de *International Standards Organization* apesar do nome verdadeiro ser *International Organization for Standardization.*

[9] Imagens que possuem muitos níveis de resolução de amplitude, permitindo assim a noção de continuidade na variação de cores: em geral são 256 níveis (8 bits) ou mais por componente de cor.

- Codificação hierárquica: a imagem é codificada em múltiplas resoluções, de modo que se tenha acesso a versões de menor resolução sem que seja preciso descomprimir a imagem em sua resolução total.

O padrão proposto contém os quatro modos de operação citados acima. Para cada modo são especificados um ou mais codificadores e decodificadores que diferem entre si de acordo com a precisão das imagens que manipulam. Entretanto é improvável que muitas implementações utilizem todos os modos estabelecidos: o método utilizado para o modo seqüencial já cobre as necessidades da maioria das aplicações. A seguir damos uma breve descrição dos passos do esquema seqüencial baseado em DCT (*discrete cosine transform*) e do método com predição utilizado para o modo de operação sem perda.

Codificação Baseada em DCT

Os passos de processamento que descrevemos aqui são os utilizados no modo seqüencial, mas também são a base de funcionamento dos modos progressivo e hierárquico. O leitor que não é familiarizado com métodos de compressão de imagens baseados em DCT pode consultar o Capítulo 5 antes de prosseguir com a leitura desta seção, isto facilitará a compreensão do método.

A Figura 2.7 mostra os passos de processamento deste esquema para imagens de um único componente, isto é, em tons de cinza (imagens coloridas podem ser tratadas como múltiplas imagens em tons de cinza). É feita uma divisão do quadro em blocos de 8×8 pixels, e cada um deles é codificado separadamente. O primeiro passo é o FDCT (*forward DCT*), que em si não produz nenhuma compressão, apenas dá uma nova representação para a imagem, de modo que ela possa ser codificada de maneira mais eficiente. A maior parte da informação contida no bloco fica concentrada em um pequeno número de coeficientes, de tal forma que boa parte deles pode ser codificada com menos bits ou simplesmente descartada. O coeficiente do canto superior esquerdo, chamado de nível DC, contém a média de todos os valores de pixel do bloco, e é próximo a ele que haverá a maior concentração de informação. Tendo em vista que em geral existe uma forte relação entre os valores médios de blocos vizinhos, para codificar o nível DC de cada bloco, o que será quantizado é a diferença com relação ao nível DC do bloco anterior. Para os outros 63 coeficientes, chamados níveis AC, será utilizada uma tabela de quantização, que poderá ser especificada pela aplicação (ou usuário), contendo valores inteiros entre 1 e 255, de tal forma que cada coeficiente será quantizado da seguinte maneira:

$$U^Q(k,l) = \left[\frac{U(k,l)}{Q(k,l)}\right] \tag{2.21}$$

onde $U(k,l)$ $(k,l = 0, 1, \ldots, 7)$ representa os coeficientes resultantes do FDCT sobre o bloco original u, e $Q(k,l)$ representa os valores da tabela de quantização. Os colchetes indicam arredondamento para o inteiro mais próximo (que causa alguma perda de informação). Observe que, quanto maior for o valor de $Q(k,l)$ menor será a quantidade de bits necessária para representar $U^Q(k,l)$, e é justamente daí que vem a compressão.

A dequantização é o processo inverso, os coeficientes são recuperados usando-se a mesma tabela de quantização, pela fórmula:

$$U^*(k,l) = U^Q(k,l)Q(k,l) \tag{2.22}$$

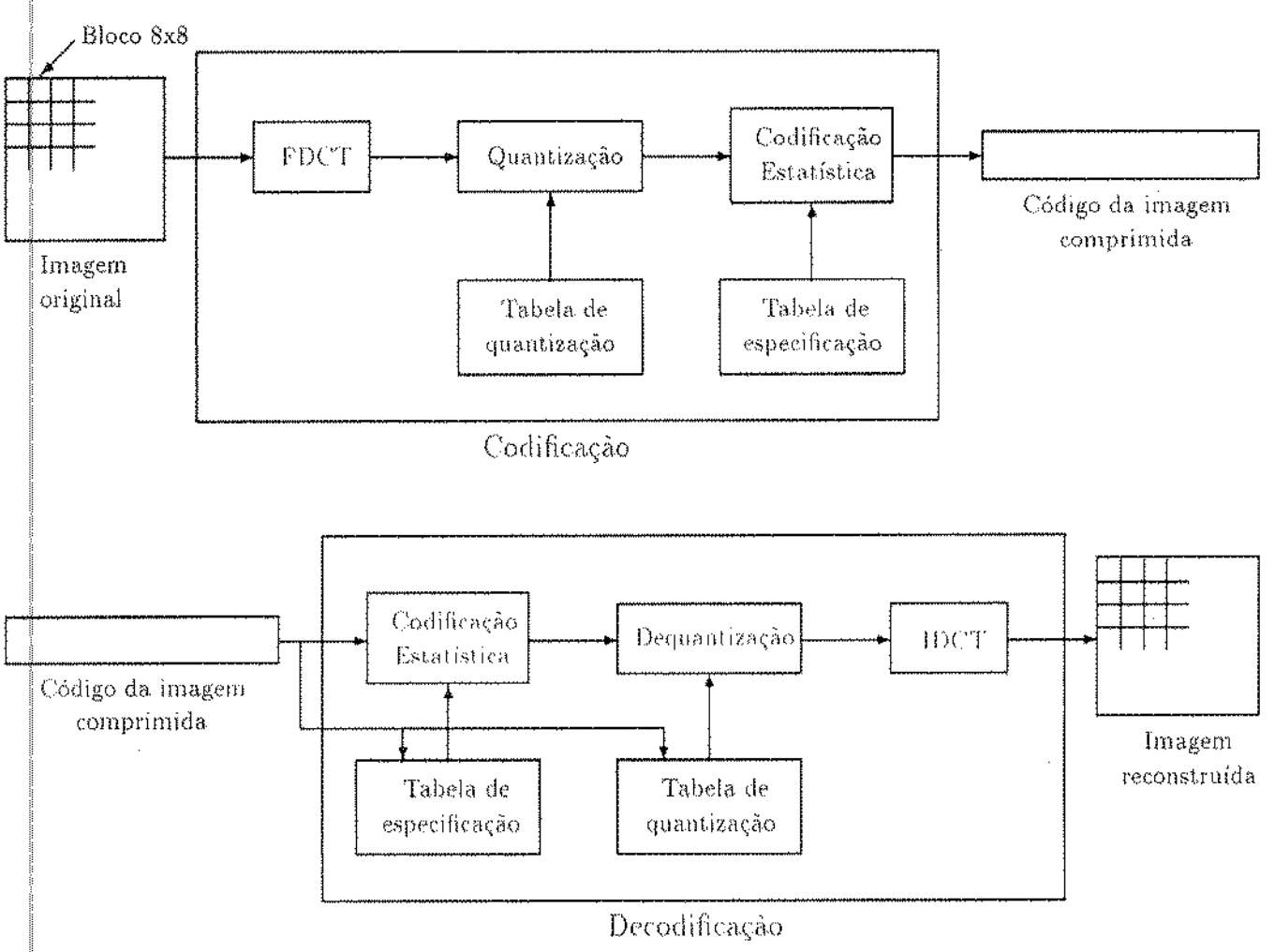


Figura 2.7: JPEG: Esquema seqüencial baseado em DCT.

Uma vez calculado U^*, o bloco reconstruído u^* é obtido através do IDCT (*inverse DCT*).

Na Figura 2.7 pode-se observar que existe mais um passo que permite atingir uma compressão adicional, a codificação estatística (ver Seção 2.3.2), que não envolve perda de informação. Existem variações que utilizam código de Huffman ou codificação aritmética neste passo, usando tabelas de especificação que podem ser determinadas pela aplicação ou predefinidas pelo método.

No caso de imagens coloridas, os resultados obtidos com os diferentes modos de operação são os seguintes:

- 0,25 - 0,5 bits/pixel: qualidade moderada a boa, suficiente para algumas aplicações;
- 0,5 - 0,75 bits/pixel: qualidade boa a muito boa, suficiente para muitas aplicações;
- 0,75 - 1,5 bits/pixel: qualidade excelente, suficiente para a maioria das aplicações;
- 1,5 - 2,0 bits/pixel: geralmente indistingüível da imagem original, suficiente para as aplicações mais exigentes.

Estes níveis se referem a cenas moderadamente complexas, e podem variar de maneira significativa de acordo com as características da imagem original e conteúdo da cena.

Codificação sem Perda

Para o modo de operação sem perda, o JPEG escolheu um simples método com predição que é totalmente independente do esquema baseado em DCT descrito acima. A Figura 2.8 ilustra os passos de processamento para uma imagem de um único componente. Para cada pixel o valor predito é subtraído do valor original, como é feito no DPCM (ver Seção 2.3.7), mas aqui, ao invés da quantização, é utilizado um método de codificação estatística para representar os erros de predição. Como no esquema anterior, pode ser usado o código de Huffman ou a codificação aritmética neste passo.

A regra de predição pode ser determinada pela aplicação a partir de uma tabela especificada no padrão, contendo regras predefinidas em uma e duas dimensões.

A codificação sem perda pode utilizar qualquer precisão de imagem, entre 2 e 16 bits/pixel, e em geral produzem taxas de compressão em torno de 2:1 para imagens coloridas com cenas moderadamente complexas.

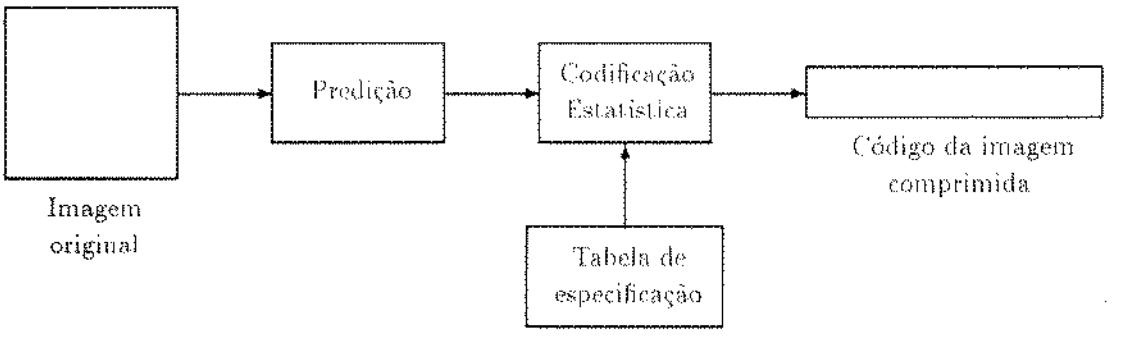


Figura 2.8: JPEG: Codificação sem perda.

2.5.2 O Padrão MPEG

Uma vez que para cada sinal de vídeo geralmente está associado um sinal de áudio, as atividades do MPEG abrangem, não apenas técnicas de compressão de seqüências de vídeo, mas também sincronização e compressão de sinais de áudio. No entanto tratamos aqui apenas das atividades referentes a sinais de vídeo, desenvolvidas por uma sub-comissão do MPEG denominada MPEG-Video, que no intuito de desenvolver um método que cubra as necessidades da maioria das aplicações, identificou os seguintes requisitos a serem levados em consideração:

- Acesso aleatório: qualquer quadro da seqüência comprimida deve estar acessível em um curto espaço de tempo.
- Busca rápida para frente e para trás: deve ser possível percorrer a seqüência comprimida e selecionar quadros apropriados para se ter o efeito de movimento acelerado para frente e para trás.
- Retorno de imagem: deve ser possível exibir a seqüência de vídeo na ordem inversa.
- Sincronização de imagem e som: um mecanismo deve manter permanentemente a resincronização entre áudio e vídeo.
- Tolerância a erros: deve haver uma proteção contra propagação de erros, tendo em vista que a maioria dos canais de comunicação não são livres de erros.
- Atraso de codificação/decodificação: o algoritmo deve ter bom desempenho dentro de um limite aceitável de atraso e este deve ser deixado como um parâmetro, permitindo assim que a aplicação possa escolher a melhor relação atraso/qualidade.
- Editabilidade: deve ser possível fazer pequenas inserções que sejam codificadas apenas com referência a si mesmas, atingindo assim, um nível aceitável de editabilidade na forma comprimida.
- Flexibilidade de formato: o algoritmo deve ser aplicável a uma grande variedade de formatos (com respeito a altura, largura e número de bits por pixel).
- Custo: o método proposto deve ser implementável em um pequeno número de chips, mantendo desempenho em tempo real.

O requisito de acesso aleatório seria mais facilmente atingido se fosse utilizada apenas a codificação intraquadros (cada quadro isoladamente), entretanto a necessidade de atingir altas taxas de compressão, mantendo uma boa qualidade, força a utilização de técnicas interquadros. Deve-se, contudo, tomar o cuidado de não comprometer a possibilidade de acesso aleatório. Encontrar o melhor balanceamento entre codificação intra e interquadros foi um dos maiores desafios do MPEG na elaboração do algoritmo do qual damos uma visão geral a seguir.

Redução de Redundância Temporal

A técnica básica utilizada nesta fase é compensação de movimento sobre blocos de 16×16, usando para estimativa de movimento um algoritmo de casamento de blocos (ver Seção 2.4.3). Para manter

o requisito de acesso aleatório são considerados três tipos de imagens no MPEG: intraimagens, imagens preditas e imagens interpoladas.

Intraimagens fornecem os pontos de referência visando acesso aleatório. Como o nome já diz, são usadas apenas técnicas intraquadros para codificar estas imagens, resultando em uma taxa apenas moderada de compressão.

Imagens preditas são codificadas com referência a uma imagem anterior (intra ou predita) e geralmente são usadas como referência para imagens preditas posteriores.

Imagens interpoladas são resultantes de predição bidirecional, ou seja, são usadas tanto imagens anteriores quanto posteriores como referência. É nestas imagens que se consegue a maior taxa de compressão, e em geral há de dois a três quadros deste tipo para cada imagem dos outros dois tipos, mas elas jamais são usadas como referência.

Em todos os casos, quando uma imagem é codificada com relação a uma ou mais referências, a compensação de movimento é usada para melhorar a eficiência da codificação.

Redução de Redundância Espacial

O MPEG, assim como o JPEG, escolheu codificação baseada em DCT com blocos de 8×8 para redução de redundância espacial, usando também codificação estatística para melhorar a taxa de compressão. Entretanto existem algumas diferenças com relação ao algoritmo JPEG. A principal delas está no processo de quantização. Existem dois tipos diferentes de tabelas de quantização para os diferentes tipos de imagens do MPEG. Um deles se destina a intraimagens, que precisam de maior exatidão pois tendem a conter energia em todas as freqüências, sendo portanto mais vulneráveis a erros de quantização. O outro tipo de tabela de quantização se destina a imagens preditas ou interpoladas, que são representadas por erros de predição e tendem a conter predominantemente componentes de alta freqüência e que podem ser submetidos a uma quantização menos rigorosa que permita uma taxa mais elevada de compressão.

Após DCT e quantização, a técnica de codificação estatística que é utilizada é uma combinação de *run-length* (ver Seção 2.3.3) com código de Huffman. Para cada par {comprimento, valor de pixel} resultante do *run-length* é utilizada uma palavra de código com tamanho tão menor quanto maior for a probabilidade de ocorrência do respectivo par. Este procedimento, que não envolve perda de informação, tem o objetivo de aumentar ainda mais a taxa de compressão.

Taxas de Compressão

O algoritmo MPEG foi projetado de modo a permitir operabilidade com uma grande variedade de formatos. Por esta razão, um conjunto de parâmetros de vídeo foi selecionado, com o intuito de estabelecer os objetivos a serem atingidos pelo MPEG quanto a taxas de compressão. A Tabela 2.2 mostra algumas das perspectivas que foram estabelecidas. Nela é indicado o número de bits por segundo a serem transmitidos na forma comprimida, mantendo uma qualidade aceitável para as imagens reproduzidas.

2.6 Conclusões

Neste capítulo demos uma visão geral sobre as principais técnicas de compressão de imagens, indicando quase sempre as taxas de compressão que podem ser atingindas por cada uma delas.

Formato	Dimensões (pixels)	Freqüência (quadros/segundo)	Forma comprimida (Mbits/segundo)
SIF	325x240	30	1,2 a 3
CCIR 601	720x486	30	5 a 10
EDTV	960x486	30	7 a 15
HDTV	1920x1080	30	20 a 40

Tabela 2.2: Perspectivas de aplicação do algoritmo MPEG.

Também foram descritos os primeiros padrões desenvolvidos para compressão de imagens isoladas e seqüências de vídeo. As descrições feitas foram um tanto sucintas, muitas vezes dando apenas uma noção básica sobre o método, mas nos capítulos seguintes descrevemos com maior riqueza de detalhes, duas das técnicas mais amplamente utilizadas atualmente: quantização vetorial e codificação por transformadas. Nosso interesse especial por estas técnicas deve-se ao fato delas permitirem atingir altas taxas de compressão, e de algumas de suas variações possibilitarem codificação e decodificação em tempo real sem um excessivo custo computacional. Além do mais estes métodos podem ser combinados com técnicas de compressão interquadros para obter esquemas bastante eficientes de compressão de seqüências de imagens.

Leituras Complementares

Várias revisões sobre codificação de imagens podem ser encontradas na literatura. Jain mostra em [Jai81] uma grande variedade de algoritmos com exemplos e resultados. Também são vistos os efeitos que podem ser causados por erros de canais de comunicação em diferentes métodos. Quase todo o conteúdo deste artigo de Jain pode também ser encontrado em diferentes partes de [Jai89], especialmente no Capítulo 11. Netravali e Limb, em uma abordagem um pouco diferente, também cobrem em [NL80] uma grande variedade de técnicas de codificação de imagens. Musmann *et al* descrevem em [MPG85] os principais avanços em codificação de sinais de vídeo até o ano de publicação do referido artigo, tratando especialmente de algoritmos de estimativa de deslocamento para compensação de movimento, codificação preditiva e por transformadas. Geus também apresenta em [Geu90] uma revisão bastante abrangente, descrevendo inclusive, métodos que não aparecem nas revisões citadas acima, como BTC e quantização vetorial.

Capítulo 3

Quantização Vetorial

3.1 Introdução

Neste capítulo, voltamos nossa atenção para uma técnica que pode atingir de maneira bastante satisfatória aos nossos objetivos, pois além de ter um bom desempenho quanto à compressão de imagens, também pode ser bastante rápida; esta técnica é a quantização vetorial, uma generalização multidimensional do processo de quantização definido na Seção 2.3.1 e que tem chamado a atenção de um grande número de pesquisadores nos últimos anos, dando origem a trabalhos como o de Gersho [Ger82], Gray [Gra84] e Nasrabadi e King [NK88], e muitos outros.

Iniciamos esta revisão dando uma definição formal de quantização vetorial e descrevendo o esquema básico de codificação de imagens que usa esta técnica, em seguida mostramos como um quantizador vetorial pode ser projetado e otimizado e levantamos alguns dos problemas de codificação que surgem com o Esquema Básico. Nas seções seguintes, passamos à descrição de algumas variações, com e sem uso de memória, dando maior ênfase àquelas que são mais diretamente relacionadas com a pesquisa aqui desenvolvida. Estas variações permitem reduzir alguns dos inconvenientes do Esquema Básico, bem como melhorar a qualidade da imagem reproduzida com menor custo computacional e de memória.

3.2 Esquema Básico de Quantização Vetorial

Como uma generalização da quantização escalar definida na Seção 2.3.1, um quantizador vetorial é definido como uma função do tipo:

$$Q : R^k \rightarrow D, \tag{3.1}$$

onde R^k é o espaço euclideano k-dimensional e D é um subconjunto finito de R^k, ou seja, ao invés de números, são mapeados vetores do tipo $\mathbf{x} = (x_1, x_2, \ldots, x_k)$ em um conjunto predefinido $D \subset R^k$. Este mapeamento ocorre na verdade por meio de duas funções, um codificador α e um decodificador β, do tipo:

$$\alpha : R^k \rightarrow M \quad \text{e} \quad \beta : M \rightarrow D, \tag{3.2}$$

onde M é o conjunto dos símbolos de canal (seqüências de bits) de comprimento igual a $\log_2 m$, supondo-se que m, o número de elementos em D, seja uma potência de 2. Q é então dado pela

composição de β com α.

$$Q = \beta \circ \alpha. \tag{3.3}$$

No caso de processamento de imagens, os vetores geralmente representam pequenos blocos de imagem de forma e tamanho fixos, e o conjunto D, que chamaremos a partir daqui de *dicionário*, conterá uma coleção convenientemente escolhida de padrões de reconstrução, que chamaremos de *vetores de código*, cada um dos quais associado a um símbolo de canal de M. A Figura 3.1 ilustra o Esquema Básico de Quantização Vetorial (supondo um canal de comunicação livre de erro). Para cada bloco da imagem original, o codificador procura o vetor de código do dicionário com relação ao qual existe a menor distorção, e transmite (ou armazena) o símbolo de canal a ele associado. O decodificador, que também dispõe do mesmo dicionário, reconstrói a imagem simplesmente montando cada um dos vetores de código para os quais as símbolos de canal recebidos apontam.

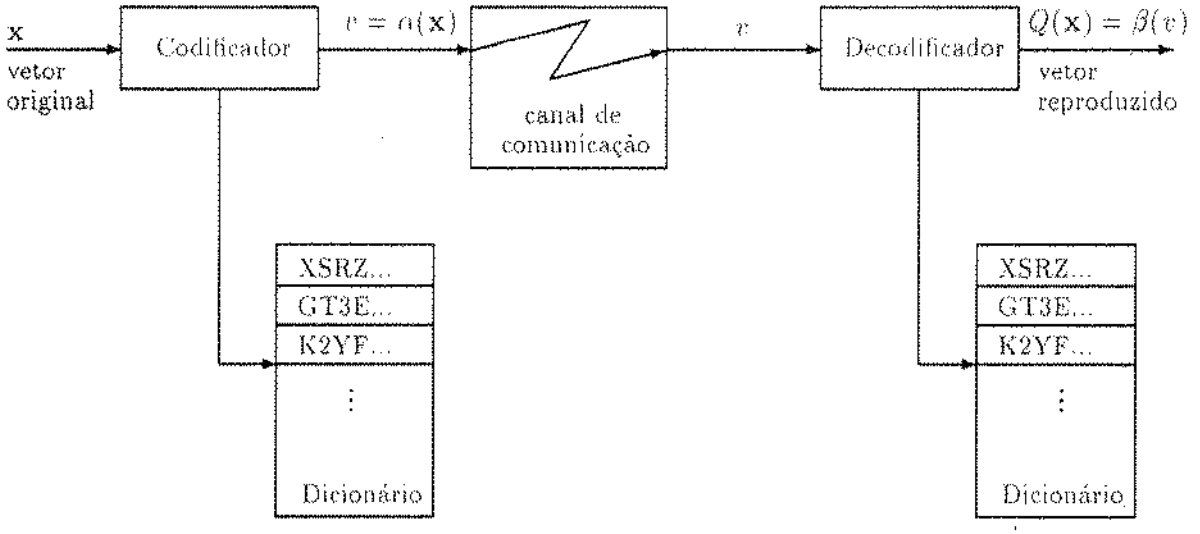


Figura 3.1: Esquema Básico de Quantização Vetorial.

Para fazer a escolha dos vetores de código que representam cada bloco, uma medida de distorção freqüentemente utilizada é o *erro quadrático*, ou seja, o quadrado da distância no espaço euclideano k-dimensional:

$$d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^2 = \sum_{i=1}^{k} (x_i - y_i)^2. \tag{3.4}$$

Existem outras medidas de distorção que podem supostamente dar resultados melhores, mas em geral aumentam bastante o custo computacional, tendo em vista a grande quantidade de computações de distorção que normalmente são necessárias.

3.2.1 Quantizadores Ótimos

O sucesso de métodos que utilizam quantização vetorial depende principalmente de uma boa escolha do dicionário que é utilizado. A melhor escolha é aquela que minimiza a distorção média (ou expectância de distorção) $E[d(\mathbf{x}, Q(\mathbf{x}))]$. Linde, Buzo e Gray [LBG80] desenvolveram um algoritmo (conhecido como LBG) para projeto de dicionários para quantizadores ótimos, que é uma generalização do algoritmo de Lloyd-Max para construção de quantizadores escalares (ver apêndice B), e cuja idéia básica é a de definir os vetores de código do dicionário como sendo os centróides (ou centros de massa) de um conjunto de vetores de entrada. Para dar início ao algoritmo são necessários dois elementos iniciais: uma seqüência de treinamento e um dicionário inicial.

A seqüência de treinamento é a coleção de vetores segundo a qual o dicionário resultante é ótimo; isto significa que quanto maior e mais variada for esta coleção, melhor será o quantizador resultante. Esta seqüência pode ser extraída de um conjunto de imagens que seja suficientemente grande e diversificado.

O dicionário inicial pode ser construído simplesmente escolhendo alguns vetores da seqüência de treinamento, de preferência bem distanciados entre si para evitar as semelhanças que normalmente existem entre blocos vizinhos. Pode-se também começar com um dicionário de vetores de menor dimensão, e então a partir de todas as concatenações possíveis construir dicionários para dimensões superiores (uma outra estratégia é mostrada na seção 3.3.1).

Uma vez de posse da seqüência de treinamento e do dicionário inicial, o algoritmo LBG prossegue da seguinte maneira:

passo 1: codifica a seqüência de treinamento em uma seqüência de símbolos de canal de M, verificando, para cada vetor de entrada, qual o vetor de código do dicionário corrente com o qual há menor distorção; reconstrói a seqüência e mede a distorção média; se for pequena o bastante termina.

passo 2: substitui cada vetor de código do dicionário corrente pelo centróide do conjunto de todos os vetores da seqüência de treinamento que foram mapeados em cada um dos respectivos símbolos de canal no passo anterior; volta ao passo 1.

Matematicamente falando, no passo 1 ocorre o seguinte: para cada símbolo de canal $v \in M$, é associado um conjunto C_v de vetores de treinamento, tais que, dado um certo $\mathbf{x}$ da seqüência de treinamento,

$$\mathbf{x} \in C_v \iff d(\mathbf{x}, \beta(v)) = \min_{w \in M} d(\mathbf{x}, \beta(w)) = \min_{\mathbf{y} \in D} d(\mathbf{x}, \mathbf{y}). \tag{3.5}$$

Em palavras, $\mathbf{x}$ está em C_v se, e somente se, $\beta(v)$ é o vetor de código que tem menor distorção com relação a $\mathbf{x}$.

No passo 2, cada vetor de código $\beta(v) \in D$ é substituído pelo centróide $c(v)$ dado por:

$$c(v) = \frac{1}{n_v} \sum_{\mathbf{x} \in C_v} \mathbf{x}, \tag{3.6}$$

onde n_v é o número de elementos em C_v.

Dicionários gerados desta maneira são ótimos apenas localmente, como já dissemos, com respeito à seqüência de treinamento utilizada. É recomendável utilizar dicionários iniciais que sejam

bastante diversificados, e pode-se, alternativamente, introduzir perturbações no processo de codificação com este objetivo. Dicionários iniciais melhores podem dar origem a quantizadores melhores, mas somente através da experimentação é que se pode determinar qual a melhor opção.

3.2.2 Problemas com o Esquema Básico de Quantização Vetorial

O primeiro problema que surge quando se utiliza o Esquema Básico de Quantização Vetorial, é o grande número de operações necessário à codificação. Para dar uma idéia, suponha que queremos codificar uma imagem de 512×512 pixels usando um dicionário de 1024 vetores de código representando blocos de 4×4 pixels. A imagem seria dividida em 16384 blocos e cada um deles seria comparado com todos os 1024 vetores, o que daria da ordem de $1,6 \times 10^7$ medidas de distorção. Isto impossibilita o uso deste método em aplicações que requerem tempo real.

Um outro inconveniente é o longo tempo necessário à construção dos dicionários, bem como a quantidade de memória necessária para armazená-los. Isto não chega a ser um impecilho se o dicionário é fixo, já que pode ser gerado uma única vez e armazenado em ROM, o que entretanto, torna impraticável o uso de dicionários adaptativos.

Vários métodos alternativos já foram propostos para resolver problemas como estes. Nas seções seguintes descrevemos alguns deles.

3.3 Variações de Quantização Vetorial sem Memória

Nesta seção consideramos algumas estratégias que podem ser utilizadas com quantização vetorial com o objetivo de reduzir o custo computacional de codificação. São estudados também, alguns métodos que visam atingir melhores resultados para dicionários de um determinado tamanho. A expressão 'sem memória' significa não utilização do passado, isto é, vetores que já foram codificados não têm nenhuma influência sobre como é codificado o vetor corrente.

3.3.1 Busca em Árvore

Esta técnica, que é conhecida como *tree-searched VQ*, foi inicialmente sugerida por Buzo *et al* [BGGM80] e é uma alternativa para a solução do problema de custo computacional de busca em quantização vetorial. Ela se baseia na construção de um dicionário formado como uma estrutura de árvore. Para facilitar a compreensão vamos considerar um caso particular de construção de um dicionário baseado em uma árvore binária.

Iniciamos com um dicionário ótimo de um único vetor $\mathbf{x}$, que é o centróide de toda a seqüência de treinamento; em seguida, usamos um pequeno vetor de perturbação ϵ, para dividir $\mathbf{x}$ em dois vetores, $\mathbf{x}$ e $\mathbf{x}+\epsilon$ que podem ser usados como vetores de código iniciais para construir um dicionário ótimo de dois elementos utilizando o algoritmo LBG. Mais uma vez, introduzimos uma pequena perturbação nos centróides, produzindo os vetores de código iniciais de que darão origem a um dicionário ótimo de quatro elementos. Dando seqüência a este processo podemos construir o dicionário do tamanho desejado (no caso uma potência de 2).

Esta estratégia pode também ser uma alternativa para obtenção do dicionário inicial para o algoritmo LBG. No entanto, no caso do *tree-searched*, cada vez que um vetor é dividido são mantidos dois apontadores para os vetores resultantes (que são posteriormente otimizados), de tal forma que é construída uma estrutura de árvore como a mostrada na Figura 3.2, na qual cada vetor

de código é representado da forma $\mathbf{X}_{ijk}$, onde ijk indica o símbolo de canal associado a cada um deles. Para codificar um vetor de entrada são feitas apenas duas comparações em cada nível da árvore, e o símbolo de canal é montado bit a bit, baseando-se na direção que é tomada em cada passo. Isto resulta em $2\log_2 n$ comparações no caso do dicionário final ter n vetores de código. Em uma situação como a do exemplo citado na Seção 3.2.2, o número de comparações cairia de 1024 para 20 por bloco.

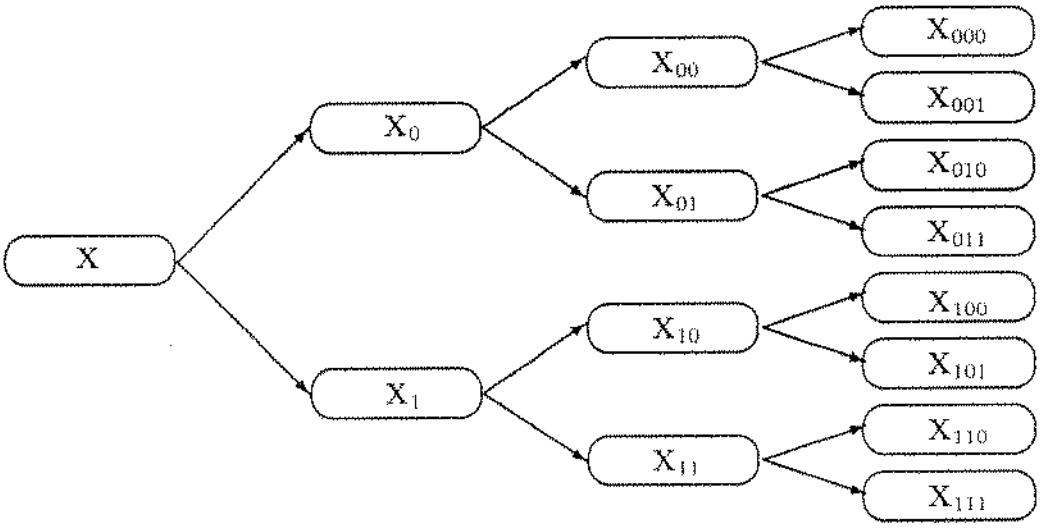


Figura 3.2: *Tree-searched VQ*.

Esta técnica tem algumas desvantagens com relação ao Esquema Básico. Uma delas é que o espaço de memória necessário é duplicado devido à necessidade de armazenar os dicionários intermediários. Um outro problema a se considerar, é que a codificação não é ótima, isto é, não há garantia de que cada vetor de código escolhido para a codificação é de fato a melhor escolha, entretanto os resultados atingidos são bem próximos daqueles obtidos com o Esquema Básico, e o ganho em eficiência torna este método bem mais atrativo, especialmente quando pensamos em aplicações que requerem tempo real.

3.3.2 Múltiplos Passos

Este método, que é chamado *Multistep VQ*, é tão rápido quanto o *tree-searched* e tem a vantagem de ocupar menor quantidade de memória. Por simplicidade, vamos considerar o caso de uma estrutura binária que é ilustrado na Figura 3.3. O primeiro dicionário (D_1), que tem apenas dois elementos, é construído exatamente como no *tree-searched VQ*. No passo seguinte toda a seqüência de treinamento é codificada utilizando D_1, e então sinais de erro são gerados para cada um dos vetores, dando origem a uma seqüência de treinamento de diferençass que é usada para construir o segundo dicionário (D_2) também de dois elementos. Mais uma vez os sinais de erro são gerados para construir o dicionário seguinte, e assim por diante. Como resultado teremos uma seqüência de dicionários com dois vetores de código cada um.

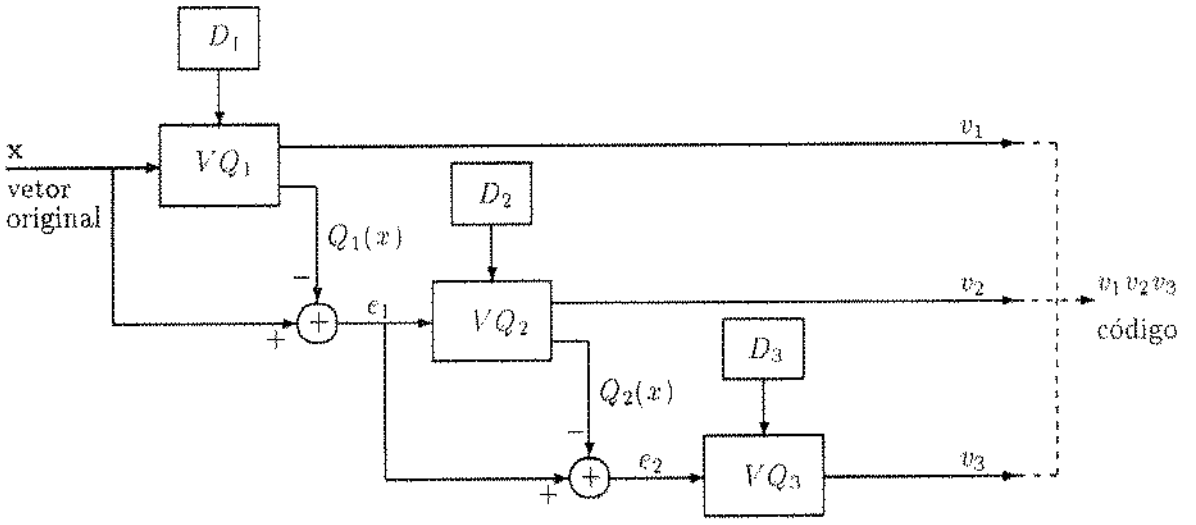


Figura 3.3: *Multistep VQ*.

Para reconstruir um vetor, basta percorrer o símbolo de canal bit por bit (no caso dos dicionários serem binários) e ir somando os vetores de código a que se referem em cada um dos dicionários. Se temos, por exemplo, r dicionários, são possíveis 2^r reconstruções diferentes, com uma armazenagem de apenas $2r$ vetores de código, enquanto que para uma situação equivalente usando *tree-searched VQ* seria preciso uma armazenagem de 2^{r+1} vetores.

Apesar do número de reconstruções possíveis ser o mesmo, este tipo de codificação não produz resultados tão bons quanto o *tree-searched VQ*, entretanto o desempenho ainda é aceitável e sua grande vantagem é que produz uma economia substancial de espaço em memória.

3.3.3 Separação de Norma

Esta técnica, que foi sugerida pela primeira vez por Buzo *et al* [BGGM80], foi posteriormente otimizada por Sabin e Gray [SG84] e é comumente chamada de *gain/shape VQ*. Sua idéia básica é a codificação de cada vetor em dois componetes separados mas interdependentes: a norma (*gain*) e o formato (*shape*).

A norma de um vetor $\mathbf{x}$ é definida pela raiz quadrada do produto interno de $\mathbf{x}$ por si mesmo, isto é,

$$\|x\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}, \tag{3.7}$$

entendendo-se produto interno entre dois vetores $\mathbf{x} = (x_1, ..., x_k)$ e $\mathbf{y} = (y_1, ..., y_k)$ como sendo

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{k} x_i y_i. \tag{3.8}$$

O formato de um vetor $\mathbf{x}$, no caso deste método, é dado a partir de sua normalização,

$$\hat{\mathbf{x}} = \frac{1}{\|\mathbf{x}\|}\mathbf{x}. \tag{3.9}$$

Para a codificação, precisamos de um dicionário $Y = \{\mathbf{y}_1, ..., \mathbf{y}_r\}$ de vetores normalizados ($\|\mathbf{y}_i\| = 1$ para $i = 1, ..., r$), e um dicionário de escalares $W = \{\sigma_1, ..., \sigma_s\}$. Ao codificar um certo vetor $\mathbf{x}$, procedemos como na Figura 3.4. A melhor descrição do formato de $\mathbf{x}$ é dada por um certo $\mathbf{y}_n \in Y$ tal que $\langle \mathbf{x}, \mathbf{y}_n \rangle$ é máximo, e a melhor aproximação para a norma é o escalar $\sigma_m \in W$ que minimiza a expressão $\sigma_j^2 - 2\sigma_j \langle \mathbf{x}, \mathbf{y}_n \rangle$ para $j = 1, ..., s$.

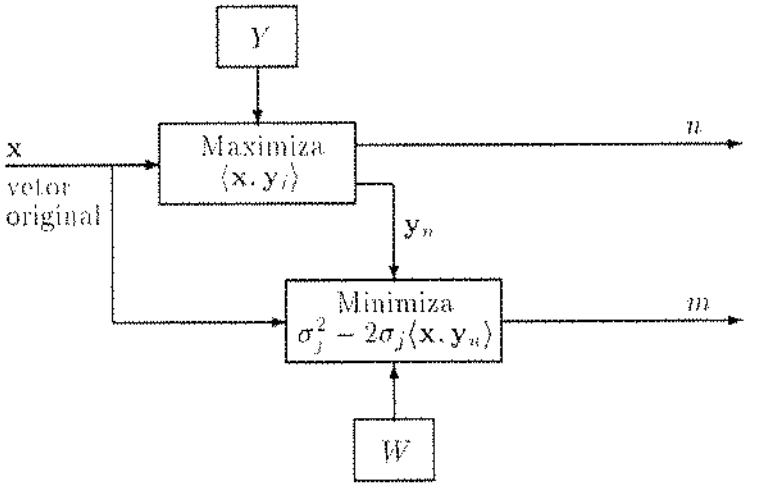


Figura 3.4: *Gain/shape VQ.*

O decodificador, que também dispõe dos dicionários Y e W reconstrói o vetor como sendo

$$\mathbf{x}^* = \sigma_m \mathbf{y}_n \tag{3.10}$$

Este método permite atingir, com dicionários menores, um padrão de qualidade que só seria atingido pelo Esquema Básico se fosse utilizado um dicionário bem maior, o que acarretaria em mais consumo de memória. O número total de bits utilizado para representar cada vetor, neste caso, é dividido entre norma e formato; a melhor divisão de bits entre estes dois componentes é algo que em geral é determinado experimentalmente.

3.3.4 Separação de Média

Esta técnica consiste, como o método anterior, em processar cada vetor de entrada em dois componentes separados, a média e o formato. A média de cada vetor do tipo $\mathbf{x} = (x_1, ..., x_k)$ é dada por,

$$\overline{x} = \frac{1}{k}\sum_{i=1}^{k} x_i, \tag{3.11}$$

e este valor é quantizado utilizando-se um dicionário de escalares $W = \{\sigma_1, ..., \sigma_r\}$. O formato neste caso tem um conceito diferente do usado no método anterior; ele é definido aqui por um vetor do tipo,

$$\hat{\mathbf{x}} = (x_1 - \sigma_m, ..., x_k - \sigma_m), \tag{3.12}$$

onde $\sigma_m \in W$ é o valor quantizado de $\overline{x}$. Este vetor é então codificado utilizando-se um dicionário $Y = \{\mathbf{y}_1, ..., \mathbf{y}_s\}$ que é projetado especialmente para este tipo de vetor, ou seja, cada um dos vetores da seqüência de treinamento tem sua média extraída durante a construção do dicionário. Este método, que é normalmente chamado de *mean/shape VQ*, é ilustrado na Figura 3.5.

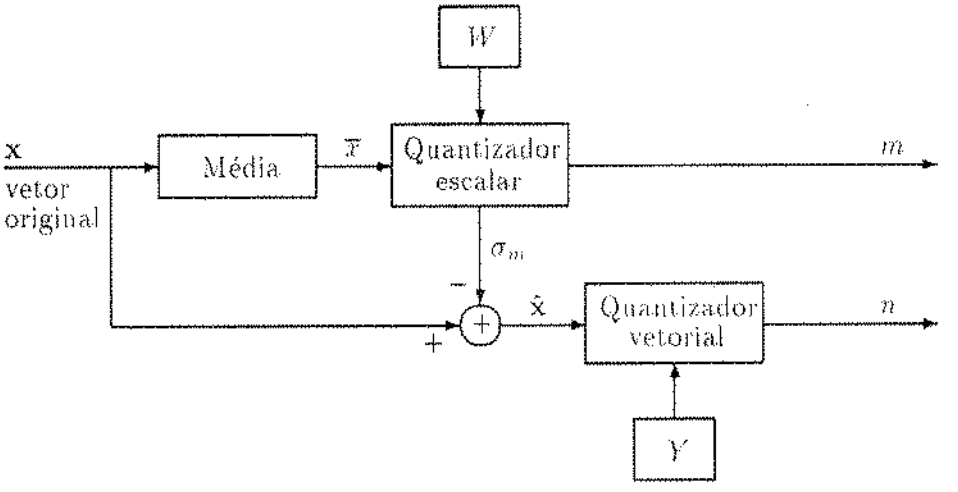


Figura 3.5: *Mean/shape VQ.*

Se $\mathbf{y}_n = (y_1, ..., y_k) \in Y$ é o vetor de código que mais se aproxima de $\hat{\mathbf{x}}$, o decodificador reconstrói o vetor como sendo,

$$\mathbf{x}^* = (y_1 + \sigma_m, ..., y_k + \sigma_m). \tag{3.13}$$

Ao extrair a média dos vetores, os blocos de imagem podem ser codificados de maneira mais eficiente, com o uso de dicionários menores.

Este método é semelhante ao *gain/shape*, e tem a vantagem de codificar a média de uma maneira mais direta do que aquela que é usada para codificar a norma no método anterior, entretanto seu resultado não é ótimo, ou seja, não há garantia de que o par média-formato determinado para reconstruir um vetor seja de fato a melhor opção.

3.3.5 Quantização Vetorial Classificada

Um efeito que surge com freqüência em imagens codificadas por quantização vetorial é a degradação de arestas, especialmente quando se utiliza dicionários pequenos, que não conseguem reproduzir com fidelidade os blocos de imagem que contém componentes de alta freqüência espacial. No intuito de minimizar este problema, Ramamurthi e Gersho propuseram a classificação de cada vetor (bloco

de imagem) em diferentes categorias, dando uma maior prioridade aos que contêm arestas [RG86]. Para tanto, eles separaram os seguintes tipos de blocos:

- sem variação: classe que foi chamada *shade*, contém os blocos cujos valores de pixel não apresentam variação significativa;
- variação suave: classe que foi chamada *midrange*, contém os blocos onde há variação de brilho sem, contudo, possuir componentes de alta freqüência (arestas);
- arestas: são definidas diversas classes para diferentes tipos de arestas, a saber, verticais, horizontais, diagonais a $+45°$ e diagonais a $-45°$, com diferentes classes para diferentes posições e polaridades;
- mixtos: classe que contêm os blocos que possuem componentes de alta freqüência mas que não se encaixam em nenhuma das classes de arestas.

A cada classe c_i é associado um dicionário D_i que pode ser construído pelo algoritmo LBG, mas com a ressalva de utilizar apenas os vetores da seqüência de treinamento que pertencem à respectiva classe. A seqüência de treinamento é dividida com o auxílio de um algoritmo de classificação que também é utilizado para decidir qual dicionário deve ser empregado para codificar cada bloco. O método básico de quantização vetorial classificada é ilustrado na Figura 3.6.

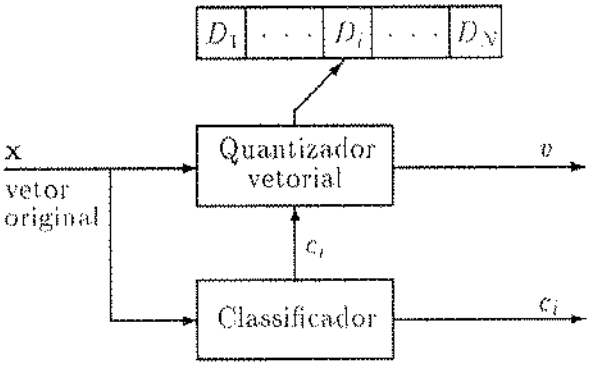


Figura 3.6: Quantização Vetorial Classificada.

Os tamanhos dos dicionários normalmente não são iguais, e determinar qual o melhor tamanho n_i para cada dicionário D_i é, em geral, um problema difícil de resolver. Uma maneira seria testar diferentes combinações de tamanhos até encontrar a melhor opção, mas isto seria muito caro computacionalmente. Uma outra alternativa é determinar analiticamente as combinações entre tamanho n_i e a distorção média d_i para cada classe, levando-se em consideração que os tamanhos ótimos satisfazem à relação [RG86].

$$\frac{p_i d_i}{n_i} = \text{constante} \qquad (i = 1, ..., N). \tag{3.14}$$

onde p_i é a probabilidade de um vetor arbitrário estar na i-ésima classe. A distorção média em classes de blocos que contêm componentes de alta freqüência tende a ser bem maior do que nas

demais classes; por isto, em geral a maior parte dos vetores de código são dedicados a classes de arestas (apesar de, na maioria dos casos, a probabilidade de um bloco não ter aresta ser maior).

Este método permite a codificação de imagens com uma melhor representação de arestas, especialmente quando é dado algum tratamento especial aos blocos que contêm coeficientes de alta freqüência. Além disto, há uma outra vantagem desta técnica, que é a redução significativa do número de comparações para encontrar o vetor de código mais semelhante. Aqui as medidas de distorção são feitas apenas nos dicionários específicos de cada classe, resultando em um custo computacional bem menor do que no Esquema Básico.

3.4 Quantização Vetorial com Memória

A idéia de métodos que fazem uso de memória é explorar a correlação entre vetores com o objetivo de melhorar o desempenho, tendo em vista que normalmente existe uma dependência estatística entre vetores consecutivos. Para tanto, são usados dicionários diferentes para codificar cada vetor. Uma grande variedade de métodos que utilizam esta estratégia pode ser encontrada na literatura (veja por exemplo [Gra84, GBS86, NK88]), e eles podem ser divididos em duas categorias cujas idéias básicas descrevemos a seguir.

3.4.1 Quantizadores Vetoriais Preditivos

Neste tipo de quantizador (também chamado *feedback VQ*) a escolha do dicionário mais adequado para codificar cada vetor é feita baseando-se apenas em vetores já codificados; desta maneira, o decodificador também pode determinar qual dicionário foi usado para codificar cada vetor.

Um quantizador vetorial preditivo pode ser visto como uma máquina de estados onde cada estado e representa um quantizador separado com seu próprio codificador α_e, decodificador β_e e dicionário D_e. Para determinar o estado seguinte em cada passo, é usado um mapeamento F, chamado função de transição de estados, ou seja, após codificar um vetor $\mathbf{x}$ em um estado e, resultando no símbolo de canal $c = \alpha_e(\mathbf{x})$, o estado seguinte é dado por $F(v, c)$. Como o próximo estado depende apenas do símbolo de canal e estado correntes, o decodificador pode sempre determiná-lo sem que seja necessária a transmissão de nenhuma informação adicional.

Se a máquina de estados é finita, então o método é normalmente chamado de *finite-state VQ* (FSVQ), e pode-se observar que um quantizador vetorial sem memória é simplesmente um FSVQ de um único estado. A liberdade de escolha de diferentes quantizadores, baseando-se no passado, permite atingir um desempenho melhor que quantizadores sem memória com a mesma dimensão e média de bits por pixel.

3.4.2 Quantizadores Vetoriais Adaptativos

De uma maneira geral, o que distingue estes quantizadores dos preditivos é que não é usado apenas o passado para selecionar o dicionário seguinte: uma informação adicional deve ser enviada ao decodificador para tanto.

Em um esquema proposto por Goldberg *et al* [GBS86] diferentes dicionários são computados a partir dos vetores da imagem, e tanto a imagem codificada quanto o dicionário são transmitidos ao decodificador. A seqüência de treinamento usada para projetar os dicionários é extraída da própria imagem que está sendo codificada.

A imagem pode ser dividida em regiões retangulares disjuntas, e para cada uma delas um dicionário diferente é projetado independentemente. Para cada região, os vetores representativos do dicionário são transmitidos, seguidos pelos símbolos de canal que representam os vetores da imagem.

O fato de ter que transmitir os dicionários junto com a imagem codificada não acarreta em um aumento significativo da informação a ser transmitida, já que os dicionários usados são muito menores do que nos métodos não adaptativos, resultando em símbolos de canal de menor comprimento para representar os vetores da imagem.

3.5 Conclusões

Neste Capítulo demos uma descrição do Esquema Básico de quantização vetorial e citamos algumas de suas principais variações. Fizemos apenas um resumo introdutório e muito ainda há por ser dito sobre esta técnica que permite combinar bom desempenho com baixo custo computacional. Algumas variações de quantização vetorial sem memória permitem atingir taxas de 0,5 a 1,5 bits/pixel com uma boa qualidade nas imagens reproduzidas, e há também a possibilidade de combiná-la com outros métodos, como por exemplo codificação por transformadas, permitindo atingir taxas de compressão ainda maiores.

A quantização vetorial é um dos principais objetos de estudo nesta dissertação e mais adiante mostraremos alguns resultados experimentais obtidos com diferentes variações desta técnica, e com um esquema de compressão de seqüências de imagens nela baseado.

Leituras Complementares

Esta técnica foi amplamente estudada na última década: em 1980 Linde, Buzo e Gray desenvolveram o algoritmo LBG para projeto de quantizadores ótimos [LBG80]; no mesmo ano, Buzo *et al* [BGGM80] apresentaram algumas aplicações desta, que até então era uma nova técnica, à codificação de voz, indicando variações que podiam também ser utilizadas em codificação de imagens. Algum tempo depois, Gersho [Ger82] fez uma descrição sistemática da estrutura de quantizadores vetorias, indicando propriedades analíticas úteis no estudo da complexidade de codificação e projeto de quantizadores ótimos. Em 1984, uma revisão bastante abrangente foi publicada por Gray [Gra84], contendo a descrição de diversas variações e alguns resultados experimentais. Mais recentemente, Nasrabadi e King [NK88] publicaram outra revisão contendo alguns trabalhos posteriores ao artigo de Gray, inclusive um método que combina quantização vetorial com transformadas. Além das citadas acima, dezenas de outras publicações abrangendo um grande número de variações podem ser encontradas na literatura.

Capítulo 4

Experimentos com Quantização Vetorial

4.1 Introdução

Neste capítulo, listamos alguns resultados experimentais obtidos a partir de métodos baseados em quantização vetorial. Apesar do principal objetivo desta dissertação ser o estudo de técnicas de compressão de seqüências de imagens, também analisamos aqui alguns métodos de codificação intraquadros cujos resultados são importantes para uma melhor compreensão do processo de quantização vetorial utilizado em esquemas de codificação interquadros.

Inicialmente, são examinados os resultados obtidos com um método de compressão intraquadros que utiliza variações de quantização vetorial e DPCM. Em seguida é examinado um esquema de compressão de seqüências de imagens que combina reconstrução condicional de blocos com as técnicas citadas acima. Por fim, é proposto um esquema de quantização vetorial classificada, que foi desenvolvido na tentativa de melhorar a qualidade visual das imagens reproduzidas sem necessáriamente diminuir a taxa de compressão atingida com as outras técnicas.

Os métodos examinados são descritos detalhadamente, mostrando, inclusive, como foram construídos os dicionários e quantizadores escalares utilizados. Também são mostradas algumas imagens reconstruídas a partir dos métodos, juntamente com as imagens de erro, relação sinal/ruído (SNR - ver Apêndice A) e análise dos resultados.

4.2 Codificação Intraquadros com Quantização Vetorial

Nesta seção, examinamos um método de codificação baseado em quantização vetorial que não explora redundância temporal. Entretanto, ele é direcionado de tal maneira que possa ser adaptado a um esquema de codificação de seqüências de imagens com possibilidade de uso em aplicações de tempo real.

4.2.1 Descrição do Método

Devido aos objetivos aqui estabelecidos, o primeiro ponto a ser considerado ao se escolher uma variação de quantização vetorial é a possibilidade de busca rápida. No Capítulo 3, vimos duas

variações que atendem a esta exigência: *tree-searched* e *multistep VQ*. Optamos por *tree-searched VQ*, pois produz resultados superiores aos obtidos por *multistep VQ* [Gra84].

Uma outra estratégia que utilizamos é a separação de média (*mean/shape VQ*), pois evita a necessidade de dicionários excessivamente grandes e permite o uso de técnicas como DPCM para codificação da média. A Figura 4.1 faz uma ilustração do esquema utilizado.

Para codificar um vetor $\mathbf{x}$, calcula-se sua média $\overline{x}$ e a estimativa da média $\tilde{x}$, que é baseada em vetores codificados anteriormente. O erro de predição e é então quantizado usando-se um dicionário de escalares $W = \{\sigma_1, ..., \sigma_r\}$; a partir de seu valor reconstruído e^* e da estimativa da média $\tilde{x}$ obtem-se então o valor reconstruído da média $\overline{x}^*$ que é extraido do vetor original obtendo-se o vetor $\hat{\mathbf{x}}$ que é codificado por quantização vetorial usando-se um dicionário $Y = \{\mathbf{y}_1, ..., \mathbf{y}_s\}$. São transmitidos (ou armazenados) dois símbolos de canal, um que representa o erro de predição da média e outro que representa o vetor $\hat{\mathbf{x}}$. O decodificador obtém $\overline{x}^*$ do mesmo modo que o codificador e soma este valor a cada posição de $\hat{\mathbf{x}}^*$ obtendo assim o vetor reconstruído $\mathbf{x}^*$.

Este método é dividido em três elementos básicos: um quantizador vetorial, um quantizador escalar e uma regra de predição. A seguir damos alguns detalhes sobre cada um desses elementos.

O Quantizador Vetorial

O quantizador vetorial, como já dissemos, é baseado em *tree-searched* e *mean/shape VQ*. O dicionário foi construído a partir de uma seqüência de treinamento contendo imagens com uma grande variedade de tipos de cenas, como pessoas, paisagens, animais e até algumas não extraídas do mundo real (geradas por computação gráfica). Houve a preocupação de escolher imagens pouco semelhantes entre si, permitindo assim uma maior variedade de tipos de blocos.

Durante a construção do dicionário, é extraído o valor médio de brilho de cada bloco da seqüência de treinamento, de modo que todos eles fiquem com média zero (os pixels dos blocos resultantes podem assumir valores negativos). A estrutura utilizada para o dicionário é uma árvore binária cuja raiz é o centróide de toda a seqüência de treinamento. Cada nível da árvore é obtido duplicando-se os nós do nível anterior por meio de uma pequena perturbação e em seguida substituindo-se os nós resultantes pelos centróides dos conjuntos de vetores de treinamento que foram mapeados em cada um deles.

Os mapeamentos são feitos por meio de apenas duas comparações em cada nível da árvore permitindo assim uma busca rápida para a codificação.

O Quantizador Escalar

O projeto do quantizador escalar foi baseado em um método de construção de quantizadores para DPCM descrito por Sharma e Netravali [SN77], do qual daremos aqui uma descrição em linhas gerais.

Baseando-se em testes subjetivos , é definida uma funcão limiar $L(x)$, onde x é a diferença de brilho entre pixels vizinhos, de tal maneira que uma aresta de amplitude x seja indistinguível visualmente de uma de amplitude $x + L(x)$ ou $x - L(x)$.

Uma vez determinada a função $L(x)$, o quantizador Q é projetado de forma que,

$$|Q(x) - x| \leq L_\epsilon(x), \tag{4.1}$$

onde $L_\epsilon(x) = (1 + \epsilon)L(x)$. Quanto menor for a constante ϵ menor será a distorção média do

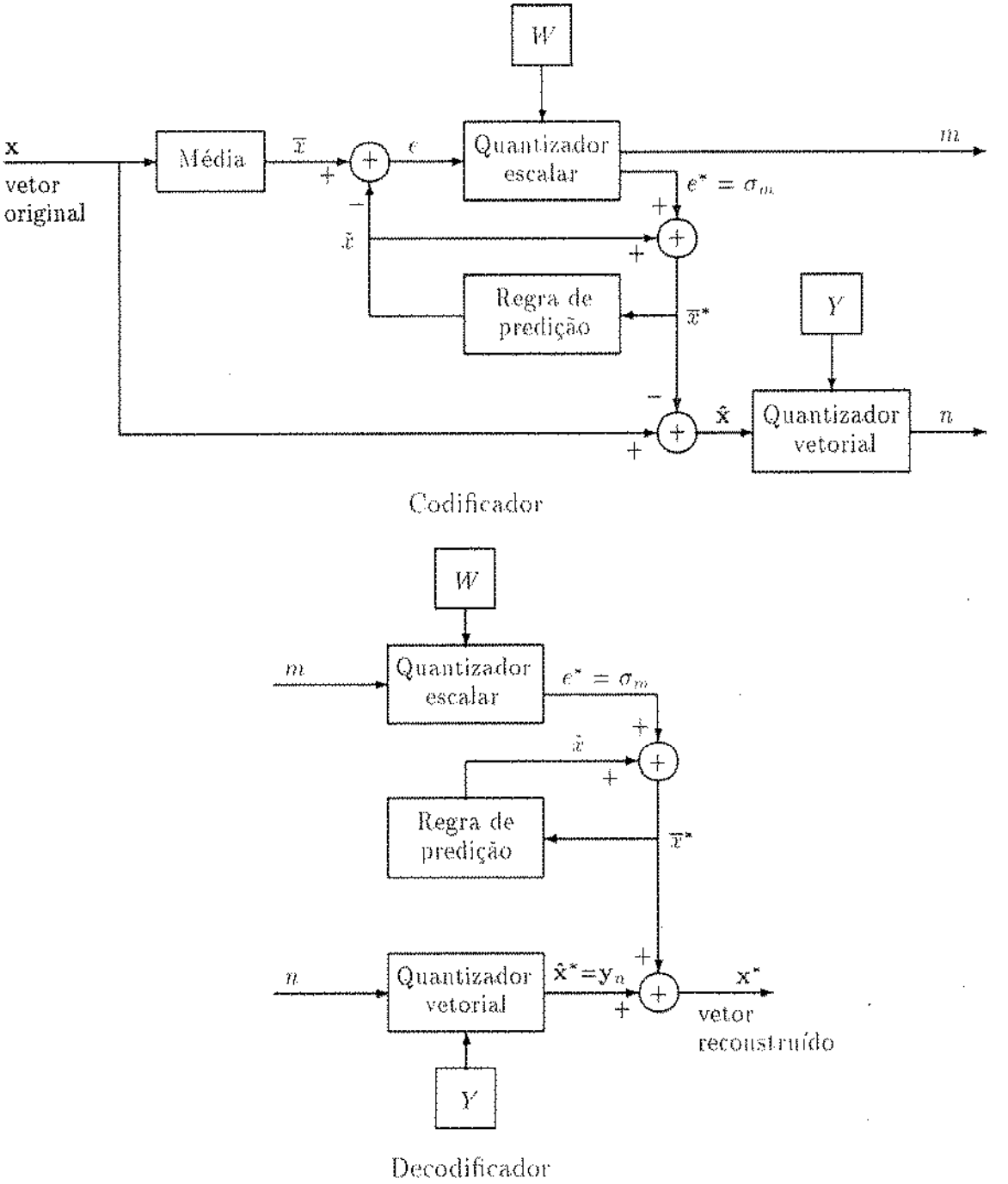


Figura 4.1: Esquema de codificação intraquadros baseado em quantização vetorial e DPCM.

quantizador, mas, por outro lado, maior será o número de níveis de quantização, e conseqüêntemente maior o número de bits necessários para a codificação.

Para determinar os níveis de decisão (d_i) e os níveis de reconstrução (r_i) do quantizador, é feito um zig-zag a 45° abaixo da curva descrita por $L_c(x)$, como é mostrado na Figura 4.2.

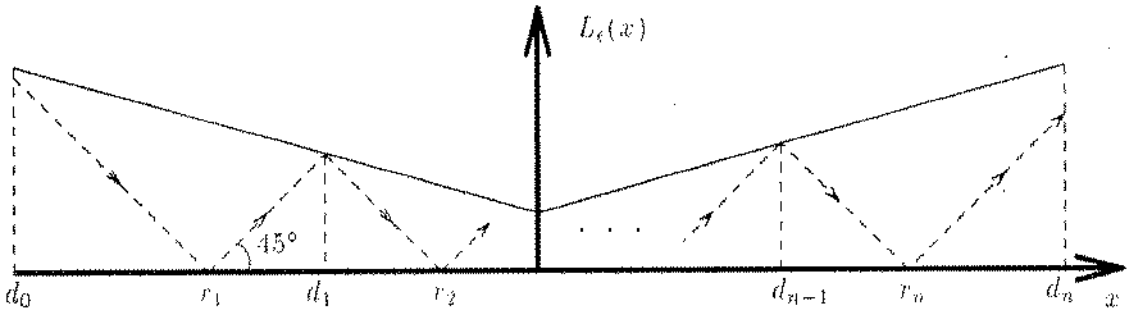


Figura 4.2: Projeto de quantizadores para DPCM.

Por simplicidade, em nossa abordagem fizemos aproximações por retas das funções $L_c(x)$, baseando-se em resultados obtidos por Sharma e Netravali em [SN77].

A Regra de Predição

A regra de predição a ser utilizada deve ser capaz de fazer uma boa estimativa da média de brilho de cada bloco, pois qualquer erro significativo na reprodução deste valor afeta não apenas um pixel, mas todo um bloco da imagem. Portanto, regras de predição unidimensionais não são recomendáveis uma vez que pode haver grandes variações entre médias de blocos vizinhos. Para obter um melhor resultado, utilizamos uma regra de predição bidimensional que utiliza as médias dos blocos das posições mostradas na Figura 4.3.

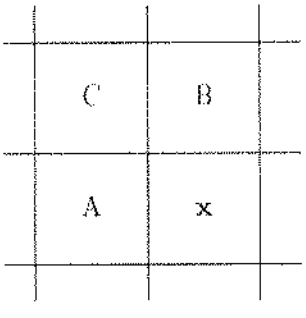


Figura 4.3: Blocos usados pela regra de predição no passo de DPCM.

Supondo-se que m_A, m_B e m_C sejam as médias de brilho dos blocos nas posições A, B e C respectivamente, a estimativa da média do bloco **x** será dada por,

$$\hat{x} = m_A + m_B - m_C. \tag{4.2}$$

Com esta regra de predição é possível atingir alguns resultados satisfatórios utilizando um quantizador de 16 níveis (4 bits) com blocos de 4×4 pixels. Utilizando um quantizador de 32 níveis (5 bits) os resultados são praticamente indistinguíveis dos obtidos com um esquema sem DPCM usando 8 bits para média.

4.2.2 Resultados

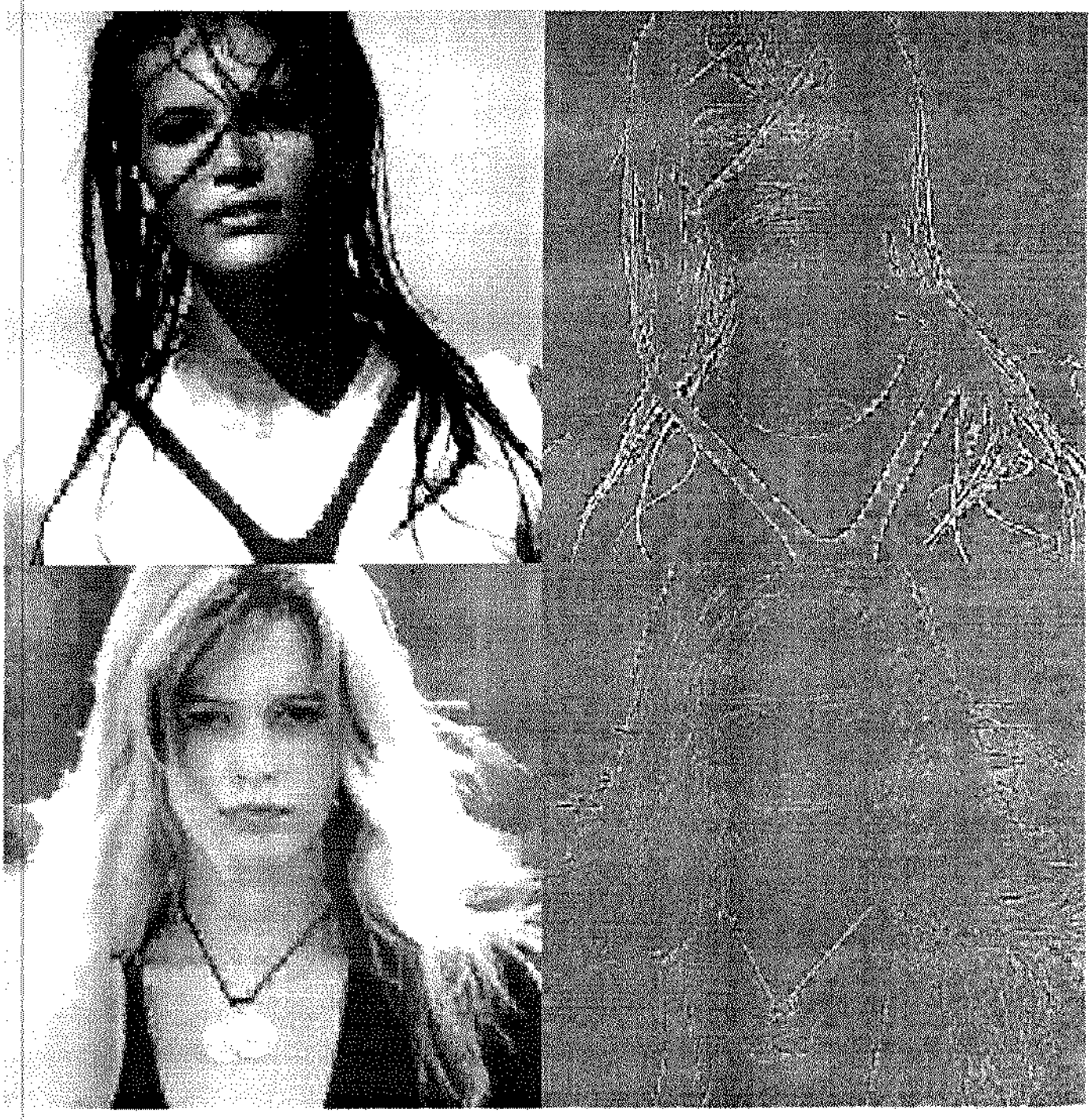
Fizemos testes usando imagens de 256×256 pixels e 256 tons de cinza (8 bits por pixel), que não fazem parte da seqüência de treinamento utilizada na construção dos dicionários. Estas imagens são mostradas na Figura 4.4.

Figura 4.4: Imagens usadas nos testes: Claudia (esquerda) e Wet-girl (direita).

A Figura 4.5 mostra, junto com as imagens de erro, as imagens Wet-girl e Claudia codificadas em blocos de 4×4 pixels usando 5 bits para média e 8 bits para o identificador do vetor de código, ou seja, utilizamos um dicionário com 256 vetores de código diferentes. Com isto obtemos uma média de 0,81 bits/pixel, o que representa uma compressão de aproximadamente 9,85:1. Esta é uma taxa relativamente alta de compressão, entretanto há visível degradação da imagem, especialmente na imagem Wet-girl que tem uma maior quantidade de arestas de grande amplitude.

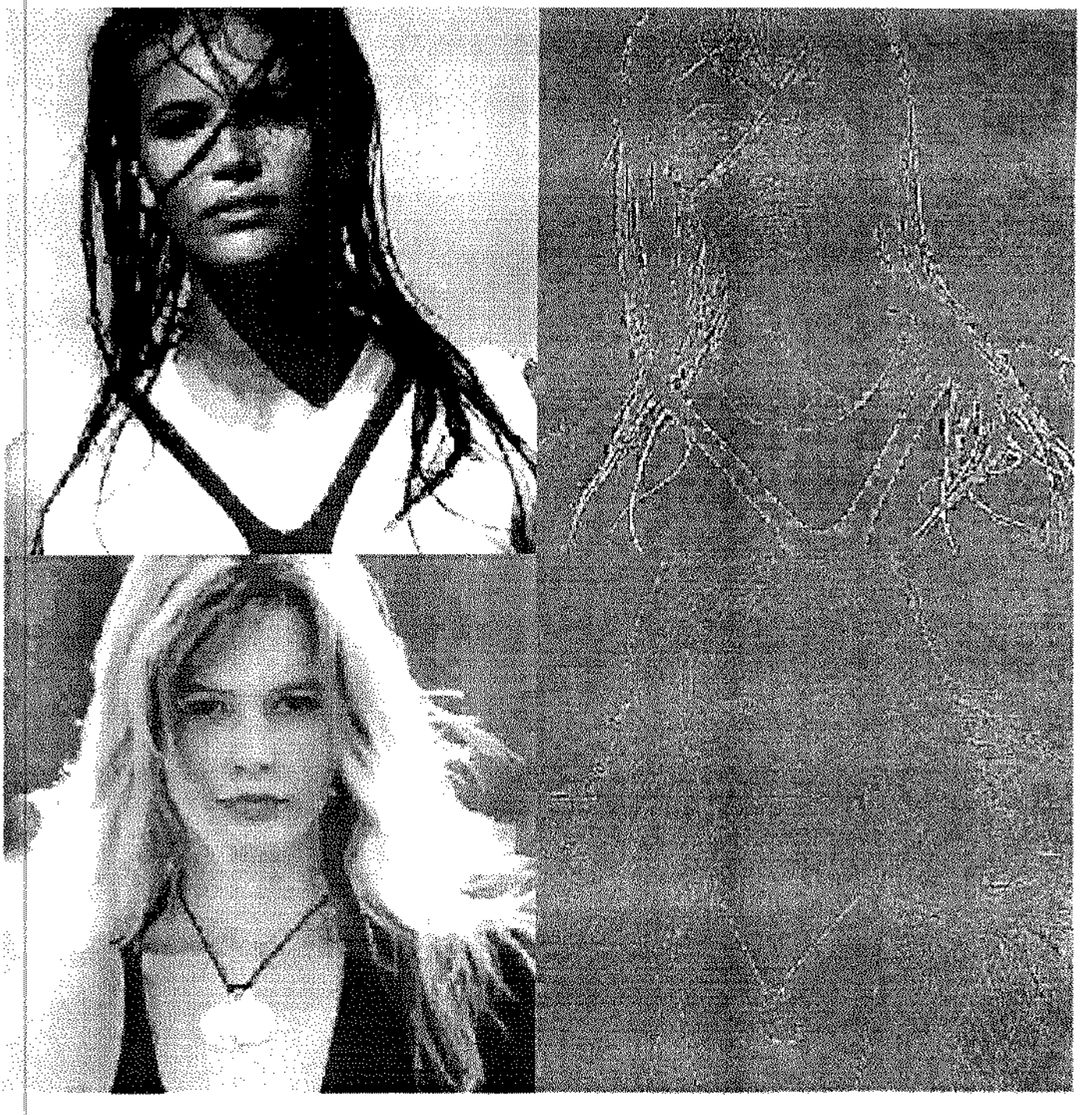
Na Figura 4.6, podemos observar uma melhora substancial na qualidade das imagens. Elas foram codificadas com 5 bits para média e 11 para o vetor de código (dicionário de 2048 vetores) mantendo assim uma média de 1 bit/pixel, o que implica em uma compressão de 8:1, que ainda é uma boa taxa e sem uma degradação tão significativa da imagem.

Podemos observar que uma das maiores dificuldades encontradas por este método é reprodução



SNR: Wet-girl - 22,42 dB; Claudia - 31,17 dB.

Figura 4.5: Imagens codificadas à média de 0,81 bits/píxel.

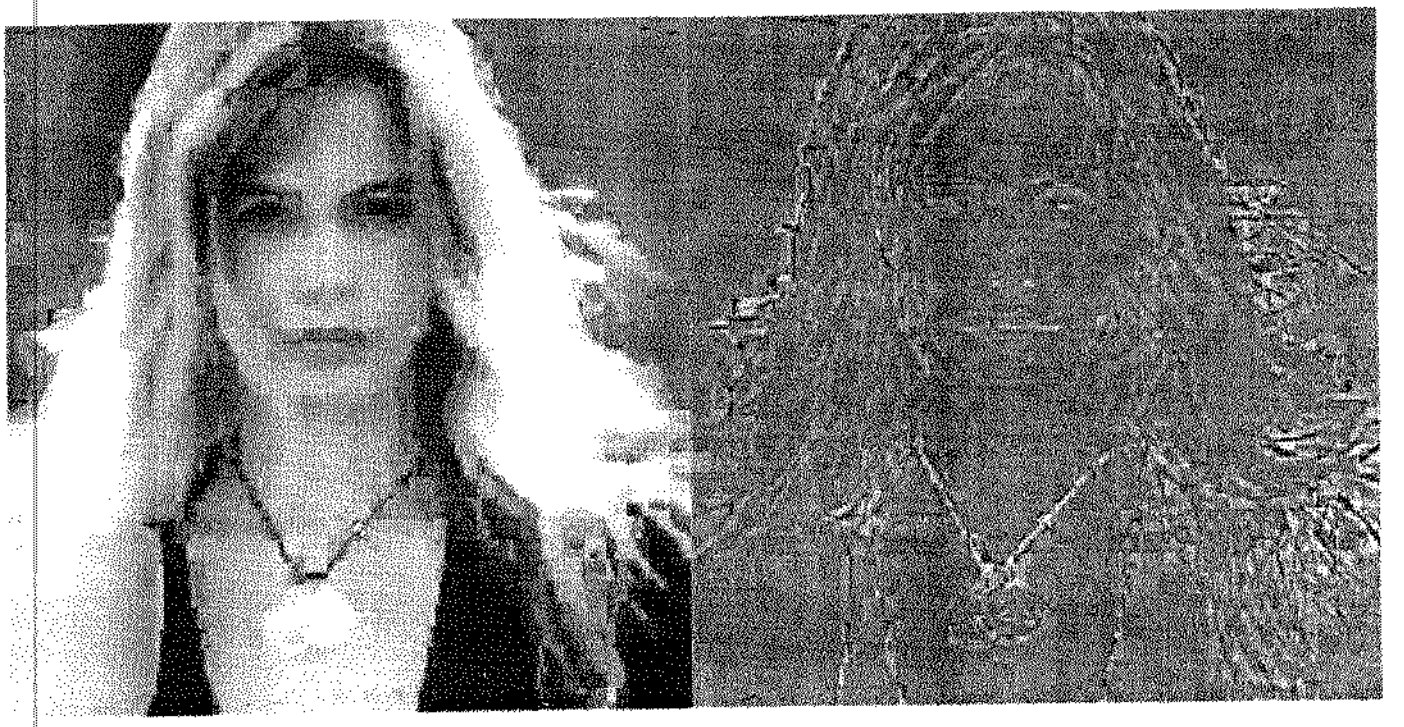


SNR: Wet-girl - 23,43 dB; Claudia - 32,80 dB.

Figura 4.6: Imagens codificadas à média de 1 bit/pixel.

de arestas de grande amplitude, especialmente quando são utilizados dicionários pequenos. O desempenho com a imagem Claudia é bem melhor do que com a imagem Wet-girl, e isto deve-se ao fato de que a primeira é constituída em sua maior parte por arestas suaves, o que não acontece com a segunda, que apresenta grande quantidade de arestas de grande amplitude. Uma tentativa de dar uma melhor representação para este tipo de aresta pode ser feita utilizando-se quantização vetorial classificada como veremos posteriormente.

Em alguns casos podemos utilizar blocos maiores de modo a atingir taxas de compressão consideravelmente mais altas. A Figura 4.7 mostra a imagem Claudia codificada com blocos de 8×8 usando 5 bits para média e 13 para o vetor de código (dicionário de 8192 vetores), ou seja, uma média de 0.28 bits/pixel que representa uma taxa de compressão de 28,57:1. A degradação da imagem, como é de se esperar, é bem maior que nos casos anteriores, entretanto, esta variação pode ser utilizada em conjunto com outros métodos sem grandes prejuízos para a qualidade das imagens reproduzidas, como veremos na seção seguinte.



SNR: 28,50 dB.

Figura 4.7: Imagem codificada à média de 0,28 bits/pixel.

4.3 Codificação Interquadros com Quantização Vetorial

Nesta seção vamos examinar um esquema de compressão de seqüências de imagens que foi desenvolvido por Geus [Geu90] e chamado de MDPT/VQ (*Movement Detected, Progressive Transmission*

with Vector Quantization). O método utiliza quantização vetorial, DPCM e um tipo de reconstrução condicional de blocos. Em resumo ele é constituído das seguintes estratégias:

Detecção de Movimento: é feita por blocos de imagem: um método adequado deve ser utilizado para minimizar o número de blocos a serem codificados sem afetar significativamente a qualidade da imagem reproduzida. Os blocos onde não é detectado movimento são simplesmente copiados do quadro anterior na decodificação. Um exemplo de método que dá bons resultados é medir o erro médio quadrático (ver Apêndice A) entre blocos na mesma posição em quadros consecutivos, e verificar se estão abaixo ou acimá de um limiar convenientemente escolhido. Este limiar é usado para evitar que blocos sejam detectados com movimento apenas por oscilação de ruído temporal.

Descrição Prévia do Bloco: uma vez detectado movimento em um bloco, é utilizado um método eficiente de codificação para dar uma primeira descrição do mesmo. Várias técnicas poderiam ser utilizadas com este objeivo, entre elas, quantização vetorial e codificação por transformadas. O método que é descrito aqui utiliza quantização vetorial; no capítulo seguinte veremos uma outra abordagem utilizando a transformada do cosseno (DCT).

Transmissão Progressiva: esta estratégia é baseada em um trabalho de Dreizen [Dre87], e consiste em transmitir a informação de atualização das imagens recontruídas ao longo de mais de um quadro da seqüência, ou seja, após ser feita a descrição prévia do bloco, continua-se a transmitir informação adicional até que a distorção esteja abaixo de um certo limiar. A medida de distorção usada neste caso também foi o erro médio quadrático.

4.3.1 Variantes de MDPT/VQ

O método descrito por Geus possui duas variantes principais: *Block Eight* baseada em blocos de 8×8 e *Block Four* baseada em blocos de 4×4. A seguir damos uma breve descrição destas variantes.

Block Eight

Nesta estratégia a descrição prévia dos blocos é dada por um vetor de código que mapeia blocos de 8×8, entretanto a detecção de movimento é feita separadamente em cada um dos quadrantes dos blocos; se apenas um quadrante for detectado com movimento, a descrição prévia será feita por um vetor de código que mapeia blocos de 4×4. A seqüência de passos de transmissão progressiva, se não for detectado novo movimento em um bloco, é a seguinte:

1. No momento em que o movimento é detectado, o identificador (símbolo de canal) de um vetor de código para blocos de 8×8 é transmitido.

2. No quadro seguinte, os dois quadrantes correspondentes a uma das diagonais do bloco reconstruído são testados com respeito à distorção em relação ao quadro corrente: se a distorção nestes quadrantes é maior que um certo limiar, os identificadores de dois vetores de código representando os blocos de 4×4 envolvidos são transmitidos; se a distorção total do bloco reconstruído com relação ao original estiver abaixo do limiar, o algoritmo avança para o passo de DPCM.

3. No próximo quadro, os dois quadrantes correspondentes à outra diagonal são testados como no passo anterior, e segue-se o mesmo procedimento.

4. No quadro seguinte, cada quadrante é testado em ordem como nos passos anteriores; se a distorção estiver abaixo do limiar, o processo é encerrado; caso contrário é feita uma reconstrução residual baseada em DPCM.

5. O passo 4 é repetido até que a distorção fique abaixo do limiar (cada quadrante é tratado separadamente).

O quantizador vetorial utilizado também usa as estratégias de *tree-searched* e *mean/shape VQ*. A média de cada bloco não é codificada integralmente, mas sim a diferença entre a do quadro corrente e a do anterior. Geus utilizou 3 bits para codificar diferenças entre -13 e 10; diferenças maiores foram codificadas integralmente, com 8 bits. A medida de distorção usada foi o erro médio quadrático. Os dicionários utilizados para blocos de 8×8 tinham 512 vetores de código (9 bits) e para blocos de 4×4 tinham 256 vetores (8 bits). O passo de DPCM residual é feito a 1 bit por pixel, ou seja, 16 bits bits para cada bloco de 4×4 que indicam adição ou subtração, em cada pixel, de um fator de 6 níveis de brilho.

As taxas de compressão atingidas com o *Block Eight* dependem muito do tipo de cena e quantidade de movimento nas imagens. Em seqüências típicas de teleconferência (com movimentos de mãos, ombros e rosto) foi possível atingir taxas entre 20:1 e 40:1 que são relativamente altas; entretanto a qualidade das imagens reconstruídas ainda deixa um pouco a desejar, muitos detalhes de regiões em movimento são perdidos. Nós introduzimos algumas modificações neste método para tentar minimizar estas falhas: os resultados que obtivemos serão mostrados mais adiante.

Block Four

O objetivo desta estratégia é reproduzir imagens com melhor qualidade, abrindo mão das altas taxas de compressão. Neste caso as imagens são divididas em blocos de 4×4 ao invés de 8×8. A descrição prévia do bloco é feita, portanto, por um vetor de código que mapeia blocos de 4×4. Se não for detectado novo movimento, a seqüência de passos de transmissão progressiva é a seguinte:

1. No momento em que o movimento é detectado o identificador de um vetor de código para o bloco de 4×4 correspondente é transmitido.

2. No quadro seguinte, é calculada a distorção do bloco reconstruído com respeito ao bloco corrente. Se estiver maior que um limiar pré-estabelecido, passa-se para uma reconstrução residual baseada em DPCM; se a distorção estiver abaixo do limiar nenhuma ação é feita.

3. No quadro seguinte, a distorção é mais uma vez medida com relação ao bloco corrente, mas desta vez a comparação é feita com um limiar menor; se a distorção estiver acima do limiar, passa-se para a reconstrução residual por DPCM; se estiver abaixo encerra-se o processo.

4. O passo anterior é repetido até que a distorção esteja abaixo do limiar.

O método de quantização vetorial usado é o mesmo adotado para o *Block Eight*, inclusive no que diz respeito à codificação da média.

As taxas de compressão atingidas com este esquema estão entre 10:1 e 30:1: não são tão altas quanto as atingidas pelo *Block Eight*, mas a qualidade das imagens reproduzidas é significativamente superior, apesar de ainda ser visível alguma degradação, especialmente nas arestas.

4.3.2 Mudanças no MDPT/VQ

De início, voltamos nossa atenção para a tentativa de melhorar a qualidade das imagens reproduzidas pelo *Block Eight*. Para isso fizemos uso da experiência obtida com testes feitos com o método de quantização vetorial que foi descrito na Seção 4.2. Não foi feita nenhuma mudança estrutural no esquema, tentamos melhorar apenas a parte de quantização vetorial, ou seja, utilizando dicionários melhores e codificando a média de maneira mais eficiente.

Os Dicionários

A principal falha existente nos dicionários utilizados por Geus está na seqüência de treinamento usada para construí-los. Ela foi extraída das mesmas seqüências de imagens usadas nos testes. Como nestas seqüências as imagens tendem a ser muito semelhantes entre sí, os dicionários resultantes tornaram-se um tanto pobres, e ao mesmo tempo especializados nas imagens que os geraram, de modo que os resultados não chegaram a ser tão ruins. Entretanto, quando são codificadas imagens que não fizeram parte da seqüência de treinamento, a qualidade da reprodução é visivelmente inferior. Para tornar o método mais robusto é necessário utilizar um dicionário que seja capaz de reproduzir bem a diferentes tipos de imagens, e é por isto que optamos por uma seqüência de treinamento bastante longa e contendo uma grande variedade de tipos de imagens. Ao codificar as imagens usadas por Geus com este novo dicionário, houve uma pequena queda na qualidade, o que é de se esperar, já que ele não é especializado em nenhum tipo de imagem. No entanto quando codificamos outras imagens, houve uma melhora significativa na qualidade, o que indica que este dicionário pode ser usado para uma quantidade bem mais abrangente de imagens.

Outra modificação que fizemos foi quanto ao tamanho. Dicionários maiores implicam, em princípio, em uma menor taxa de compressão, já que os identificadores dos vetores de código tornam-se maiores. Entretanto, se a descrição prévia do bloco for melhor, a tendência é haver menos passos de transmissão progressiva, o que compensa parte da perda. O dicionário de blocos de 8×8 foi aumentado de 512 (9 bits) para 8192 vetores (13 bits), e o de blocos de 4×4 foi aumentado de 256 (8 bits) para 2048 vetores (11 bits). Isto acarretou uma queda na taxa de compressão de pouco mais de 10%, mas com uma melhora significativa na qualidade das imagens reproduzidas.

Codificação da Média

Para codificação da média de cada bloco, Geus utilizou DPCM com uma regra de predição que leva em consideração o bloco na mesma posição no quadro anterior; o erro de predição, isto é, a diferença entre as duas médias, era então codificada com um quantizador de 3 bits (8 níveis). Normalmente, existe uma grande relação entre as médias dos blocos de quadros contíguos, de modo que o erro de predição tende a ser pequeno; mas podem ocorrer erros maiores, e para estes casos Geus preferiu codificar o erro integralmente com 8 bits. Como deve haver mais um bit para especificar como o erro foi codificado, ocupa-se no mínimo 4 bits e no máximo 9 bits para o código da média.

Com os resultados que obtivemos com o método descrito na Seção 4.2, sentimo-nos encorajados a utilizar os quantizadores escalares ali descritos também no esquema MDPT. Ao utilizar um quantizador de 4 bits para codificar o erro de predição da média, obtivemos um pequeno ganho na taxa de compressão, sem nenhum prejuízo visível na qualidade das imagens reproduzidas. O aumento na taxa de compressão foi pequeno, mas uma outra vantagem desta alteração é que ela

permite gerar um código mais limpo e elegante, uma vez que a média é codificada sempre com 4 bits.

Resultados

Nos testes aqui descritos utilizamos as mesmas seqüências de imagens que foram usadas por Geus. Os quadros 31 e 63 de duas destas seqüências são mostrados na Figura 4.8. São seqüências com 64 quadros de 256×256 pixels e 256 níveis de cinza. A seqüência *hand3* contém bastante movimento e foi usada para testar a reconstrução do fundo após a passagem de um objeto em movimento, e a seqüência *talk2* é uma cena típica de teleconferência, e contém uma quantidade moderada de movimento (rosto e ombros).

Em primeiro lugar, testamos a versão original do *Block Eight* mas com os dicionários gerados pela nova seqüência de treinamento. Alguns resultados são listados na Tabela 4.1.

Seqüência	Taxa Média		Taxa Mínima	
	Compressão	SNR (dB)	Compressão	SNR (dB)
hand3	26.26:1	26,99	24.12:1	26,32
talk2	37.96:1	28,50	29,54:1	27,53

Tabela 4.1: Resultados com a versão original do MDPT/VQ (Block Eight).

As taxas de compressão atingidas são praticamente as mesmas obtidas por Geus, entretanto houve uma pequena queda na qualidade das imagens reproduzidas, uma vez que a seqüência de treinamento usada para construir os dicionários não foi extraída das seqüências de imagens usadas nos testes.

Após mudarmos os tamanhos dos dicionários e a maneira de codificação da média, obtivemos os resultados que são mostrados na Tabela 4.2.

Seqüência	Taxa Média		Taxa Mínima	
	Compressão	SNR (dB)	Compressão	SNR (dB)
hand3	23,49:1	28,28	21,54:1	27,52
talk2	34,33:1	30,02	27,20:1	29,00

Tabela 4.2: Resultados com a versão modificada do MDPT/VQ (Block Eight).

Os quadros 31 e 63 das seqüências resultantes são mostrados nas Figuras 4.9 e 4.10, juntamente com as imagens de erro. O ganho em qualidade é significativo: há uma melhor representação das arestas e as fronteiras entre os blocos tornou-se menos visível, mas ainda se pode perceber alguma degradação em certas regiões das imagens. O aumento do número de vetores de código dos dicionários resultou em um ganho em qualidade mais significativo do que a perda quanto à taxa de compressão; entretanto, há um inconveniente em aumentar indiscriminadamente o tamanho dos dicionários, que é a quantidade de memória necessária para armazená-los, especialmente para blocos de 8×8. Aumentar o tamanho dos dicionários também não é uma boa alternativa para melhorar

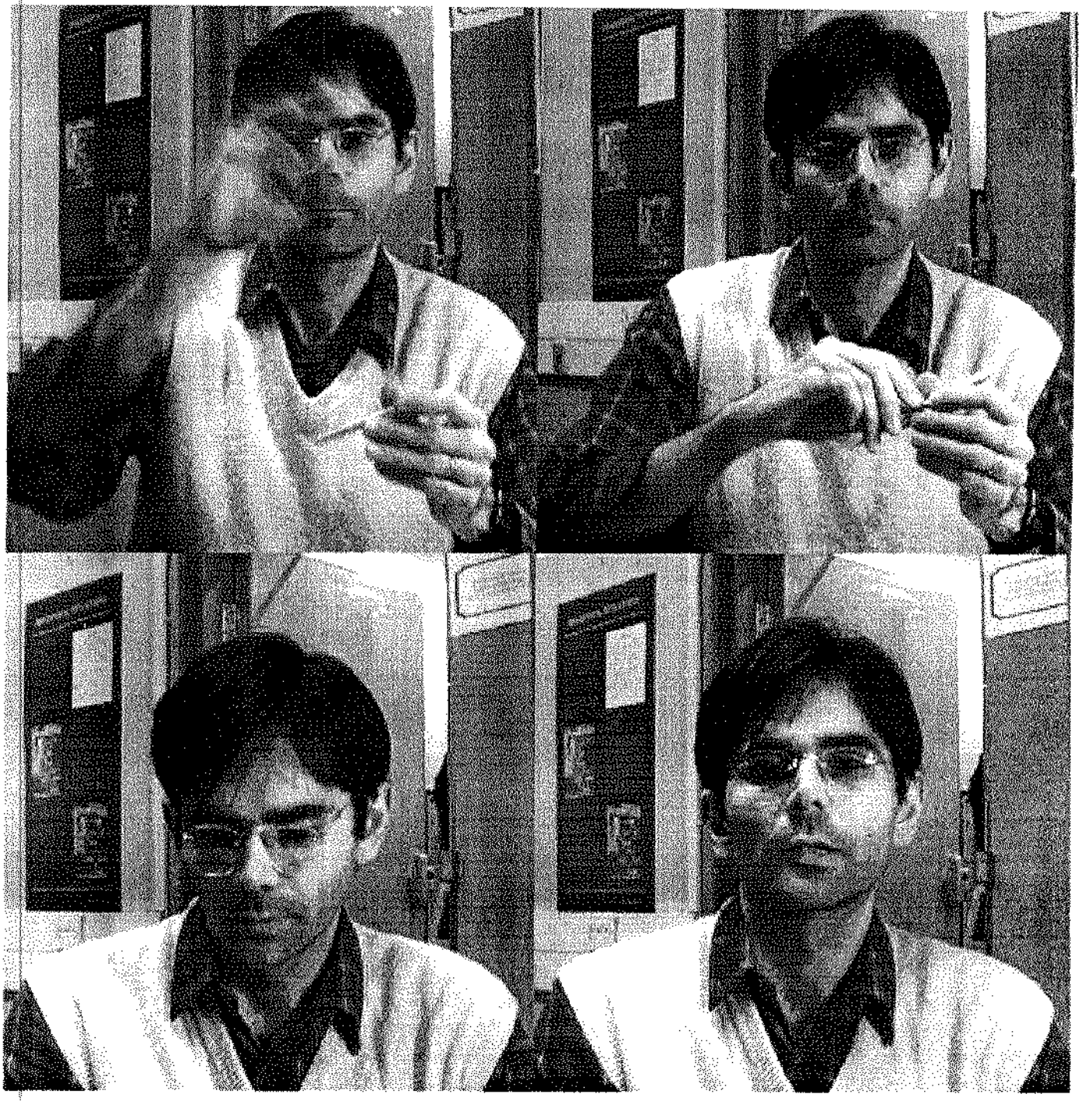
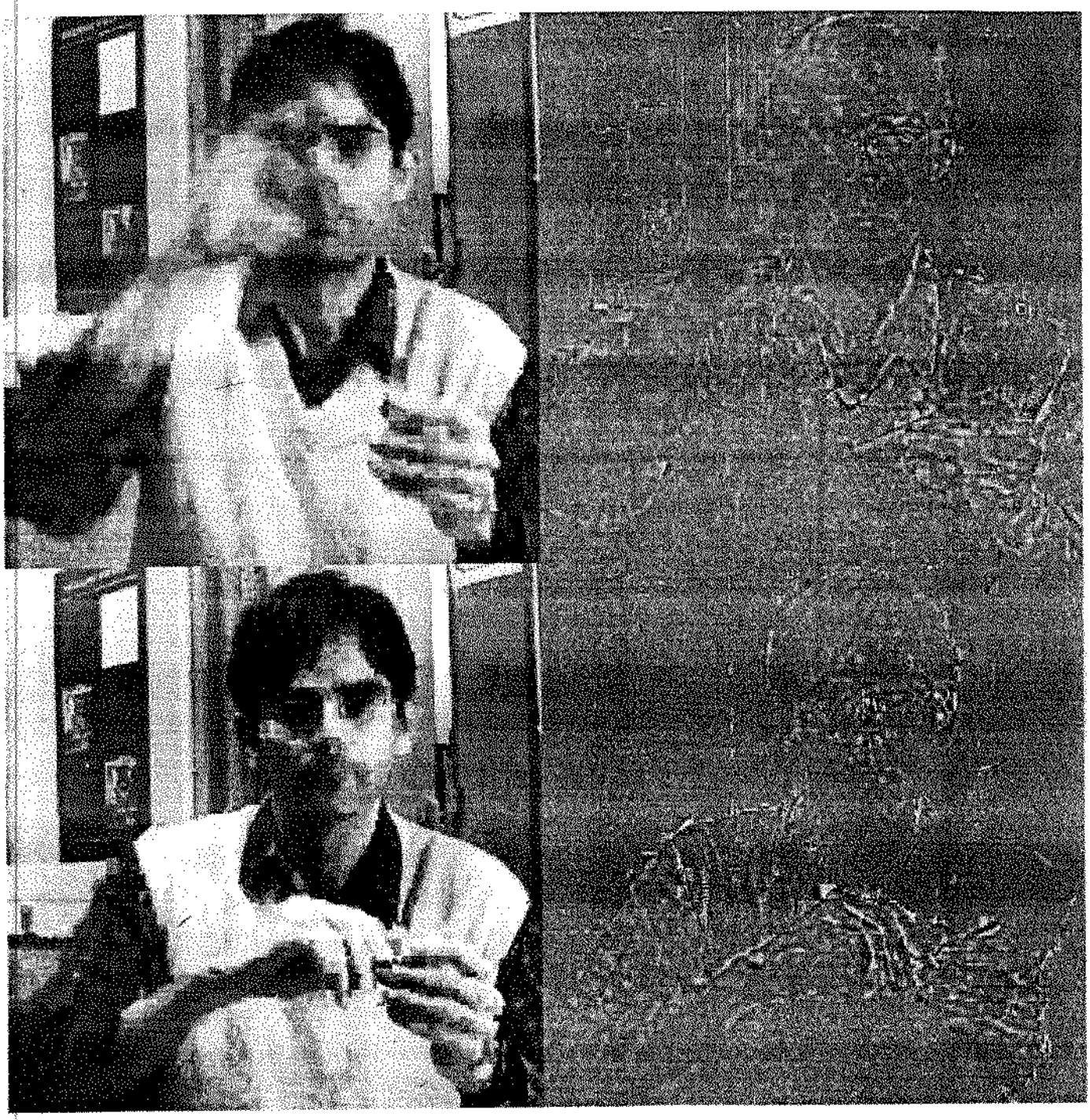
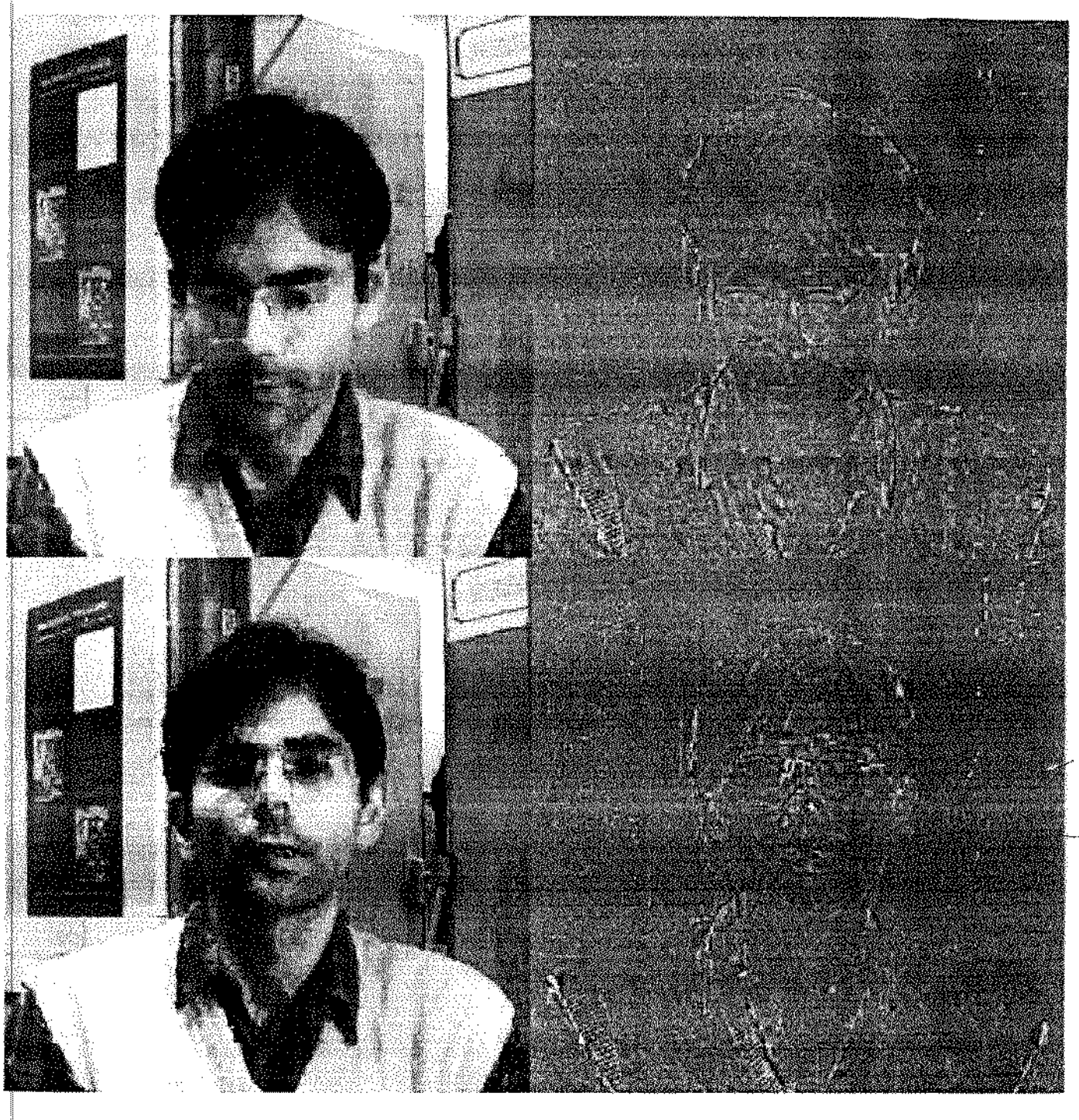


Figura 4.8: Seqüências originais: *hand3* (acima) e *talk2* (abaixo).



SNR: quadro 31 - 28.79 dB; quadro 63 - 28,26 dB.

Figura 4.9: Seqüência *hand3* codificada por MDPT/VQ modificado.



SNR: quadro 31 - 29,23 dB; quadro 63 - 29,60 dB.

Figura 4.10: Seqüência *talk2* codificada por MDPT/VQ modificado.

o desempenho de um esquema como o *Block Four*, pois isto implicaria em uma queda na taxa de compressão e, em alguns casos, os resultados obtidos com este método já estão bem próximos do mínimo aceitável para transmissão através de redes como *Ethernet* [Geu90], como mostra a Tabela 4.3. Tendo isto em mente, pensamos em usar uma outra técnica que fosse capaz de melhorar a qualidade das imagens reproduzidas sem que necessariamente houvesse uma queda na taxa de compressão. Foi a partir daí que surgiu a idéia de usar quantização vetorial classificada. Na seção seguinte, é descrita a estratégia desenvolvida e são mostrados alguns resultados, juntamente com as conclusões obtidas sobre esta técnica.

Seqüência	Taxa Média		Taxa Mínima	
	Compressão	SNR (dB)	Compressão	SNR (dB)
hand3	12,91:1	30,40	10,58:1	29,74
talk2	23,07:1	31,72	16,29:1	31,05

Tabela 4.3: Resultados obtidos com o *Block Four*.

4.4 Quantização Vetorial Classificada

O principal objetivo desta técnica, como vimos na Seção 3.3.5, é dar uma melhor representação para as arestas, por meio da separação de diferentes tipos de blocos dando maior ênfase àqueles que contêm componentes de alta freqüência espacial. Em uma abordagem feita por Ramamurthi e Gersho para blocos de 4×4 [RG86], foram definidas 31 classes diferentes, sendo que 28 delas foram dedicadas a arestas. Em nossa abordagem, preferimos definir um classificador mais simples, com um número menor de classes, pois precisamos manter a possibilidade de compressão e descompressão de seqüências de imagens em tempo real. A seguir é dada uma descrição da versão básica do classificador que foi desenvolvido.

4.4.1 O Classificador

Nós definimos um total de 12 classes que são ilustradas na Figura 4.11. A classe *plano* é destinada aos blocos que não possuem variação significativa no valor de brilho de seus pixels; a classe *variação suave* contém os blocos que possuem variação de baixa intensidade, ou seja, não contêm componentes de alta freqüência espacial; na classe *textura*, os blocos são compostos quase que exclusivamente por componentes de alta freqüência, mas não chegando a configurar nenhum tipo de aresta; a classe *aresta complexa* destina-se a blocos que contém arestas irregulares ou múltiplas; as classes de arestas simples são em um total de 8, divididas em horizontais, verticais, diagonais positivas e diagonais negativas com as diferentes polaridades. O algoritmo de classificação que desenvolvemos é bastante simples e não totalmente livre de falhas, entretanto baseando-se nos resultados obtidos, podemos garantir que o número de blocos classificados erradamente é muito pequeno e não chega a prejudicar o desempenho do método.

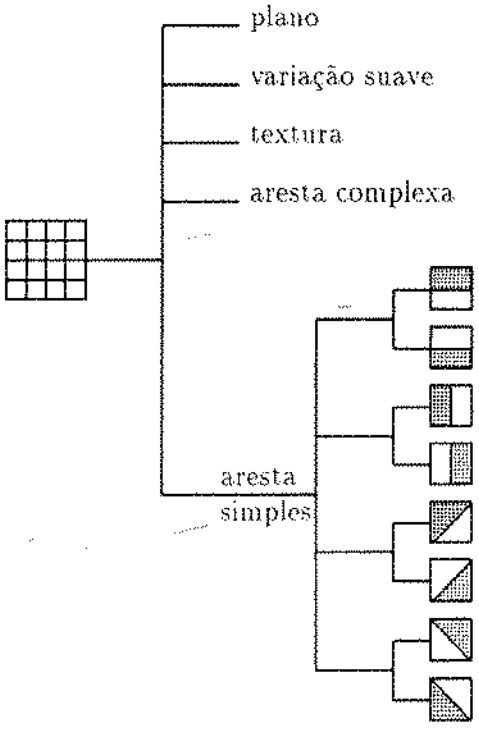


Figura 4.11: Classificação.

O Algoritmo de Classificação

O algoritmo usa um mecanismo semelhante ao descrito por Ramamurthi e Gersho [RG86], mas os procedimentos são diferentes. Ele é dividido em duas fases: na primeira, é feita a atualização de uma série de contadores, baseando-se em duas tabelas que descrevem a configuração do bloco; a segunda é a árvore de decisão, que chega ao tipo do bloco baseando-se nos valores dos contadores. Nos ítens abaixo descrevemos cada uma destas fases para o caso de blocos de 4×4.

Atualização dos Contadores

As tabelas a que nos referimos são na verdade matrizes que chamaremos de D_h e D_v, de tamanhos 4×3 e 3×4 respectivamente, que são preechidas com informações sobre as diferenças de brilho entre pixels vizinhos na horizontal e na vertical. São definidos dois limiares convenientes L_1 e L_2 de tal maneira que os valores nas matrizes são dados por:

$$D_h(i,j) = \begin{cases} -2 & \text{se} & & p(i,j) - p(i,j+1) & < -L_2 \\ -1 & \text{se} & -L_2 \leq & p(i,j) - p(i,j+1) & < -L_1 \\ 0 & \text{se} & -L_1 \leq & p(i,j) - p(i,j+1) & \leq L_1 \\ 1 & \text{se} & L_1 < & p(i,j) - p(i,j+1) & \leq L_2 \\ 2 & \text{se} & L_2 < & p(i,j) - p(i,j+1) & \end{cases} \tag{4.3}$$

para $i = 0, 1, 2, 3$; $j = 0, 1, 2$;

$$D_v(i,j) = \begin{cases} -2 & \text{se} \quad p(i,j) - p(i+1,j) < -L_2 \\ -1 & \text{se} \quad -L_2 \leq p(i,j) - p(i+1,j) < -L_1 \\ 0 & \text{se} \quad -L_1 \leq p(i,j) - p(i+1,j) \leq L_1 \\ 1 & \text{se} \quad L_1 < p(i,j) - p(i+1,j) \leq L_2 \\ 2 & \text{se} \quad L_2 < p(i,j) - p(i+1,j) \end{cases} \tag{4.4}$$

para $i = 0, 1, 2$; $j = 0, 1, 2, 3$; onde $p(i,j)$ é o valor de brilho do pixel na i-ésima linha e na j-ésima coluna do bloco. O limiar L_1 é usado para detectar pequenas variações e seu valor fica em torno de 3 para imagens de 256 tons de cinza; o limiar L_2 é usado para detectar arestas e seu valor fica em torno de 12.

Uma vez definidas as matrizes D_h e D_v, passa-se para a atualização dos contadores: h_{-2}, h_{-1}, h_1 e h_2, contam a quantidade de $-2, -1, 1$ e 2, respectivamente, em D_h; analogamente v_{-2}, v_{-1}, v_1 e v_2, contam estes valores em D_v.

Existe ainda um contador tx que é usado para contar pontos onde há grande diferença de brilho, mas que não chegam a definir arestas por estarem isolados; se o número de tais pontos for suficientemente grande o bloco é classificado como *textura*, uma classe que existe para evitar que blocos com grande variação sejam classificados erradamente como arestas. Existem também os contadores c_h e c_v, que indicam os componentes horizontal e vertical de uma possível aresta. c_h e c_v são incrementados no máximo uma vez para cada linha de D_h ou coluna de D_v.

A Árvore de Decisão

As decisões são feitas baseando-se na comparação dos valores dos contadores com uma série de limiares. O primeiro teste verifica se o bloco possui algum tipo de acidente (componente de alta freqüência espacial). O bloco é considerado com acidente se:

$$c_h + c_v \geq L_{ar}. \tag{4.5}$$

Se o bloco não tem acidente, ele é considerado *variação suave* se:

$$h_{-1} + h_1 + v_{-1} + v_1 \geq L_{va}. \tag{4.6}$$

Caso contrário o bloco será considerado *plano*. Observe que os contadores h_{-1}, h_1, v_{-1} e v_1 poderiam, neste caso, ser substituídos por um único contador, entretanto a existência dos quatro contadores permite a possibilidade de, se for preciso, dividir a classe *variação suave* em diferentes subclasses representando diferentes orientações de variação. Isto pode ser útil no caso de lidarmos com blocos maiores.

No caso do bloco ter acidente ele será considerado *textura* se:

$$tx \geq L_{tx}; \tag{4.7}$$

e aresta complexa se:

$$\begin{gathered} tx < L_{tx} \\ \text{e} \quad \max(v_{-2}, v_2) + \max(h_{-2}, h_2) - c_h - c_v \geq L_{ar} \\ \text{ou} \quad \min(v_{-2}, v_2) + \min(h_{-2}, h_2) \geq L_{ar}; \end{gathered} \tag{4.8}$$

onde L_{ar} é um valor mínimo para que se possa definir uma aresta, ou seja, se a expressão acima é verdadeira o bloco pravavelmente conterá mais de uma aresta. Caso contrário, o bloco é considerado aresta simples e neste caso a orientação da aresta é determinada baseando-se na tangente do ângulo de inclinação, que é estimado por:

$$\tan(\alpha) = \frac{c_v}{c_h}. \tag{4.9}$$

Uma vez determinado o ângulo de inclinação, as polaridades horizontal e vertical, são consideradas positivas se, respectivamente:

$$v_2 > v_{-2} \quad e \quad h_2 > h_{-2}. \tag{4.10}$$

Caso contrário são consideradas negativas. Baseando-se nestes resultados, o bloco será classificado em uma das oito classes de arestas simples.

Os valores dos limiares foram determinados experimentalmente por meio de exaustivos testes. Eles são diretamente relacionados com o tamanho dos blocos. Para blocos de 4×4, nós chegamos aos seguintes valores:

$$L_{ar} = 3, \quad L_{va} = 12 \quad e \quad L_{tc} = 6. \tag{4.11}$$

A Figura 4.12 mostra os dicionários que foram construídos para as diferentes classes usando este algoritmo, e a Tabela 4.4 mostra a porcentagem de blocos da seqüência de treinamento que foram classificados em cada classe e exemplos de possíveis tamanhos para os dicionários resultantes. O fato de estarmos usando *tree-searched VQ* tira bastante da flexibilidade quanto à escolha destes tamanhos (todos devem ser potências de 2), de modo que eles não são ótimos. Contudo procuramos adaptar da melhor maneira possível, levando em conta a probabilidade de um bloco estar em cada classe e a distorção média de cada uma delas.

4.4.2 Alterações e Resultados

Com os primeiros testes com a classificação descrita acima, os resultados não chegaram a ser muito compensadores. Em geral, as arestas apareceram com uma melhor representação, mas algumas outras regiões da imagem foram prejudicadas de tal maneira que, como um todo, não houve melhora visível na qualidade das imagens. Isto nos levou a pensar onde estaria a falha do método, uma vez que Ramamurthi e Gersho afirmaram ter obtido uma melhora significativa na qualidade de suas imagens. É bem verdade que eles compararam seus resultados com os da quantização vetorial padrão, isto é, sem separação de média, sendo que eles usaram esta técnica nos blocos que são classificados como sendo de arestas. Entretanto, esperávamos que houvesse uma melhora, mesmo que pequena, na qualidade das imagens. Ramamurthi e Gersho utilizaram diferentes classes para diferentes posições de arestas, mas esta não parece ser a solução no nosso caso, uma vez que nas imagens que obtivemos, podemos observar que o erro não está na posição mas sim na intensidade de brilho em cada um dos lados da aresta. Após uma exaustiva observação das imagens resultantes, detectamos os seguintes problemas que prejudicavam a qualidade das imagens reproduzidas:

- Regiões que são classificadas como aresta complexa ou textura, não são reproduzidas de maneira satisfatória.

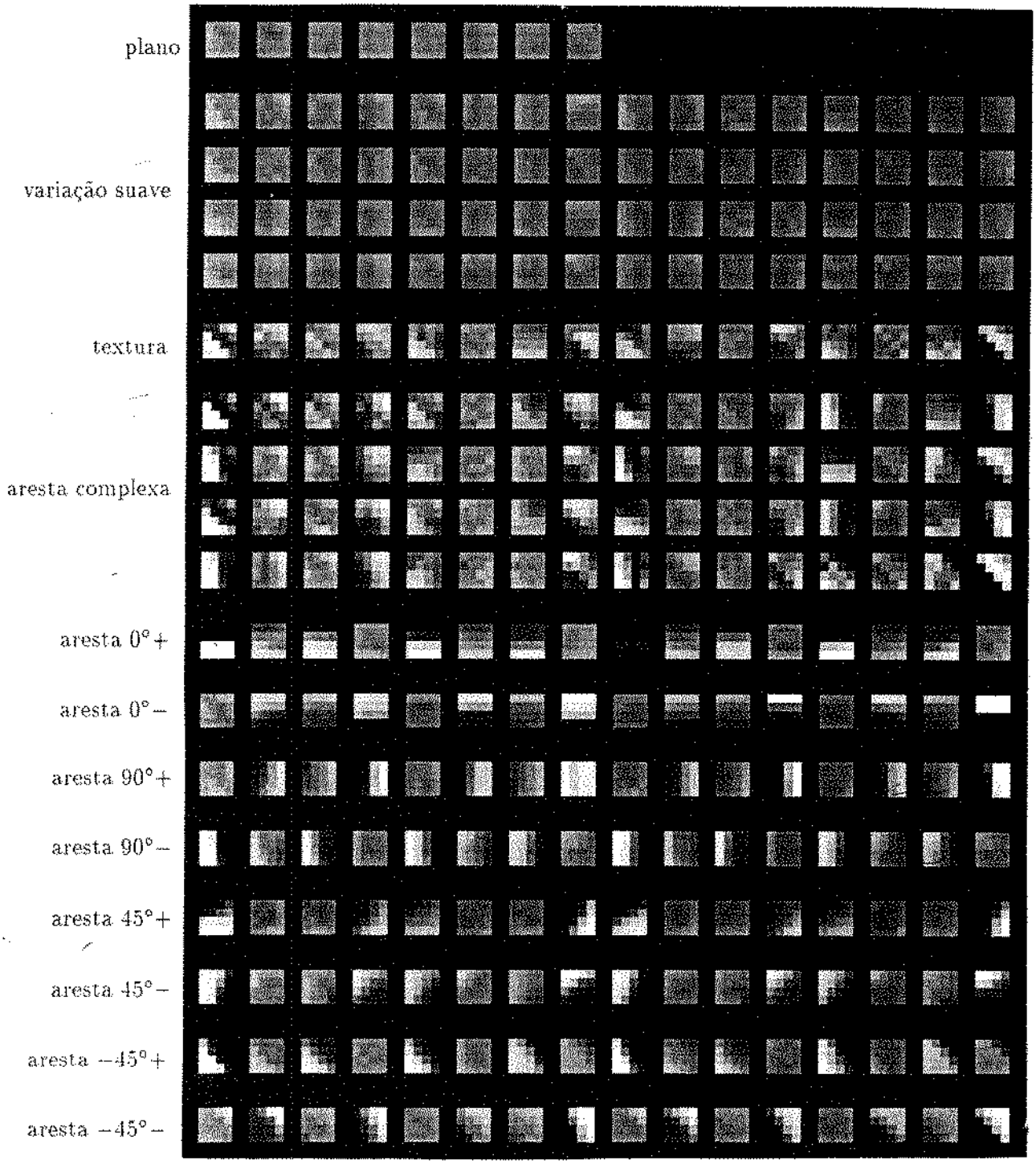


Figura 4.12: Dicionários para as diferentes classes de blocos.

Classe	Polaridade	Porcentagem de blocos da S.T.	Tamanho do dicionário (blocos)
plano		36,96	8
variação suave		25,57	64
textura		2,50	16
aresta complexa		11,26	64
aresta 0°	+	2,07	16
	−	2,03	16
aresta 90°	+	3,28	16
	−	2,98	16
aresta 45°	+	4,46	16
	−	2,24	16
aresta −45°	+	3,33	16
	−	3,32	16
Total		100,00	280

Tabela 4.4: Porcentagem de blocos da seqüência de treinamento em cada classe.

- O "efeito escada", isto é, a visibilidade dos blocos em arestas diagonais, continua bastante evidente em arestas de grande intensidade.
- As fronteiras dos blocos em regiões de variação suave tornaram-se mais visíveis.

Nós diagnosticamos as causas destes problemas e tentamos minimizá-los. O problema dos blocos que são classificados como *textura* ou *aresta complexa* deve-se à grande entropia destas classes. Seria preciso um dicionário excessivamente grande para poder representar satisfatoriamente estes blocos. Como esta solução não é nem um pouco atrativa, preferimos tentar diminuir o número de blocos classificados nestas categorias. Isto foi feito com a ajuda de um filtro de média[1] usando vizinhanças de 2×2 pixels. Cada bloco detectado como sendo *textura* ou *aresta complexa* é filtrado e classificado novamente. O processo de filtragem suaviza os blocos sem chegar a desconfigurá-los. Com este procedimento houve uma queda substancial no número de blocos destas classes, o que permitiu que diminuíssemos os tamanhos dos respectivos dicionários: a maioria dos blocos passou a ser classificado como aresta simples ou variação suave e puderam ser reproduzidos com maior fidelidade.

O efeito escada que citamos acima ocorre porque os dicionários para arestas simples não são capazes de reproduzir bem as arestas onde há grande variação de brilho. Para resolver este problema introduzimos mais uma categoria de bloco: aresta de alta intensidade. Os dicionários usados para este tipo de bloco são os mesmos usados para arestas simples, entretanto, se o bloco for considerado aresta de alta intensidade, o vetor de código terá sua aresta acentuada no processo de decodificação, o que diminui significativamente o efeito escada.

A visibilidade das fronteiras dos blocos em regiões de variação suave é algo que já acontecia mesmo na versão sem classificação, quando usávamos dicionários pequenos (256 vetores ou menos),

[1] Tipo de filtro que substitui o valor de brilho de cada pixel pela média de brilho de uma vizinhança.

entretanto, com a classificação o problema tornou-se mais evidente. Para diminuir este efeito resolvemos aumentar o tamanho do dicionário da classe *variação suave*, uma vez que pudemos diminuir os dicionários das classes *textura* e *aresta complexa*. Para manter o mesmo total de 280 vetores de código que mostramos na Tabela 4.4, alteramos os tamanhos dos dicionários das classes *plano*, *variação suave*, *textura* e *aresta complexa* como é mostrado na Tabela 4.5.

Classe	Tamanho do dicionário
plano	4
variação suave	128
textura	4
aresta complexa	16

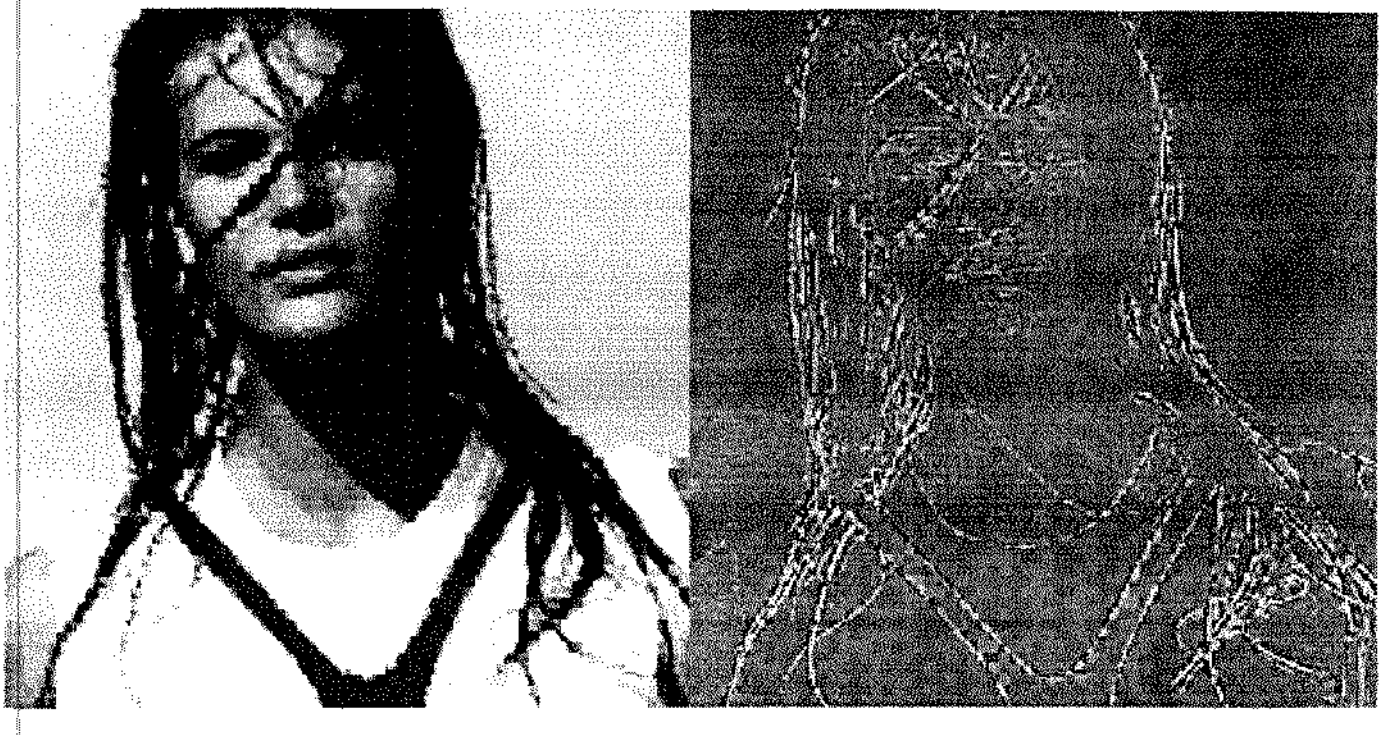
Tabela 4.5: Alteração dos tamanhos dos dicionários.

Uma outra alternativa para diminuir ainda mais esse problema, seria usar um filtro de média nas fronteiras dos blocos em regiões de variação suave, mas não chegamos a implementar esta solução.

A Figura 4.13 mostra um resultado que obtivemos após introduzirmos as alterações que citamos acima. O método utilizado para codificação é o mesmo descrito na Seção 4.2 mas com classificação no processo de quantização vetorial e um total de 280 vetores de código. Quando comparamos o resultado com o da mesma imagem codificada sem classificação, usando um dicionário de 256 vetores (ver Figura 4.5), observa-se que não houve melhora em termos de relação sinal/ruído, entretanto pode-se observar que houve uma melhora na representação das arestas, o efeito escada tornou-se bem menos visível. Ainda é possível encontrar alguns pontos com falhas, mas de uma maneira geral, a qualidade da imagem melhorou.

Ao introduzirmos classificação no MDPT/VQ também foi possível observar alguma melhora na qualidade das imagens, inclusive na relação sinal/ruído. A Figura 4.14 mostra os quadros 31 e 63 da seqüência *hand3* codificada pelo *Block Four* com classificação. Houve uma pequena queda na taxa de compressão, como mostra a Tabela 4.6, mas vale lembrar que usamos 4 bits para identificar a classe de cada bloco; usando um método mais eficiente, como código de Huffman, podemos codificar o identificador da classe com uma média em torno de 2,7 bits, o que provavelmente eliminaria a diferença.

Nós chegamos também a tentar usar quantização vetorial classificada em blocos de 8×8, visando adaptar ao *Block Eight*, mas os resultados não foram nem um pouco animadores. A classificação só produz bons resultados para blocos pequenos (menores que 6×6 [RG86]), quando aumentamos o tamanho do bloco, o número de variações possíveis cresce exponencialmente. Neste caso seria preciso usar um número muito grande de classes, o que poderia dar origem a um algoritmo de classificação extremamente complexo. Isto não é nem um pouco convidativo, levando-se em conta que temos que manter a possibilidade de codificação e decodificação em tempo real.

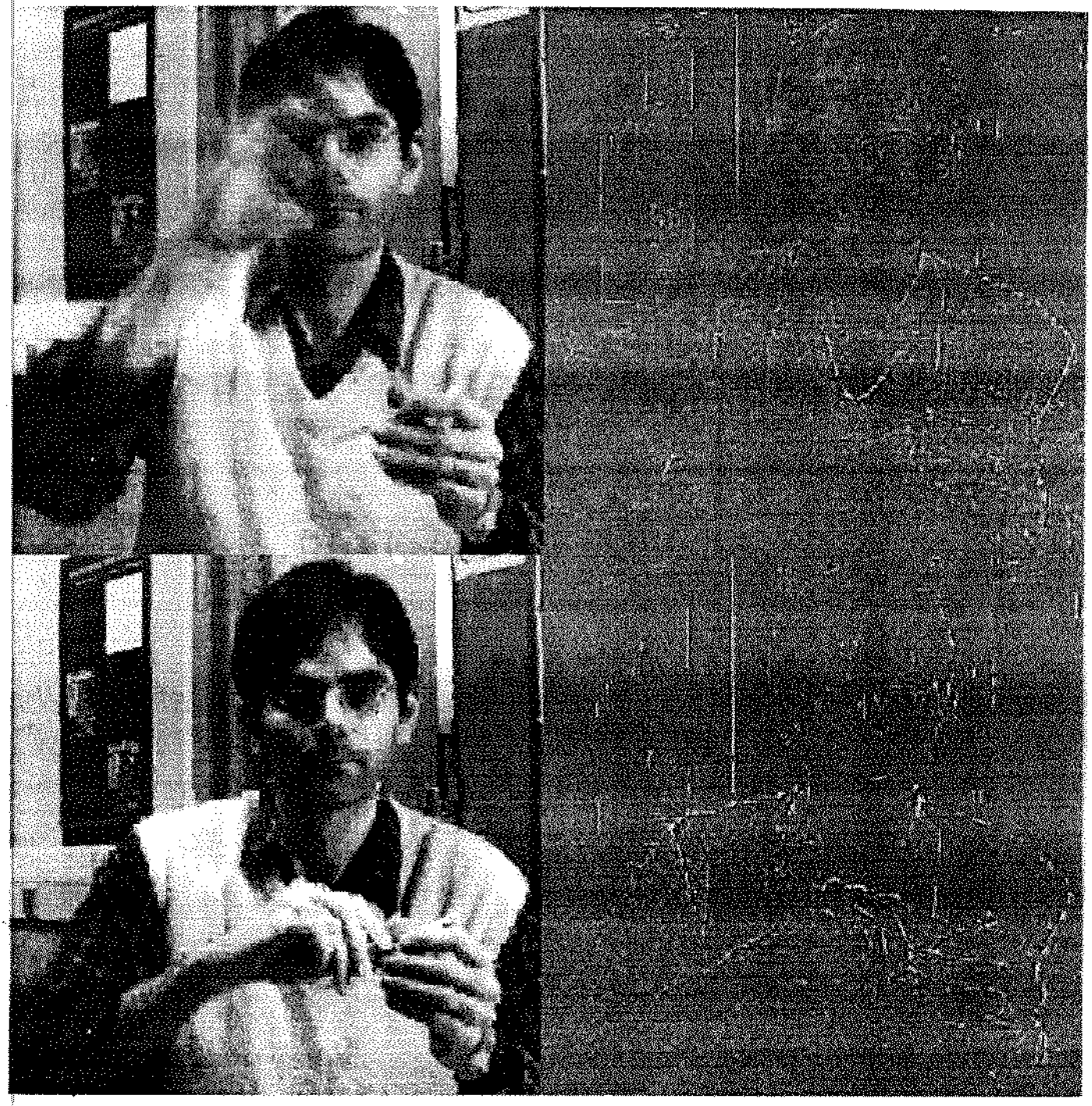


SNR: 21.83 dB.

Figura 4.13: Imagem codificada por quantização vetorial classificada.

Seqüência	Taxa Média		Taxa Mínima	
	Compressão	SNR (dB)	Compressão	SNR (dB)
hand3	12,13:1	30.92	9,71:1	30.28
talk2	22,10:1	32.00	15,21:1	31.52

Tabela 4.6: Resultados obtidos com o *Block Four* com classificação.



SNR: quadro 31 - 31,73 dB; quadro 63 - 31,13 dB.

Figura 4.14: Seqüência *hand3* codificada pelo *Block Four* com classificação.

4.5 Conclusões

Neste capítulo, analisamos o desempenho de diferentes métodos baseados em quantização vetorial. Inicialmente, testamos um método de codificação intraquadros que combina quantização vetorial com DPCM, e cujo objetivo principal foi verificar a qualidade das imagens reproduzidas com auxílio dos dicionários e quantizadores escalares que projetamos, visando adaptá-los ao MDPT/VQ. Conseguimos melhorar significativamente a qualidade das imagens reproduzidas pelo *Block Eight* com apenas uma pequena queda na taxa de compressão.

Como uma tentativa de melhorar a qualidade das imagens reproduzidas sem uma conseqüente queda na taxa de compressão, desenvolvemos um método de quantização vetorial classificada, que chegou a produzir resultados positivos, mas que em alguns casos ainda deixou um pouco a desejar. A melhora produzida pela classificação é bem mais evidente em imagens que possuem muitas arestas bem definidas e de alta intensidade. Em imagens muito ruidosas, ou constituídas, em sua maioria, por variações suaves, a diferença não chega a ser tão significativa. Talvez, resultados melhores possam ser obtidos com um algoritmo de classificação mais sofisticado; entretanto, como necessitamos de processamento em tempo real, não podemos usar um método com custo computacional muito elevado. Vale lembrar que não esgotamos todas as possibilidades desta técnica; uma alternativa seria combiná-la com outros métodos; com o uso de transformadas, por exemplo, blocos dos tipos plano ou variação suave podem ser codificados de maneira bastante eficiente.

Métodos que usam transformadas são o tema do Capítulo 6, onde descrevemos um novo esquema que usa estratégias bastante semelhantes às do MDPT/VQ, mas substituindo a quantização vetorial por DCT, que é uma técnica que se adapta melhor à estratégia de transmissão progressiva.

Capítulo 5

Codificação por Transformadas

5.1 Introdução

Existe uma classe de transformadas discretas bidimensionais que possuem propriedades de grande utilidade para diversas aplicações em processamento de imagens. Estas transformadas são úteis, por exemplo, na análise de imagens, pois permitem extrair determinadas características como amplitude e orientação de arestas. A confecção de filtros passa-altas ou passa-baixas é outro exemplo típico do uso de transformadas, pois elas possuem a propriedade de separar os componentes de alta e baixa freqüência espacial. Neste Capítulo, tratamos de uma outra aplicação bastante comum desta ferramenta matemática: a codificação por transformadas. Como a maioria das imagens do mundo real são constituídas em sua maior parte por componentes de baixa freqüência espacial, a energia tende a ficar concentrada em uma pequena quantidade de coeficientes. Mais adiante, veremos como esta propriedade pode ser usada para obter compressão a altas taxas mantendo uma qualidade aceitável nas imagens reproduzidas. Nas seções seguintes, fazemos uma breve revisão teórica sobre transformadas discretas unitárias e damos os conceitos e as principais propriedades daquelas que despertam maior interesse no estudo de codificação por transformadas.

5.2 Transformadas Unitárias Ortogonais

Uma transformada unitária ortogonal é um tipo de transformação linear inversível que é representada, para uma dada imagem u de tamanho $M \times N$, como um par de operações da forma,

$$\begin{aligned} U(k,l) &= \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} u(m,n)A_{k,l}(m,n) \\ u(m,n) &= \sum_{k=0}^{M-1}\sum_{l=0}^{N-1} U(k,l)B_{k,l}(m,n) \end{aligned} \tag{5.1}$$

onde $U(k,l)$ são chamados coeficientes da imagem transformada e $A_{k,l}$ e $B_{k,l}$ são chamados de núcleos da transformada e de sua inversa, respectivamente, e obedecem às seguintes propriedades:

Normalidade. Para todo k, l,

$$\|A_{k,l}\|^2 = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} A_{k,l}(m,n)\overline{A}_{k,l}(m,n) = 1$$
$$\|B_{k,l}\|^2 = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} B_{k,l}(m,n)\overline{B}_{k,l}(m,n) = 1 \tag{5.2}$$

As barras que aparecem sobre $A_{k,l}$ e $B_{k,l}$ indicam conjugado complexo.

Ortogonalidade. Se $k \neq k'$ ou $l \neq l'$, então,

$$\langle A_{k,l}, A_{k',l'}\rangle = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} A_{k,l}(m,n)\overline{A}_{k',l'}(m,n) = 0$$
$$\langle B_{k,l}, B_{k',l'}\rangle = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} B_{k,l}(m,n)\overline{B}_{k',l'}(m,n) = 0 \tag{5.3}$$

Completude. Se $m \neq m'$ ou $n \neq n'$, então,

$$\sum_{k=0}^{N-1}\sum_{l=0}^{N-1} A_{k,l}(m,n)\overline{A}_{k,l}(m',n') = 0$$
$$\sum_{k=0}^{N-1}\sum_{l=0}^{N-1} B_{k,l}(m,n)\overline{B}_{k,l}(m',n') = 0 \tag{5.4}$$

Dizemos que a transformada é *separável* se os núcleos podem ser representados da forma,

$$A_{k,l}(m,n) = A_k(m)A_l(n)$$
$$B_{k,l}(m,n) = B_k(m)B_l(n) \tag{5.5}$$

Uma transformada bidimensional separável pode ser computada em dois passos: primeiro uma transformada unidimensional é tomada para cada coluna da imagem, ou seja,

$$U'(k,n) = \sum_m u(m,n)A_k(m); \tag{5.6}$$

em seguida, uma outra transformada unidimensional é tomada para cada linha de $U'(k,n)$,

$$U(k,l) = \sum_n U'(k,n)A_l(n). \tag{5.7}$$

Esta propriedade é útil, por exemplo, na elaboração de algoritmos rápidos para transformadas bidimensionais usando algoritmos de transformadas unidimensionais.

Uma outra propriedade interessante das transformadas unitárias é a *conservação de energia*, isto é, se u é uma imagem e U a respectiva imagem transformada então,

$$\|U\|^2 = \sum_k \sum_l |U(k,l)|^2 = \sum_m \sum_n |u(m,n)|^2 = \|u\|^2. \tag{5.8}$$

Antes de demonstrarmos esta propriedade, vamos introduzir um outro tipo de notação. Transformadas unidimensionais são freqüentemente representadas por meio de matrizes da forma,

$$\mathbf{V} = \mathbf{A}\mathbf{v}, \tag{5.9}$$

onde um vetor de N posições é representado como uma matriz coluna $\mathbf{v}_{N\times 1}$ e $\mathbf{A}_{N\times N}$ é a matriz que representa o núcleo da transformada.

Fazendo-se algumas manipulações, esta notação pode ser estendida também para transformadas bidimensionais. Vejamos, por exemplo, a transformada mostrada nas expressões 5.1. Vamos considerar o vetor ψ de MN posições como sendo a representação vetorial de u colocando-se as colunas uma após a outra, e a matriz $\mathbf{A}_{MN\times MN}$ cujas linhas são as representações vetoriais das matrizes A_{kl}; deste modo as expressões 5.1 ficam equivalentes a,

$$\begin{aligned} \Psi &= \mathbf{A}\psi, \\ \psi &= \mathbf{B}\Psi, \end{aligned} \tag{5.10}$$

e U pode ser obtido de Ψ como,

$$U(k,l) = \Psi(k + lM). \tag{5.11}$$

Este tipo de notação facilita bastante o estudo de transformadas, uma vez que vários resultados bem conhecidos da teoria das matrizes podem ser utilizados. Usando esta notação, pode-se verificar que se a transformada é unitária, então

$$\mathbf{B} = \mathbf{A}^{-1} = \overline{\mathbf{A}}^T \tag{5.12}$$

e deste modo podemos facilmente demonstrar a igualdade 5.8 (o símbolo • será usado a partir daqui para indicar fim de demonstração).

$$\|U\|^2 = \|\Psi\|^2 = \sum_k^{MN-1} |\Psi(k)|^2 = \overline{\Psi}^T\Psi = \overline{\psi}^T\overline{\mathbf{A}}^T\mathbf{A}\psi = \overline{\psi}^T\psi = \sum_m^{MN-1} |\psi(m)|^2 = \|\psi\|^2 = \|u\|^2 \ \bullet$$

Existem várias transformadas que podem ser utilizadas em diferentes aplicações de processamento de imagens. Elas podem ser divididas em senoidais e não-senoidais. As transformadas de *Fourier*, do *cosseno* e do *seno* são exemplos de transformadas senoidais; as transformadas de *Hadamard*, *Haar* e *Slant* são exemplos de não-senoidais [Pra78, Jai89]. Além destas, existe também a transformada de *Karhunen-Loeve* (KLT) que é ótima no sentido de agrupar o máximo de energia em um mínimo de coeficientes, e que é, portanto, a que produz melhores resultados quanto a compressão de imagens; mas não é utilizada em geral, uma vez que é difícil de implementar e não possui algoritmo rápido. Além do mais, resultados bastante semelhantes podem ser obtidos com as transformadas senoidais, especialmente com a do cosseno. Com as não-senoidais os resultados são um pouco inferiores; entretanto, elas possuem a vantagem de serem muito simples de implementar e com algoritmos bastante rápidos. Nas seções seguintes damos uma breve descrição das transformadas de Fourier, do cosseno e Karhunen-Loeve. Por simplicidade, a partir daqui, e até o final deste Capítulo, consideramos todas as imagens como sendo quadradas de lado N.

5.2.1 A Transformada de Fourier

A transformada de Fourier discreta, ou simplesmente DFT (*Discrete Fourier Transform*), de uma imagem u é definida como,

$$U(k,l) = \frac{1}{N}\sum_{m=0}^{N-1}\sum_{n=0}^{N-1} u(m,n)W_N^{km+ln}. \tag{5.13}$$

e a sua inversa,

$$u(m,n) = \frac{1}{N}\sum_{k=0}^{N-1}\sum_{l=0}^{N-1} U(k,l)W_N^{-km-ln}. \tag{5.14}$$

onde W_N é uma unidade complexa que é definida por,

$$W_N = \exp\left(-\frac{2\pi i}{N}\right) = \cos\left(\frac{2\pi}{N}\right) - i\,\mathrm{sen}\left(\frac{2\pi}{N}\right) \tag{5.15}$$

onde $i = \sqrt{-1}$.

As definições que apresentamos nas expressões 5.13 e 5.14 não são universais; alguns autores preferem incluir $1/N^2$ na definição da inversa e não colocar nenhuma constante na definição da transformada principal: ou vice-versa (por simplicidade, em algumas ocasiões usaremos estas definições). As expressões que apresentamos acima representam a DFT unitária.

Os índices (k,l) são chamados de freqüências espaciais, ou seja, a transformada de Fourier dá uma representação da imagem no domínio da freqüência, enquanto que a imagem original é uma representação no domínio do espaço. Esta e outras interessantes características da DFT fizeram com que ela se tornasse uma das mais importantes transformadas no processamento de imagens e sinais digitais. A seguir, damos algumas de suas principais propriedades.

1. Separabilidade. A DFT é separável, isto é,

$$\begin{aligned} W_N^{km+ln} &= W_N^{km} W_N^{ln} \\ W_N^{-km-ln} &= W_N^{-km} W_N^{-ln} \end{aligned} \tag{5.16}$$

2. Extensões periódicas. As extensões da transformada e de sua inversa são periódicas de período N, ou seja, se $U = \mathrm{DFT}\{u\}$, então temos que, para todo k, l,

$$U(k,l) = U(k+aN, l+bN), \tag{5.17}$$

onde a e b são quaisquer números inteiros.

Demonstração: Basta verificar que,

$$W_N^{(k+aN)m+(l+bN)n} = W_N^{km+ln+(am+bn)N} = W_N^{km+ln} W_N^{(am+bn)N} = W_N^{km+ln} \quad \bullet$$

3. Algoritmo rápido (FFT). Existem algoritmos que são capazes de calcular a DFT e sua inversa para imagens de tamanho $N \times N$ com $O(N^2 \log N)$ operações (na seção 5.5.1 mostramos exemplos destes algoritmos).

4. Simetria conjugada. Se u é uma imagem real, se $u'(m,n) = u(N-m, N-n)$, e temos que $U = \mathrm{DFT}\{u\}$ e $U' = \mathrm{DFT}\{u'\}$, então,

$$U'(k,l) = \overline{U}(k,l) = U(N-k, N-l). \tag{5.18}$$

Demonstração:

$$U'(k,l) = \sum_m \sum_n u'(m,n) W_N^{mk+nl} = \sum_m \sum_n u(N-m, N-n) W_N^{(m+N-N)k+(n+N-N)l} =$$
$$\sum_m \sum_n u(N-m, N-n) W_N^{-(N-m)k-(N-m)l} W_N^{N(-l-k)}$$

e como $W_N^N = 1$ e $W_N^{-(N-m)k-(N-m)l}$ é o conjugado complexo de $W_N^{(N-m)k+(N-m)l}$, temos que,

$$U'(k,l) = \overline{U}(k,l);$$

de maneira análoga,

$$U(N-k, N-l) = \sum_m \sum_n u(m,n) W_N^{m(N-k)+n(N-l)} = \sum_m \sum_n u(m,n) W_N^{-mk-nl} W_N^{N(m+n)} = \overline{U}(k,l) \bullet$$

5. Deslocamento espacial. Consideremos uma imagem u juntamente com suas extensões periódicas, e consideremos a imagem u' dada por um deslocamento espacial de u, ou seja,

$$u'(m,n) = u(m+p, n+q). \tag{5.19}$$

onde p e q são quaisquer números inteiros. Se $U' = \mathrm{DFT}\{u'\}$ e $U = \mathrm{DFT}\{u\}$, então,

$$U'(k,l) = W_N^{-pk-ql} U(k,l). \tag{5.20}$$

Demonstração:

$$U'(k,l) = \sum_m \sum_n u'(m,n) W_N^{-mk+nl} = \sum_m \sum_n u(m+p, n+q) W_N^{(m+p-p)k+(n+q-q)l}$$
$$= \sum_m \sum_n u(m+p, n+q) W_N^{(m+p)k+(n+q)l} W_N^{-pk-ql} = W_N^{-pk-ql} U(k,l) \bullet$$

6. Deslocamento de freqüência. Se $U = \mathrm{DFT}\{u\}$ e U' é um deslocamento de freqüência de U dado por,

$$U'(k,l) = U(k+p, l+q) \tag{5.21}$$

e se $u' = \mathrm{DFT}^{-1}\{U'\}$ (onde DFT^{-1} indica tranformada inversa) então,

$$u'(m,n) = W_N^{pk+ql} u(m,n). \tag{5.22}$$

A demonstração é totalmente análoga à do item anterior.

Observação. A propriedade de deslocamento espacial que demonstramos acima é válida somente para p e q inteiros. Qual seria então o sentido da expressão

$$U'(k,l) = W_N^{-rk-sl}U(k,l) = \frac{1}{N}\sum_m\sum_n u(m,n)W_N^{k(m-r)+l(n-s)} \tag{5.23}$$

com r e s não inteiros? Não podemos dizer que $U'(k,l)$ é a DFT de uma imagem u com deslocamento espacial, pois $u(m+r, n+s)$ é algo que não faz sentido. Entretanto, expressões como a 5.23 podem ser úteis em alguns casos e resolvemos considerá-la como sendo uma especie de generalização da DFT que chamaremos de *DFT em torno do ponto* (r,s), e cuja inversa é obtida observando-se que

$$\begin{aligned} u(m,n) = \frac{1}{N}\sum_k\sum_l U(k,l)W_N^{-km-ln} &= \frac{1}{N}\sum_k\sum_l W_N^{rk+sl}U'(k,l)W_N^{-km-ln} \\ &= \frac{1}{N}\sum_k\sum_l U'(k,l)W_N^{-k(m-r)-l(n-s)} \end{aligned} \tag{5.24}$$

Na seção a seguir, mostramos uma aplicação desta generalização da DFT.

5.2.2 A Transformada do Cosseno

Sabe-se que as representações por séries de Fourier de funções reais pares[1] contêm apenas coeficientes reais que correspondem a uma série de cossenos [Fig87]. O que vamos verificar a seguir é que no caso de DFT's de imagens simétricas ocorre algo semelhante. Uma imagem simétrica v de tamanho $2N \times 2N$ pode ser obtida a partir de uma imagem u de tamanho $N \times N$ por meio da seguinte definição:

$$v(m,n) = \begin{cases} u(m,n) & \text{se} \quad m \geq 0 \ \text{ e } \ n \geq 0 \\ u(-1-m,n) & \text{se} \quad m < 0 \ \text{ e } \ n \geq 0 \\ u(m,-1-n) & \text{se} \quad m \geq 0 \ \text{ e } \ n < 0 \\ u(-1-m,-1-n) & \text{se} \quad m < 0 \ \text{ e } \ n < 0 \end{cases} \tag{5.25}$$

Esta imagem é simétrica em relação à posição $(-1/2, -1/2)$, então vamos obter sua DFT em torno deste ponto, que pela expressão 5.23 é dada por:

$$V'(k,l) = W_{2N}^{k/2+l/2}V(k,l) = \frac{1}{2N}\sum_{m=-N}^{N-1}\sum_{n=-N}^{N-1} v(m,n)W_{2N}^{k(m+\frac{1}{2})+l(n+\frac{1}{2})} \tag{5.26}$$

onde $V = \text{DFT}\{v\}$. Esta quantidade pode também ser representada da seguinte maneira:

$$\begin{aligned} V'(k,l) = \frac{1}{2N}\Bigg(&\sum_{m=0}^{N-1}\sum_{n=0}^{N-1} u(m,n) + \sum_{m=-N}^{-1}\sum_{n=0}^{N-1} u(-1-m,n) + \sum_{m=0}^{N-1}\sum_{n=-N}^{-1} u(m,-1-n) \\ &+ \sum_{m=-N}^{-1}\sum_{n=-N}^{-1} u(-1-m,-1-n)\Bigg) W_{2N}^{k(m+\frac{1}{2})+l(n+\frac{1}{2})} \end{aligned} \tag{5.27}$$

[1] Uma função real é par se $f(x) = f(-x)$, para todo x.

onde podemos verificar que,

$$\sum_{m=-N}^{-1}\sum_{n=0}^{N-1} u(-1-m,n)W_{2N}^{k(m+\frac{1}{2})+l(n+\frac{1}{2})} = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} u(m,n)W_{2N}^{-k(m+\frac{1}{2})+l(n+\frac{1}{2})}$$

$$\sum_{m=0}^{N-1}\sum_{n=-N}^{-1} u(m,-1-n)W_{2N}^{k(m+\frac{1}{2})+l(n+\frac{1}{2})} = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} u(m,n)W_{2N}^{k(m+\frac{1}{2})-l(n+\frac{1}{2})}$$

$$\sum_{m=-N}^{-1}\sum_{n=-N}^{-1} u(-1-m,-1-n)W_{2N}^{k(m+\frac{1}{2})+l(n+\frac{1}{2})} = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} u(m,n)W_{2N}^{-k(m+\frac{1}{2})-l(n+\frac{1}{2})}$$

ou seja, como $u(m,n)$ é real para todo (m,n), podemos observar que a expressão 5.27 é constituída de somas de fatores com seus conjugados complexos, de modo que os termos com seno, que correspondem às partes imaginárias, se anulam mutuamente, restando apenas os termos com coseno, logo, temos que,

$$V'(k,l) = \frac{2}{N}\sum_{m=0}^{N-1}\sum_{n=0}^{N-1} u(m,n)\cos\left(\frac{\pi}{N}k(m+\frac{1}{2})\right)\cos\left(\frac{\pi}{N}l(n+\frac{1}{2})\right) \tag{5.28}$$

Esta expressão representa a *Transformada do Cosseno Discreta*, ou simplesmente DCT (*Discrete Cosine Transform*); normalizando seus termos, obtemos a DCT unitária, que é dada por,

$$U(k,l) = \frac{2}{N}c(k)c(l)\sum_{m=0}^{N-1}\sum_{n=0}^{N-1} u(m,n)\cos\left(\frac{\pi}{N}k(m+\frac{1}{2})\right)\cos\left(\frac{\pi}{N}l(n+\frac{1}{2})\right) \tag{5.29}$$

onde $c(0) = \frac{1}{\sqrt{2}}$ e $c(j) = 1$ para $j \neq 0$.

Para obter a inversa, basta observar, levando em conta as expressões 5.24 e 5.25, que

$$u(m,n) = v(m,n) = \frac{1}{2N}\sum_{k=-N}^{N-1}\sum_{l=-N}^{N-1} V'(k,l)W_{2N}^{-k(m+\frac{1}{2})-l(n+\frac{1}{2})} \tag{5.30}$$

Pela expressão 5.28 pode-se perceber que V' também é simétrica, isto é,

$$V'(k,l) = V'(-k,l) = V'(k,-l) = V'(-k,-l) \tag{5.31}$$

logo, na expressão 5.30 existe uma soma de conjugados complexos semelhante à da expressão 5.26, e usando o fato de que para todo k,l,

$$V'(k,N) = V'(N,l) = 0 \tag{5.32}$$

podemos concluir que

$$u(m,n) = \frac{2}{N}\sum_{k=0}^{N-1}\sum_{l=0}^{N-1} C(k)C(l)V'(k,l)\cos\left(\frac{\pi}{N}k(m+\frac{1}{2})\right)\cos\left(\frac{\pi}{N}l(n+\frac{1}{2})\right) \tag{5.33}$$

onde $C(0) = \frac{1}{2}$ e $C(j) = 1$ para $j \neq 0$. E no caso da DCT unitária,

$$u(m,n) = \frac{2}{N}\sum_{k=0}^{N-1}\sum_{l=0}^{N-1} c(k)c(l)U(k,l)\cos\left(\frac{\pi}{N}k(m+\frac{1}{2})\right)\cos\left(\frac{\pi}{N}l(n+\frac{1}{2})\right) \tag{5.34}$$

O coeficiente $U(0,0)$, normalmente chamado de nível DC, possui um valor proporcional à média aritmética dos valores de $u(m,n)$. Os demais coeficientes, chamados de níveis AC, representam os componentes de freqüência espacial, como na DFT. Quando lidamos com imagens do mundo real a maior parte da energia tende a ficar concentrada nas proximidades do nível DC, onde estão os componentes de baixa freqüência espacial, e é justamente este fato que é explorado para se obter a compressão, como veremos mais adiante.

A DCT é a transformada mais usada para codificação de imagens, pois, além de ter a vantagem de ser real, entre as que possui algoritmo rápido, é a que produz resultados mais próximos dos obtidos com a KLT, que é ótima, e da qual damos uma rápida descrição a seguir.

5.2.3 A Transformada de Karhunen-Loeve

A transformada de Karhunen-Loeve (KLT) e sua inversa são definidas para uma imagem u como,

$$U(k,l) = \sum_m \sum_n u(m,n)\Phi_{k,l}(m,n) \tag{5.35}$$

$$u(m,n) = \sum_k \sum_l U(k,l)\Phi_{k,l}(m,n) \tag{5.36}$$

onde os núcleos $\Phi_{k,l}$ satisfazem à condição,

$$\lambda(k,l)\Phi_{k,l}(m,n) = \sum_{m'} \sum_{n'} R(m,n;m',n')\Phi_{k,l}(m',n') \tag{5.37}$$

sendo que $\lambda(k,l)$ são constantes para cada (k,l) e $R(m,n;m',n')$ é a função de covariância da imagem, que é definida por,

$$R(m,n;m',n') = E\left[(u(m,n) - E[u(m,n)])(u(m',n') - E[u(m',n')])\right] \tag{5.38}$$

onde $E[\cdot]$ indica expectância matemática. Considerando a representação vetorial das matrizes $\Phi_{k,l}$, observamos que elas são na verdade os autovetores da função de covariância e que $\lambda(k,l)$ são os respectivos autovalores.

Usando representação por matrizes também podemos representar a KLT como,

$$\Psi = \Phi u, \tag{5.39}$$

e sua inversa como,

$$u = \Phi^T \Psi. \tag{5.40}$$

Apesar de ser ótima, dificilmente a KLT é usada na prática. A maior dificuldade para utilização desta transformada está na determinação de seus núcleos, pois eles dependem de características estatísticas das imagens que normalmente não podem ser explicitadas de maneira analítica. Outra desvantagem da KLT é que seu custo computacional tende a ser muito elevado, uma vez que ela não possui algoritmo rápido. Por estas razões, a KLT é geralmente substituída pela DCT, ou outra transformada, em esquemas de codificação de imagens.

5.3 Esquema Básico de Codificação por Transformadas

A codificação por transformadas é um dos métodos que produz melhores resultados em compressão de imagens; ela é capaz de atingir altas taxas de compressão (8:1 ou mais), mantendo uma qualidade aceitável na imagem reproduzida. Isto é conseguido graças à propriedade que a DCT e outras transformadas possuem de concentrar a maior parte da energia da imagem em uma pequena quantidade de coeficientes (no caso de imagens de mundo real). A compressão é atingida fazendo-se uma alocação seletiva de bits, isto é, mais bits onde há maior concentração de energia e menos (ou nenhum) onde há menor concentração.

Por razões práticas, normalmente a transformada não é computada sobre a imagem inteira; isto tornaria a codificação muito lenta mesmo utilizando algoritmos rápidos, e seria impraticável para aplicações que necessitam de tempo real. Ao invés disso, a imagem é dividida em blocos retangulares de tamanho fixo e cada um deles é processado separadamente. Usando um algoritmo rápido, a transformada de uma imagem de $N \times N$ pixels seria computada com $O(N^2 \log N)$ operações; dividindo-se esta imagem em blocos de tamanho $n \times n$, teríamos um total de N^2/n^2 blocos, cada um dos quais transformado com $O(n^2 \log n)$ operações, o que resulta em um total de $O(N^2 \log n)$ operações. Quanto menor for o bloco, maior será a velocidade do método; entretanto, com blocos maiores é possível atingir melhores resultados no que diz respeito a qualidade e taxa de compressão.

Uma vez determinado o tamanho do bloco, que normalmente está entre 8×8 e 32×32, o passo seguinte é a alocação de bits, que pode ser baseada na variância (ou desvio padrão) dos coeficientes da imagem transformada. Ou seja, nos coeficientes onde a variância é maior, um número maior de bits é alocado, ao passo que, nos coeficientes onde há menor variância, são alocados poucos bits ou nenhum. No Capítulo 6 descrevemos uma técnica pela qual esta alocação pode ser feita. A Figura 5.1 mostra uma típica alocação de bits para DCT de um bloco de 16×16 mantendo uma média de 1 bit/pixel.

Além da alocação de bits, um outro ponto essencial para um bom desempenho do método é a quantização dos valores dos coeficientes. Em um caso como o ilustrado na Figura 5.1, seriam necessários 8 quantizadores diferentes; um para cada número de bits indicado. Bons resultados podem ser obtidos utilizando-se quantizadores de Lloyd-Max (ver Apêndice B), que minimizam o erro médio quadrático.

O método que descrevemos acima é o mais simples no que diz respeito à codificação por transformadas e existem inúmeras variações que tentam melhorar o seu desempenho. Na seção a seguir citamos algumas destas variações.

5.4 Variações de Codificação por Transformadas

A codificação por transformadas divide-se basicamente em três elementos: a transformada, a alocação de bits e a quantização. Qualquer um destes elementos pode ser alterado para que se possa atingir resultados melhores; além disso, também podem ser utilizadas estratégias como predição e adaptatividade.

Um exemplo de alteração na transformada seria, por exemplo, o uso de transformadas adaptativas, isto é, os núcleos se modificam cada vez que ocorrem mudanças em alguns parâmetros estatísticos da imagem. O problema com esta solução é que ela seria bastante cara computacionalmente, o que faz com que ela não seja recomendável para aplicações em tempo real.

8	7	6	5	3	3	2	2	2	1	1	1	1	1	0	0
7	6	5	4	3	3	2	2	1	1	1	1	1	0	0	0
6	5	4	3	3	2	2	2	1	1	1	1	1	0	0	0
5	4	3	3	3	2	2	2	1	1	1	1	1	0	0	0
3	3	3	3	2	2	2	1	1	1	1	1	0	0	0	0
3	3	2	2	2	2	2	1	1	1	1	1	0	0	0	0
2	2	2	2	2	2	1	1	1	1	1	0	0	0	0	0
2	2	2	2	1	1	1	1	1	1	1	0	0	0	0	0
2	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 5.1: Exemplo de alocação de bits para DCT.

Uma outra modificação que poderia ser feita é quanto à maneira como são escolhidos os coeficientes a serem codificados. No método que descrevemos na seção anterior, o mapeamento é fixo para todos os blocos, ou seja, se para um determinado coeficiente foi alocado um ou mais bits, ele é codificado, caso contrário ele não é codificado e é substituído por zero na decodificação. Esta técnica pode ser chamada de *codificação por região*. Alternativamente, uma outra estratégia pode ser adotada: são codificados apenas os coeficientes cujo valor é superior a um determinado limiar convenientemente escolhido; os demais são sustituídos por zeros. Esta segunda técnica é conhecida como *codificação por limiar* e permite que se atinja resultados melhores, uma vez que os coeficientes com valores próximos a zero não precisam ser codificados. Entretanto, como não é predeterminado quais coeficientes devem ser codificados, um mapeamento deve ser transmitido (ou armazenado) para cada bloco da imagem. Técnicas como *run-length* podem ser utilizadas para codificar éste mapeamento, de modo a diminuir o custo adicional de memória. As taxas de compressão atingidas com este método são variáveis e depedem da imagem; em geral são superiores às atingidas com o método básico, entretanto, ele possui a desvantagem de gerar um código mais complexo e exigir maior custo computacional.

Pode-se ainda utilizar esquemas adaptativos com relação à alocação de bits e à quantização. Uma possibilidade seria dividir os blocos em diferentes categorias e fazer uma alocação de bits diferente para cada uma delas; outra alternativa seria alocar os bits da maneira diferente para cada bloco, de modo a manter a distorção constante ao longo de toda a imagem. Também podem ser utilizados quantizadores adaptativos, que mudam de acordo com variações estatísticas ao longo da imagem.

Existe também um método denominado codificação híbrida, que combina DPCM com codi-

ficação por transformadas. A idéia básica é utilizar uma transformada unidimensional nas colunas da imagem e então codificar as linhas da imagem resultante por DPCM. Adaptatividade também pode ser usada com este método.

No caso de estarmos lidando com seqüências de vídeo, uma outra possibilidade é o uso de transformadas tridimensionais, mas este método tende a ter um custo computacional bastante elevado, tanto no que diz respeito ao número de operações, quanto à memoria, pois seria preciso armazenar um número de quadros suficiente para computar a transformada e sua inversa.

As taxas de compressão atingidas com as diferentes variações podem ser bastante elevadas, a tabela 5.1 mostra uma comparação entre diferentes esquemas de codificação por transformadas mantendo relação sinal/ruído entre 30 e 36 dB [Jai89].

Método	Taxa de compressão
Unidimensional	2-4:1
Bidimensional	4-8:1
Bidimensional adaptativa	8-16:1
Tridimensional	8-16:1
Tridimensional adaptativa	16-32:1

Tabela 5.1: Comparação entre diferentes métodos de codificação por transformadas.

5.5 Transformadas Rápidas

Um incoveniente que pode surgir quando se pensa inicialmente em usar transformadas é o grande número de operações necessário para computá-las. Para se calcular a DFT, por exemplo, de uma imagem de tamanho $N \times N$ seriam necessárias N^4 multiplicações e $N^4 - N^2$ somas de números complexos. No caso de uma imagem de 512×512 pixels, teríamos um total de 2^{36} ($\approx 6,87 \times 10^{10}$) multiplicações complexas, o que é completamente impraticável para aplicações com necessidade de tempo real. Entretanto, a maioria das transformadas usadas em processamento de imagens possuem algoritmos rápidos que permitem baixar substancialmente o número de operações. A seguir veremos alguns destes algoritmos.

5.5.1 FFT

Nesta Seção, vamos descrever algumas variações do algoritmo FFT (*Fast Fourier Transform*) com o qual é possível computar a DFT de uma imagem de tamanho $N \times N$ com $O(N^2 \log N)$ operações. Inicialmente vamos considerar sua versão unidimensional, e em seguida mostraremos duas maneiras pelas quais podemos obter FFT's bidimensionais.

FFT Unidimensional

Seja v um vetor de tamanho N (por simplicidade estamos supondo que N é uma potência de 2), sua DFT (não unitária) é definida como,

$$V(k) = \sum_{n=0}^{N-1} v(n) W_N^{nk} \tag{5.41}$$

Consideremos os vetores v_0 e v_1 de tamanhos $N/2$ definidos como:

$$v_0(n) = v(2n) \tag{5.42}$$

$$v_1(n) = v(2n+1) \tag{5.43}$$

para $n = 0, \ldots, N/2 - 1$. As DFT's de v_0 e v_1 são dadas por:

$$V_0(k) = \sum_{n=0}^{N/2-1} v_0(n) W_{N/2}^{nk} \tag{5.44}$$

$$V_1(k) = \sum_{n=0}^{N/2-1} v_1(n) W_{N/2}^{nk} \tag{5.45}$$

Mas, uma vez que $W_{N/2}^{nk} = W_N^{2nk}$, verificamos que,

$$V_0(k) = \sum_{n=0}^{N/2-1} v(2n) W_N^{2nk} \tag{5.46}$$

corresponde aos fatores pares da expressão 5.41 e que, por outro lado,

$$W_N^k V_1(k) = \sum_{n=0}^{N/2-1} v(2n+1) W_N^{(2n+1)k} \tag{5.47}$$

corresponde aos fatores ímpares, de modo que,

$$V(k) = V_0(k) + W_N^k V_1(k) \tag{5.48}$$

e como $W_N^{N/2} = W_2 = -1$, e sabendo-se que V_1 e V_2 são periódicos de período $N/2$,

$$V(k + N/2) = V_0(k) - W_N^k V_1(k) \tag{5.49}$$

para $k = 0, \ldots, N/2 - 1$.

Aplicando-se as expressões acima recursivamente, temos um típico algoritmo de divisão e conquista cuja fórmula de recorrência é dada por

$$T(N) = 2T(N/2) + O(N) \tag{5.50}$$

de onde podemos concluir que a complexidade de tempo do algoritmo é $O(N \log N)$ [Man89]. Mais precisamente, podemos verificar que a DFT de um vetor de tamanho N pode ser calculada com $N/2 \log_2 N$ multiplicações e $N \log_2 N$ somas de números complexos.

O cálculo da transformada inversa é totalmente análogo, ou seja, se $V = \overline{\text{DFT}}\{v\}$ então definimos os vetores V_0 e V_1 como sendo.

$$V_0(k) = V(2k) \tag{5.51}$$
$$V_1(k) = V(2k+1) \tag{5.52}$$

para $k = 0, \ldots, N/2-1$. Deste modo, se $v_0 = \text{DFT}^{-1}\{V_0\}$ e $v_1 = \text{DFT}^{-1}\{V_1\}$, então:

$$v(n) = v_0(n) + W_N^{-n} v_1(n) \tag{5.53}$$
$$v(n+N/2) = v_0(n) - W_N^{-n} v_1(n) \tag{5.54}$$

para $n = 0, \ldots, N/2-1$.

FFT Bidimensional

Os algoritmos acima permitem que se calcule com rapidez a DFT e sua inversa para seqüências unidimensionais. Para calcular a transformada de uma imagem de tamanho $N \times N$ precisamos obter um FFT bidimensional. Uma maneira de conseguir isto seria simplesmente usando o fato de que a DFT é separável e aplicar o FFT unidimensional primeiro nas colunas e depois nas linhas da imagem, como vimos na Seção 5.2. Desta forma seriam computadas $2N$ FFT's resultando em um total de $N^2 \log_2 N$ multiplicações e $2N^2 \log_2 N$ somas de números complexos. Isto já nos daria um ganho substancial quanto à velocidade de processamento: no caso de uma imagem com 512×512 pixels, por exemplo, o número de multiplicações complexas cairia de 6.78×10^{10} para 2.36×10^6, ou seja, um número quase 30.000 vezes menor.

Uma outra opção, que pode produzir um resultado um pouco melhor, seria generalizar para duas dimensões o FFT unidimensional que descrevemos acima. Isto pode ser feito da seguinte maneira: seja u uma imagem de tamanho $N \times N$ e $U = \text{DFT}\{u\}$; consideremos as imagens u_0, u_1, u_2 e u_3 de tamanhos $N/2 \times N/2$ definidas como:

$$u_0(m,n) = u(2m, 2n) \tag{5.55}$$
$$u_1(m,n) = u(2m+1, 2n) \tag{5.56}$$
$$u_2(m,n) = u(2m, 2n+1) \tag{5.57}$$
$$u_3(m,n) = u(2m+1, 2n+1) \tag{5.58}$$

para $m, n = 0, \ldots, N/2-1$. Se $U_j = \text{DFT}\{u_j\}$ $(j = 0,1,2,3)$, podemos verificar que para $k, l = 0, \ldots, N/2-1$,

$$U(k,l) = (U_0(k,l) + W_N^k U_1(k,l)) + (W_N^l U_2(k,l) + W_N^{k+l} U_3(k,l)) \tag{5.59}$$
$$U(k+N/2,l) = (U_0(k,l) - W_N^k U_1(k,l)) + (W_N^l U_2(k,l) - W_N^{k+l} U_3(k,l)) \tag{5.60}$$
$$U(k,l+N/2) = (U_0(k,l) + W_N^k U_1(k,l)) - (W_N^l U_2(k,l) + W_N^{k+l} U_3(k,l)) \tag{5.61}$$
$$U(k+N/2,l+N/2) = (U_0(k,l) - W_N^k U_1(k,l)) - (W_N^l U_2(k,l) - W_N^{k+l} U_3(k,l)) \tag{5.62}$$

Podemos usar algumas variáveis auxiliares para diminuir o número de operações para cada (k,l), ou seja, os valores acima podem ser calculados da seguinte maneira:

$$Z_0 = U_0(k,l)$$

$$
\begin{aligned}
Z_1 &= W_N^k U_1(k,l) \\
Z_2 &= W_N^l U_2(k,l) \\
Z_3 &= W_N^{k+l} U_3(k,l)
\end{aligned}
$$

$$
\begin{aligned}
Z_{01}^+ &= Z_0 + Z_1 \\
Z_{01}^- &= Z_0 - Z_1 \\
Z_{23}^+ &= Z_2 + Z_3 \\
Z_{23}^- &= Z_2 - Z_3
\end{aligned}
$$

$$
\begin{aligned}
U(k,l) &= Z_{01}^+ + Z_{23}^+ \\
U(k+N/2,l) &= Z_{01}^- + Z_{23}^- \\
U(k,l+N/2) &= Z_{01}^+ - Z_{23}^+ \\
U(k+N/2,l+N/2) &= Z_{01}^- - Z_{23}^-
\end{aligned}
$$

Desta maneira, teremos uma média de 3/4 multiplicações e 2 somas para cada (k,l) em cada passo de recursão, o que nos dá um total $\frac{3}{4}N^2 \log_2 N$ multiplicações e $2N^2 \log_2 N$ somas. Este método permite uma queda de 1/4 no número de multiplicações em relação ao método das linhas e colunas, e também possui a vantagem de poder ser implementado com um número significativamente menor de chamadas de função, o que contribui para agilizar ainda mais o processamento.

Existem algoritmos baseados em transformadas polinomiais que são capazes de calcular a DFT bidimensional com um número ainda menor de operações [Nus82]; entretanto, os algoritmos que descrevemos acima têm como vantagem sua simplicidade e facilidade de implementação.

5.5.2 FCT

Na Seção 5.2.2, vimos que a DCT de uma imagem de tamanho $N \times N$ pode ser obtida a partir da DFT de uma imagem de tamanho $2N \times 2N$; logo, um algoritmo rápido para a transformada do cosseno, isto é, um FCT (*Fast Cosine Transform*) pode ser obtido apenas definindo uma imagem simétrica como na expressão 5.25 e aplicando o FFT. Alternativamente, para evitar índices negativos, podemos definir v como,

$$
v(m,n) = \begin{cases}
u(m,n) & \text{se} \quad m < N \ \text{e} \ n < N \\
u(2N-1-m,n) & \text{se} \quad m \geq N \ \text{e} \ n < N \\
u(m,2N-1-n) & \text{se} \quad m < N \ \text{e} \ n \geq N \\
u(2N-1-m,2N-1-n) & \text{se} \quad m \geq N \ \text{e} \ n \geq N
\end{cases} \tag{5.63}
$$

para $m,n = 0,\ldots,2N-1$. Como $W_{2N}^{2N} = 1$, isto não alteraria os resultados que obtivemos na Seção 5.2.2.

O que vamos mostrar a seguir é que, utilizando um método introduzido por Makhoul [Mak80], a DCT de uma imagem de tamanho $N \times N$ pode ser obtida a partir da DFT de uma imagem também de tamanho $N \times N$, de modo que é necessário um número quatro vezes menor de operações. Consideremos as seguintes imagens de tamanho $N \times N$ definidas a partir de v como,

$$
x(m,n) = v(2m,2n) \tag{5.64}
$$

$$y_1(m,n) = v(2m+1, 2n) \tag{5.65}$$
$$y_2(m,n) = v(2m, 2n+1) \tag{5.66}$$
$$y_3(m,n) = v(2m+1, 2n+1) \tag{5.67}$$

para $m, n = 0, \ldots, N-1$. Desta forma, a DFT de v pode ser dada por,

$$V(k,l) = \frac{1}{2N}\left(\sum_{m=0}^{N-1}\sum_{n=0}^{N-1} x(m,n)W_{2N}^{2mk+2nl} + \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} y_1(m,n)W_{2N}^{(2m+1)k+2nl}\right.$$
$$\left. + \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} y_2(m,n)W_{2N}^{2mk+(2n+1)l} + \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} y_3(m,n)W_{2N}^{(2m+1)k+(2n+1)l}\right) \tag{5.68}$$

Olhando mais atentamente para cada um dos fatores da expressão acima, e levando em conta que $W_{2N}^{2mk+2nl} = W_N^{mk+nl}$, observamos que:

$$\frac{1}{N}\sum_{m=0}^{N-1}\sum_{n=0}^{N-1} x(m,n)W_{2N}^{2mk+2nl} = X(k,l) \tag{5.69}$$
$$\frac{1}{N}\sum_{m=0}^{N-1}\sum_{n=0}^{N-1} y_1(m,n)W_{2N}^{(2m+1)k+2nl} = W_{2N}^{k}Y_1(k,l) \tag{5.70}$$
$$\frac{1}{N}\sum_{m=0}^{N-1}\sum_{n=0}^{N-1} y_2(m,n)W_{2N}^{2mk+(2n+1)l} = W_{2N}^{l}Y_2(k,l) \tag{5.71}$$
$$\frac{1}{N}\sum_{m=0}^{N-1}\sum_{n=0}^{N-1} y_3(m,n)W_{2N}^{(2m+1)k+(2n+1)l} = W_{2N}^{k+l}Y_3(k,l) \tag{5.72}$$

onde X, Y_1, Y_2 e Y_3 são as DFT's de x, y_1, y_2 e y_3, respectivamente. Mas também temos que:

$$y_1(m,n) = x(N-1-m, n) \tag{5.73}$$
$$y_2(m,n) = x(m, N-1-n) \tag{5.74}$$
$$y_3(m,n) = x(N-1-m, N-1-n) \tag{5.75}$$

sendo assim, usando as propriedades de simetria conjugada e deslocamento espacial da DFT, podemos verificar que:

$$Y_1(m,n) = W_N^{-k}X(N-k, l) \tag{5.76}$$
$$Y_2(m,n) = W_N^{-l}X(k, N-l) = W_N^{-l}\overline{X}(N-k, l) \tag{5.77}$$
$$Y_3(m,n) = W_N^{-k-l}\overline{X}(k,l) \tag{5.78}$$

Unindo-se estes resultados aos das expressões 5.70 a 5.72 temos:

$$V(k,l) = \frac{1}{2}\left(X(k,l) + W_{2N}^{-k}X(N-k,l) + W_{2N}^{-l}\overline{X}(N-k,l) + W_{2N}^{-k-l}\overline{X}(k,l)\right) \tag{5.79}$$

e daí temos que,

$$V'(k,l) = W_{2N}^{k/2+l/2}V(k,l) = \frac{1}{2}\left(W_{2N}^{-k/2+l/2}X(k,l) + W_{2N}^{-k/2+l/2}X(N-k,l)\right.$$
$$\left. + W_{2N}^{k/2-l/2}\overline{X}(N-k,l) + W_{2N}^{-k/2-l/2}\overline{X}(k,l)\right) \tag{5.80}$$

Como temos uma soma de conjugados complexos na expressão acima, as partes imaginárias se anulam e, levando-se em conta que $W_{2N}^{1/2} = W_{4N}$, $V'(k,l)$ pode ser reescrito como,

$$V'(k,l) = Re\left[W_{4N}^{k+l}X(k,l) + W_{4N}^{-k+l}X(N-k,l)\right] \tag{5.81}$$

Que é a DCT não unitária de u. Para obter a DCT unitária, basta normalizar os termos de V', conforme foi visto na Seção 5.2.2, e para facilitar a obtenção de x, podemos extraí-la diretamente da imagem original u, por meio de,

$$x(m,n) = \begin{cases} u(2m,2n) & \text{se} \quad m \leq N/2 \ \text{e} \ n < N/2 \\ u(2N-2m-1,n) & \text{se} \quad m \geq N/2 \ \text{e} \ n < N/2 \\ u(m,2N-2n-1) & \text{se} \quad m < N/2 \ \text{e} \ n \geq N/2 \\ u(2N-2m-1,2N-2n-1) & \text{se} \quad m \geq N/2 \ \text{e} \ n \geq N/2 \end{cases} \tag{5.82}$$

para $m,n = 0,\ldots,N-1$.

Para conseguir a transformada inversa, o primeiro passo é a obtenção de X a partir de V', usando o fato de que,

$$V'(k,l) = \frac{1}{2}\left(W_{4N}^{k+l}X(k,l) + W_{4N}^{-k+l}X(N-k,l) + W_{4N}^{k-l}X(k,N-l) + W_{4N}^{-k-l}X(N-k,N-l)\right) \tag{5.83}$$

e observando que $W_{4N}^{N} = W_4 = -i$, podemos verificar que

$$X(k,l) = \frac{1}{2}W_{4N}^{-k-l}(V'(k,l) - V'(N-k,N-l) - i[V'(N-k,l) + V'(k,N-l)]) \tag{5.84}$$

Uma vez que X esteja determinada, podemos aplicar a DFT inversa para obter x e em seguida usar 5.82 para chegar a u.

Podemos verificar pelas expressões 5.81 e 5.84 que a DCT e sua inversa podem ser obtidas da DFT com $O(N^2)$ operações; usando um algoritmo rápido para calcular a DFT teremos, então, uma complexidade total de $O(N^2 \log N + N^2) = O(N^2 \log N)$.

5.6 Conclusões

Este capítulo contém os principais conceitos necessários para se compreender o funcionamento de métodos de codificação por transformadas. Inicialmente apresentamos os princípios básicos sobre transformadas unitárias ortogonais; como não faz parte do objetivo deste trabalho dar uma visão ampla sobre os diversos tipos de transformadas, resolvemos escolher apenas algumas delas para dar uma descrição mais detalhada, a saber: transformadas de Fourier, do cosseno e de Karhunen-Loeve. A transformada de Fourier por sua grande importância, não só para codificação, mas também para processamento de imagens e sinais em geral; a transformada do cosseno por ser, entre as que possui algoritmo rápido, a que permite melhores resultados para codificação; e, por fim, a transformada de Karhunen-Loeve, por sua importância teórica no sentido de ser ótima quanto à concentração de energia.

Dando continuidade, passamos a descrever o método básico de codificação por transformadas, e em seguida citamos rapidamente suas principais variações. Para finalizar, descrevemos alguns

algoritmos rápidos, que são indispensáveis para que se possa utilizar transformadas em aplicações em tempo real.

Procuramos fazer uma abordagem o mais sucinta possível; entretanto, sempre que possível, não abrimos mão de dar a demonstração das afirmações que foram feitas, pois acreditamos serem estas de fundamental importância quando lidamos com uma teoria matemática. Tais demonstrações permitem que o leitor tenha uma maior compreessão sobre o que está sendo exposto.

Leituras Complementares

A descrição dos principais tipos de transformadas usados em processamento de imagens pode ser encontrada em [Pra78] e [Jai89]. Em [Pra78] encontra-se ainda uma visão mais generalizada sobre operadores lineares bidimensionais, categoria na qual se enquadram as transformadas discretas. Em [Jai89] há também uma descrição teórica sobre codificação ótima por transformadas. Quanto aos algoritmos rápidos, uma grande variedade pode ser encontrada na literatura; em [Nus82] são descritas diversas variações de FFT's para uma, duas ou múltiplas dimensões; também surgiram vários algoritmos rápidos para DCT desde que Ahmed *et al* propuseram a primeira versão [ANR74]; alguns exemplos são encontrados em [CSF77], [NP78], [Mak80], [Lee84] e mais recentemente [CL91]. Existem versões baseadas ou não no FFT; usando estratégias como, por exemplo, a troca de multiplicações (que são mais caras) por adições foi possível observar uma progressiva queda no número de operações necessárias para a computação da DCT em uma e duas dimensões.

Capítulo 6

Um Esquema de Codificação Baseado em DCT

6.1 Introdução

Neste capítulo descrevemos um novo esquema de compressão de imagens que é baseado nas mesmas estratégias do MDPT/VQ. Entretanto aqui substituímos a quantização vetorial pela transformada do cosseno, de modo que decidimos chamar o método de MDPT/DCT. A transformada do cosseno se adapta de maneira bem mais natural à estratégia de transmissão progressiva, resultando em um método mais simples e flexível que o MDPT/VQ, e com um desempenho superior.

A escolha da DCT deve-se ao seu bom desempenho quanto à compressão de imagens; além do mais, com a evolução da tecnologia de VLSI, já existe uma disponibilidade de chips que implementam DCT para seqüências de imagens de vídeo (padrão de televisão) em tempo real. O impacto causado pelos padrões JPEG e MPEG, que são baseados em DCT, também tem levado a indústria de processadores a investir na produção de tais chips, de modo que o custo computacional da DCT muito em breve não mais representará um problema.

Iniciamos o capítulo dando alguns detalhes sobre a codificação por DCT, como alocação de bits e quantização escalar. Em seguida damos uma descrição detalhada do MDPT/DCT e de suas variações, juntamente com a análise dos resultados experimentais. Por fim, fazemos uma comparação entre este novo método e o MDPT/VQ.

6.2 Detalhes sobre a codificação por DCT

No Capítulo 5 vimos que todo método de codificação por transformadas tem como elementos fundamentais, além da transformada, a alocação de bits e os quantizadores escalares. Nesta seção damos detalhes sobre estes dois elementos. Descrevemos o algoritmo de alocação de bits que utilizamos e mostramos como foram projetados os quantizadores escalares.

No esquema desenvolvido, optamos por utilizar DCT em blocos de 8×8. Com blocos maiores pode-se atingir resultados melhores, mas com maior custo computacional; além do mais, a escolha de blocos de 8×8 favorece a implementação em *hardware* com a tecnologia disponível atualmente.

6.2.1 Alocação de Bits

O algoritmo que descrevemos aqui é baseado em uma solução apresentada por Jain [Jai81, Jai89] que leva em consideração as variâncias dos coeficientes da imagem transformada. No nosso caso, estimamos estas variâncias selecionando diversas imagens diferentes e calculando suas transformadas, dividindo-as em blocos de 8×8.

Para representar o algoritmo, vamos considerar $\sigma^2(k,l)$ como sendo as variâncias das imagens transformadas e $b(k,l)$ como sendo o número de bits alocados na posição (k,l). Temos então:

passo 1. Inicializa $b(k,l) = 0$, para todo k,l.

passo 2. Dada a função de distorção f, encontra os índices k,l tais que

$$D_f(k,l) = \sigma^2(k,l) f(b(k,l))$$

é máxima, e então faz,

$$b(k,l) = b(k,l) + 1$$

passo 3. Se $\sum_k \sum_l b(k,l) =$ número total de bits, pára; caso contrário volta ao **passo 2.**

Neste algoritmo, normalmente, supõe-se uma situação simplificada em que a função de distorção é da forma,

$$f(n) = a2^{-bn} \tag{6.1}$$

Na prática, utilizamos uma função do tipo,

$$f(n) = r^{-n} \tag{6.2}$$

o que implica simplesmente em dividir o valor de $\sigma^2(k,l)$ por r a cada vez que um bit é alocado na posição (k,l). A constante r foi determinada experimentalmente.

Apesar desta ser uma versão bastante simplificada do algoritmo apresentado por Jain, conseguimos obter resultados bastante satisfatórios. A Figura 6.1 mostra as alocações que obtivemos para blocos de 8×8 mantendo uma média de 1 e 2 bits por pixel.

6.2.2 Quantizadores Escalares

Nós nos baseamos no algoritmo de Lloyd-Max (ver Apêndice B) para construir os quantizadores escalares para DCT. Para fazer uma estimativa da função de densidade de probabilidade também selecionamos algumas imagens transformadas e dividimos todos os valores dos coeficientes pelos respectivos desvios padrão[1], de modo a fazer com que os quantizadores resultantes destinem-se à variáveis com desvio padrão unitário. Isto evita a necessidade de diferentes quantizadores para cada coeficiente do bloco transformado, entretanto os valores devem ser divididos pelo desvio padrão antes de serem quantizados, e os resultados devem ser posteriormente multiplicados por esta quantidade.

Para resolver as equações B.3 e B.4 do Apêndice B utilizamos um método iterativo que define inicialmente os níveis de decisão e reconstrução como estando entre intervalos com comprimentos

[1] A variância também pode ser usada, mas obtivemos melhores resultados com o desvio padrão

(a)

8	6	5	3	2	2	1	0
6	4	2	1	1	1	0	0
5	2	1	1	0	0	0	0
3	1	1	1	0	0	0	0
2	1	0	0	0	0	0	0
2	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(b)

8	7	6	5	4	3	2	2
7	5	4	3	2	2	1	1
6	4	3	2	2	1	1	0
5	3	2	2	2	1	1	0
4	2	2	2	1	1	1	0
3	2	1	1	1	1	0	0
2	1	1	1	1	0	0	0
2	1	0	0	0	0	0	0

Figura 6.1: Alocações de bits com médias de 1 bit/pixel (a) e 2 bits/pixel (b).

iguais, e então cada um deles é recalculado a cada iteração segundo as expressões B.3 e B.4, usando métodos numéricos para estimar as integrais de B.4.

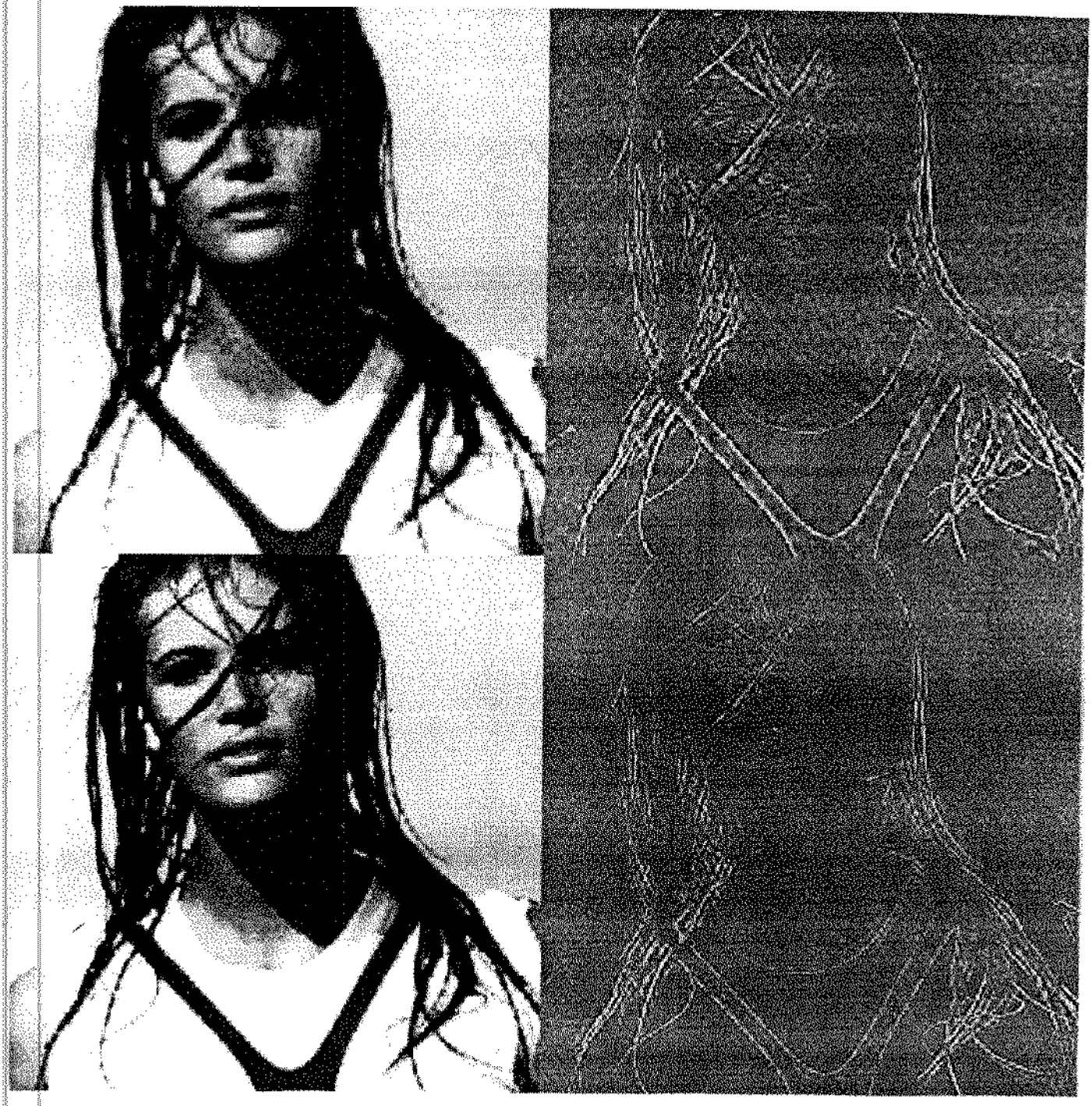
Usamos este esquema para construir os quantizadores para 1 a 8 bits (2 a 256 níveis). A tabela 6.1 mostra os níveis de decisão e reconstrução positivos do quantizador para 4 bits (16 níveis) que obtivemos.

Níves de decisão	Níveis de reconstrução
0.000 - 0.139	0,052
0.139 - 0.331	0,226
0,331 - 0,567	0,435
0,567 - 0,878	0,698
0,878 - 1,316	1,056
1,316 - 1,983	1,575
1,983 - 3,081	2,390
3,081 - ∞	3,771

Tabela 6.1: Quantizador para variáveis com desvio padrão unitário (4 bits).

6.2.3 Resultados com Codificação Intraquadros por DCT

Com o objetivo de testar a alocação de bits e os quantizadores que desenvolvemos, implementamos o método básico de codificação por DCT dividindo as imagens em blocos de 8×8. Quando mantemos uma média de 1 bit por pixel, os resultados obtidos, quanto a qualidade, são semelhantes aos obtidos com o método baseado em quantização vetorial descrito na Seção 4.2 (ver Figura 4.6). A Figura 6.2 mostra a imagem *Wet-girl* codificada à média de 1 e 2 bits por pixel, juntamente com as respectivas imagens de erro.



SNR: Acima (1 bit/píxel) - 22,68 dB; Abaixo (2 bits/píxel) - 25,95 dB.

Figura 6.2: Imagens resultantes do método básico de codificação por DCT.

6.3 O MDPT/DCT

Este método, como já dissemos, segue as mesmas estratégias do MDPT/VQ; não há nenhuma diferença no que diz respeito à detecção de movimento, mas nos demais passos do método houve uma mudança fundamental: usamos codificação por DCT ao invés de quantização vetorial. A idéia básica do método é fazer uma alocação de bits como as mostradas na Figura 6.1, e transmitir os coeficientes ao longo de mais de um quadro. Para a descrição prévia do bloco seriam transmitidos o nível DC e alguns outros coeficientes representando os componentes de mais baixa freqüência espacial. Nos quadros seguintes são transmitidos os coeficientes que representam freqüências espaciais maiores.

Uma outra alteração que introduzimos em relação ao MDPT/VQ é que a informação referente à descrição prévia do bloco, ou a qualquer dos passos de transmissão progressiva, só é transmitida se ela for capaz de produzir uma queda significativa na distorção do bloco com relação à imagem original. Pode acontecer, com alguma freqüência, que o bloco reconstruído em certo estágio de transmissão não representa uma melhora significativa em relação à representação corrente do mesmo, e o bloco reconstruído pode até ter uma distorção maior; isto acontece, por exemplo, quando é detectado um movimento muito suave, de modo que ao dar a descrição prévia do bloco a distorção torna-se maior do que se o bloco não fosse atualizado, podendo haver desta maneira uma transmissão desnecessária de informação. Para evitar este problema, a distorção do bloco reconstruído é comparada com a distorção da representação corrente do bloco: a informação só é transmitida se houver uma queda significativa na distorção. Com este procedimento conseguimos obter tanto um aumento na taxa de compressão quanto uma melhora na qualidade das imagens reproduzidas.

O método tem como entrada, além das imagens originais, uma alocação de bits dividida em n regiões r_i ($i = 0, \ldots, n-1$). No momento em que é detectado movimento no bloco, faz-se sua descrição prévia, quantizando-se os coeficientes indicados na região r_0 e substituindo os restantes por zeros; a distorção do bloco reconstruído é medida e subtraída da distorção corrente; se a diferença for maior que um limiar L_1 a informação é transmitida, caso contrário não há transmissão. Caso não seja detectado novo movimento no bloco passamos para o esquema de transmissão progressiva, que pode ser visto como uma máquina de estados finita cuja função de transição de estados depende do estado corrente e da distorção do bloco reconstruído. No momento em que é detectado movimento estamos no estado inicial e_0 e a descrição prévia do bloco pode ser considerada como sendo o primeiro passo de transmissão progressiva, após o qual passamos ao estado e_1; a partir daí o processamento e a transição de estados ocorre da seguinte maneira: supondo-se que estamos em um estado e_i tal que $1 \leq i \leq n-1$,

1. Mede a distorção do bloco corrente D_c e compara com o limiar L_1.
2. Se $D_c > L_1$: atualiza os coeficientes indicados na região r_i e reconstrói o bloco aplicando a transformada inversa; calcula nova distorção D_r; se $D_c - D_r > L_1$ transmite informação, caso contrário não há transmissão; estado passa a ser e_{i+1}.
3. Se $D_c \leq L_1$: não há processamento; estado passa a ser e_n.

Se estamos em um estado e_j tal que $n \leq j \leq 2n-1$, o processamento é similar, mas com algumas diferenças:

1. Mede a distorção do bloco corrente D_c e compara com um limiar L_2 ($< L_1$).

2. Se $D_c > L_2$: atualiza os coeficientes indicados na região r_{j-n} e reconstrói o bloco aplicando a transformada inversa; calcula nova distorção D_r; se $D_c - D_r > L_2$ transmite informação, caso contrário não há transmissão; estado passa a ser e_{j+1}; se $j+1 = 2n$ (e_{2n} = estado final) pára.

3. Se $D_c \leq L_2$: não há processamento; estado passa a ser e_{2n} e o processo de transmissão progressiva é encerrado.

A região r_0 contém o nível DC e outros coeficientes de mais baixa freqüência espacial, as regiões seguintes contém coeficientes correspondentes a freqüências progressivamente maiores. O uso de dois limiares diferentes, sendo o primeiro maior que o segundo, faz com que sejam atualizados em primeiro lugar apenas os coeficientes das regiões que mais influenciam na qualidade do bloco.

Neste método, além dos dois *buffers* com imagens originais para detecção de movimento e de um *buffer* com a imagem reconstruída, temos também um *buffer* que contém os coeficientes reconstruídos da imagem transformada. Na descrição prévia do bloco são atualizados os coeficientes indicados na região r_0 e os demais são substituidos por zeros; para as demais regiões são atualizados os coeficientes indicados e os outros permanecem com o mesmo valor. Caso não seja transmitida a informação de atualização, todos os coeficientes permanecem inalterados e o bloco é tratado como inativo.

Um esquema baseado em *run-length* é usado para codificar os blocos inativos, apenas os tamanhos dos vãos de blocos inativos são codificados, de modo que o decodificador possa calcular exatamente qual o bloco da imagem que deve ser alterado cada vez que é recebida uma informação de atualização. A Figura 6.3 mostra uma possível estrutura de dados para o código gerado pelo MDPT/DCT. Os comprimentos dos vãos de blocos inativos são codificados com 3 bits se forem menores que 8, caso contrário são codificados com 8 bits; vãos maiores que 255 são divididos em 2 ou mais vãos; o estado do bloco indica qual região deve ser atualizada e é codificado com $\lceil \log_2 n + 1 \rceil$ bits, onde n é o número de regiões. Esta codificação certamente não é ótima: técnicas de codificação estatística sem perda poderiam ser usadas para minimizar o volume de dados; entretanto, esta estrutura tem a vantagem de ser bastante simples e fácil de implementar, além de termos obtido resultados bastante satisfatórios com este esquema; mais adiante mostramos alguns destes resultados, que foram obtidos com diferentes versões de MDPT/DCT.

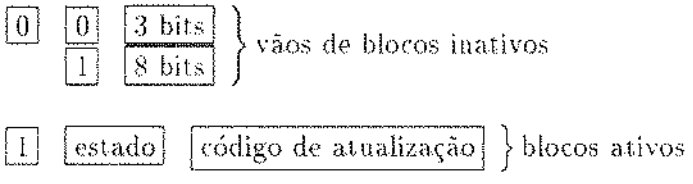


Figura 6.3: Código gerado para MDPT/DCT.

6.3.1 Variações

A qualidade das imagens reproduzidas e as taxas de compressão atingidas com o MDPT/DCT são diretamente ligadas à média de bits alocados por coeficiente e ao número de regiões em que o bloco transformado é dividido. Da média de bits por coeficiente depende a qualidade final do bloco, após todo o processo de transmissão progressiva, e do número de regiões depende a qualidade do bloco durante os passos intermediários. A taxa de compressão depende de ambos os fatores; observe-se ainda que, à medida que se diminui o número de bits para cada passo, também diminui o número de blocos para os quais a informação é transmitida, uma vez que a qualidade da reprodução tende a ser inferior, e isto faz com que a taxa de compressão aumente ainda mais.

A alocação de bits e as regiões processadas em cada passo podem ser determinadas pela própria aplicação, de acordo com suas necessidades, ou escolhidas dentro de um conjunto predefinido contendo diferentes variações. Isto faz com que este método tenha uma flexibilidade muito maior que o MDPT/VQ.

Para demonstrar esta flexibilidade, elaboramos algumas variações do MDPT/DCT, para diferentes taxas de compressão e níveis de qualidade. O procedimento para todas as variações é igual; o que muda é apenas a média de bits alocados por coeficientes e o número de regiões em que a alocação é dividida. Para facilitar a exposição, vamos chamar a versão do MDPT/DCT que usa uma alocação de b bits por coeficiente dividida em r regiões de *Versão* b/r.

Foram três as versões que usamos em nossos testes: a Versão 2/4, cuja alocação é mostrada na Figura 6.4, destina-se a aplicações onde há necessidade de se atingir taxas elevadas de compressão (superiores a 40:1) e onde seja admissível alguma degradação da imagem; a Versão 3/3, que usa a alocação mostrada na Figura 6.5, onde abrimos mão de altas taxas de compressão em favor de manter uma maior qualidade nas imagens reproduzidas; e a Versão 3/4, intermediária entre as outras duas, que usa a alocação que é mostrada na Figura 6.6.

$$
(r_0)\begin{bmatrix} 8 & 7 & 6 & 0 & 0 & 0 & 0 & 0 \\ 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}
\quad
(r_1)\begin{bmatrix} 0 & 0 & 0 & 5 & 4 & 0 & 0 & 0 \\ 0 & 5 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}
$$

$$
(r_2)\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 3 & 2 & 0 \\ 0 & 0 & 0 & 3 & 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 2 & 2 & 0 & 0 & 0 \\ 0 & 3 & 2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 2 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}
\quad
(r_3)\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 2 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 & 1 & 1 & 0 \\ 0 & 0 & 0 & 2 & 1 & 1 & 1 & 0 \\ 0 & 2 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}
$$

Figura 6.4: Alocações de bits para Versão 2/4.

(r_0)									(r_1)									(r_2)								
	8	8	7	6	0	0	0	0		0	0	0	0	5	4	3	0		0	0	0	0	0	0	0	3
	8	6	5	0	0	0	0	0		0	0	0	4	3	3	0	0		0	0	0	0	0	0	2	2
	7	5	0	0	0	0	0	0		0	0	4	3	3	0	0	0		0	0	0	0	0	2	2	2
	6	0	0	0	0	0	0	0		0	4	3	3	0	0	0	0		0	0	0	0	2	2	2	2
	0	0	0	0	0	0	0	0		5	3	3	0	0	0	0	0		0	0	0	2	2	2	2	1
	0	0	0	0	0	0	0	0		4	3	0	0	0	0	0	0		0	0	2	2	2	2	1	1
	0	0	0	0	0	0	0	0		3	0	0	0	0	0	0	0		0	2	2	2	2	1	1	1
	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0		3	2	2	1	1	1	1	1

Figura 6.5: Alocações de bits para Versão 3/3.

(r_0)									(r_1)								
	8	8	7	6	0	0	0	0		0	0	0	0	5	4	0	0
	8	6	0	0	0	0	0	0		0	0	5	4	0	0	0	0
	7	0	0	0	0	0	0	0		0	5	4	0	0	0	0	0
	0	0	0	0	0	0	0	0		6	4	0	0	0	0	0	0
	0	0	0	0	0	0	0	0		5	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0		4	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0

(r_2)									(r_3)								
	0	0	0	0	0	0	3	3		0	0	0	0	0	0	0	0
	0	0	0	0	3	3	2	2		0	0	0	0	0	0	0	0
	0	0	0	3	3	2	0	0		0	0	0	0	0	0	2	2
	0	0	3	3	0	0	0	0		0	0	0	0	2	2	2	2
	0	3	3	0	0	0	0	0		0	0	0	2	2	2	2	1
	0	3	0	0	0	0	0	0		0	0	2	2	2	2	1	1
	3	2	0	0	0	0	0	0		0	0	2	2	2	1	1	1
	3	2	0	0	0	0	0	0		0	0	2	1	1	1	1	1

Figura 6.6: Alocações de bits para Versão 3/4.

6.3.2 Resultados

Foram processadas, com as diferentes versões, as mesmas seqüências usadas para testar o MDPT/VQ, e obtivemos resultados significativamente superiores, como mostramos a seguir.

Versão 2/4

A Tabela 6.2 sumariza os resultados que obtivemos quanto à taxa de compressão e relação sinal/ruído com as seqüências *hand3* e *talk2* codificadas pela Versão 2/4; os quadros 31 e 63 da seqüência *hand3* reconstruída são mostrados na Figura 6.7, juntamente com as imagens de erro.

Seqüência	Taxa Média		Taxa Mínima	
	Compressão	SNR (dB)	Compressão	SNR (dB)
hand3	48.00:1	28.81	25,45:1	27,76
talk2	101.01:1	30.95	45,33:1	29,42

Tabela 6.2: Resultados obtidos com a Versão 2/4.

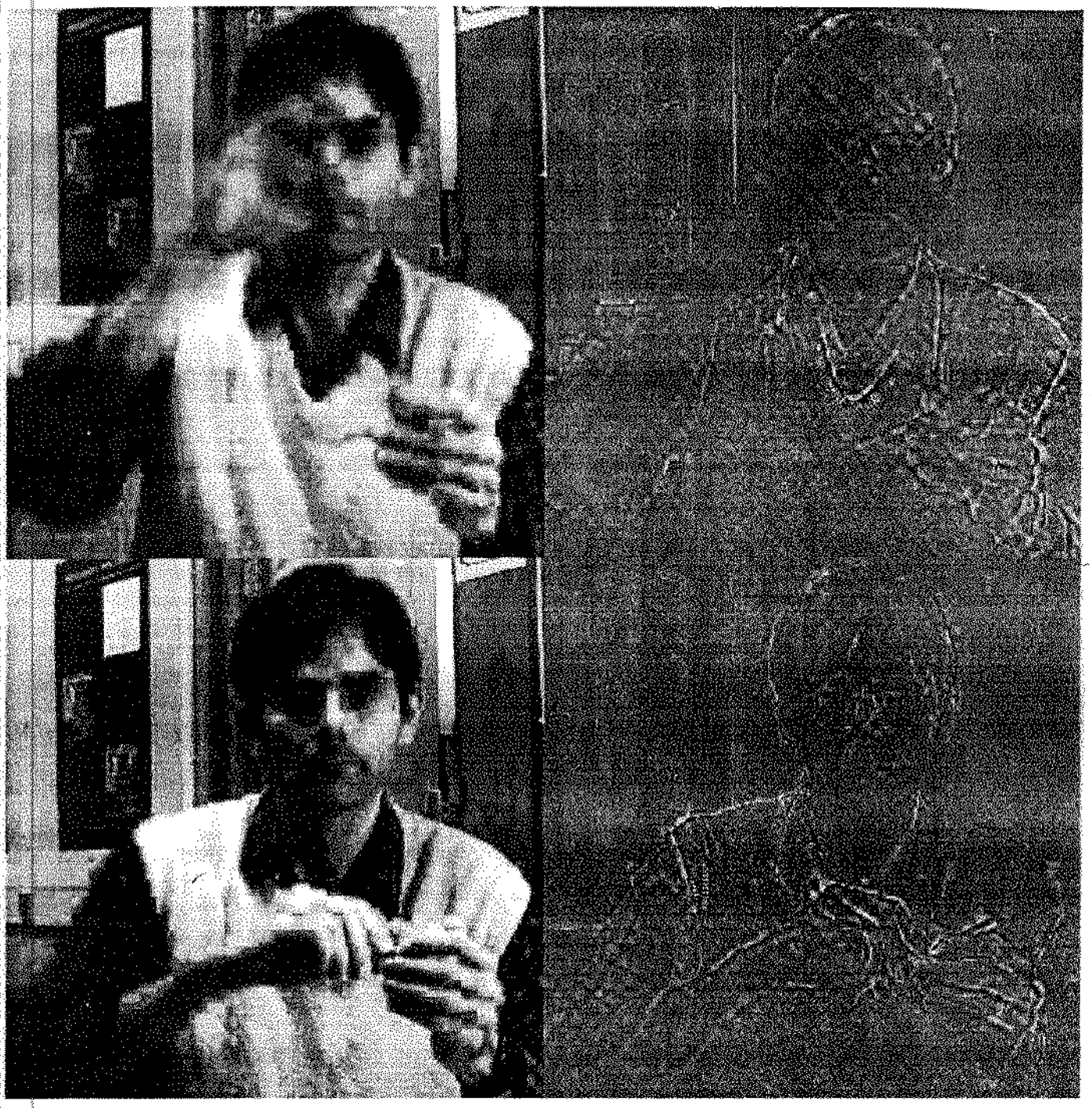
Os principais defeitos visíveis nas imagens concentram-se nas arestas em movimento ou em regiões com muitos detalhes, onde muitas vezes é possível visualizar as fronteiras entre os blocos. Olhando-se mais atentamente, pode-se também perceber alguns erros em regiões sem movimento que foram reconstruídas pelo processo de transmissão progressiva; entretanto, estas falhas são pequenas e pouco interferem na qualidade da imagem. Quando há uma maior exigência quanto à qualidade estes defeitos podem ser diminuídos usando-se uma alocação que mantenha uma maior média de bits por coeficientes, como no caso das outras duas versões.

De uma maneira geral as falhas são um pouco menores que as produzidas pelo *Block Eight* (Seção 4.3.1), e como as imagens são mostradas em movimento não chegam a representar uma degradação excessiva, especialmente quando consideramos a elevada taxa de compressão que atingimos (mais de duas vezes maior que a atingida pelo *Block Eight*). O resultado final pode ser considerado satisfatório para várias aplicações.

Versão 3/3

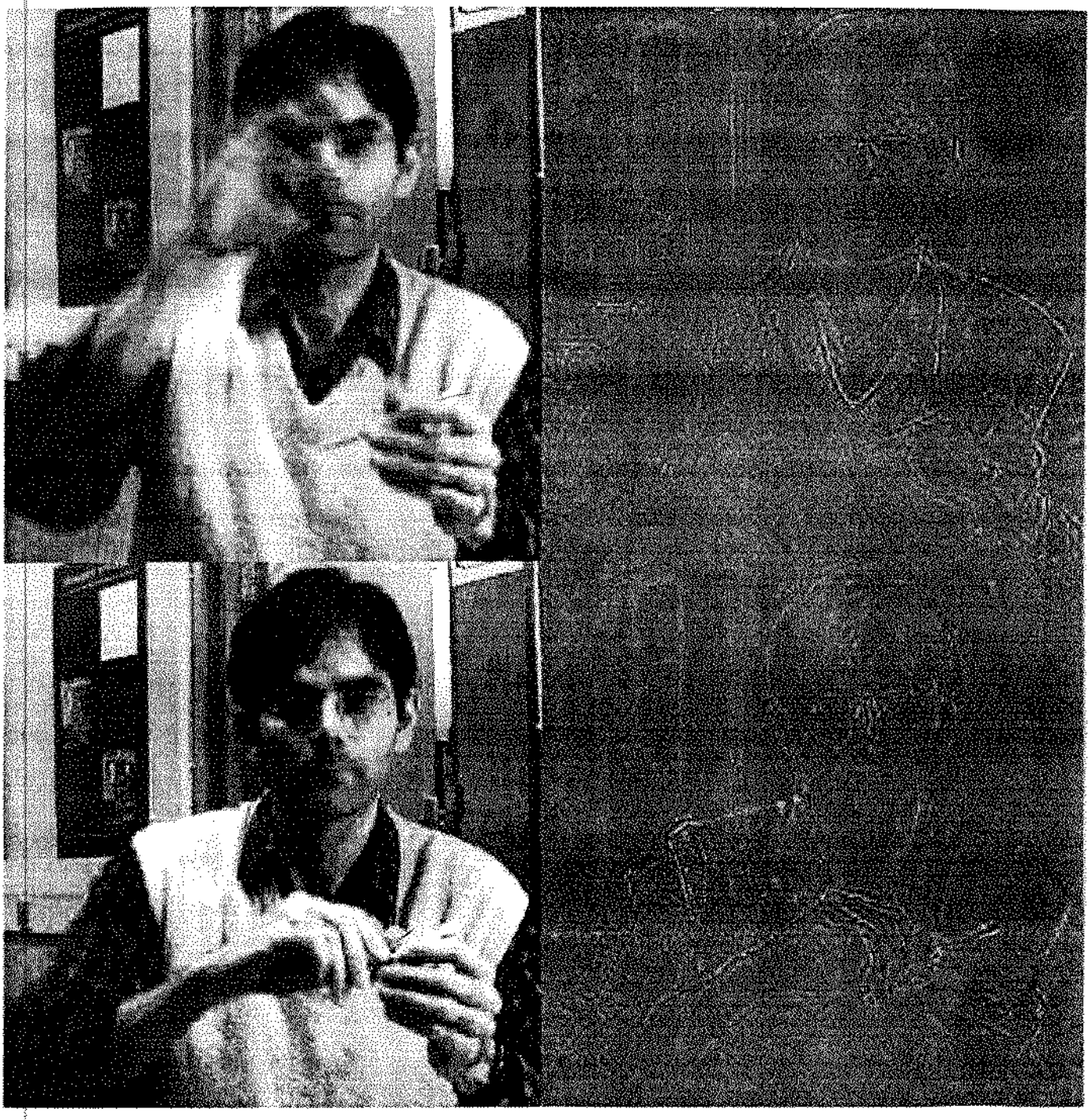
Nesta versão, como a média de bits alocados por coeficiente foi aumentada de 2 para 3, e o número de regiões foi diminuído de 4 para 3, a quantidade média de bits processados em cada passo de transmissão progressiva dobrou em relação à versão anterior; isto resultou em uma melhora bastante significativa na qualidade da imagem, mas sob o preço de uma queda também significativa na taxa de compressão. Contudo, os resultados que obtivemos superam os do *Block Four* (Seção 4.3.1) tanto na qualidade quanto na taxa de compressão, e são sumarizados na Tabela 6.3. Os quadros 31 e 63 da seqüência *hand3* são mostrados na Figura 6.8, junto com as imagens de erro.

Ainda existem algumas falhas nas imagens reproduzidas, principalmente nas arestas em movimento, mas são pouco visíveis quando observamos as imagens dinamicamente. O esquema de transmissão progressiva reproduziu os blocos sem movimento de maneira bem mais fiel e as fronteiras entre os blocos tornaram-se bem menos visíveis. As taxas de compressão que atingimos,



SNR: quadro 31 - 28,91 dB; quadro 63 - 28,64 dB.

Figura 6.7: Seqüência *hand3* codificada pela Versão 2/4.



SNR: quadro 31 - 31,26 dB; quadro 63 - 30,80 dB.

Figura 6.8: Seqüência *hand3* codificada pela Versão 3/3.

Seqüência	Taxa Média		Taxa Mínima	
	Compressão	SNR (dB)	Compressão	SNR (dB)
hand3	17,39:1	30,91	11,73:1	29,98
talk2	29,22:1	32,42	16,88:1	31,25

Tabela 6.3: Resultados obtidos com a Versão 3/3.

apesar de serem bem menores que as da Versão 2/4, ainda são satisfatórias para muitos casos, e a qualidade das imagens pode ser considerada boa e suficiente para um grande número de aplicações.

Versão 3/4

Esta Versão serve para dar mais uma demonstração da flexibilidade do MDPT/DCT; ela é intermediária entre as outras duas e permite atingir níveis moderados de compressão e qualidade. A Tabela 6.4 mostra alguns resultados obtidos e a Figura 6.9 mostra os quadros 31 e 63 da seqüência *hand3* reproduzida.

Seqüência	Taxa Média		Taxa Mínima	
	Compressão	SNR (dB)	Compressão	SNR (dB)
hand3	27,45:1	29,91	16,24:1	28,90
talk2	48,13:1	31,74	25,65:1	30,35

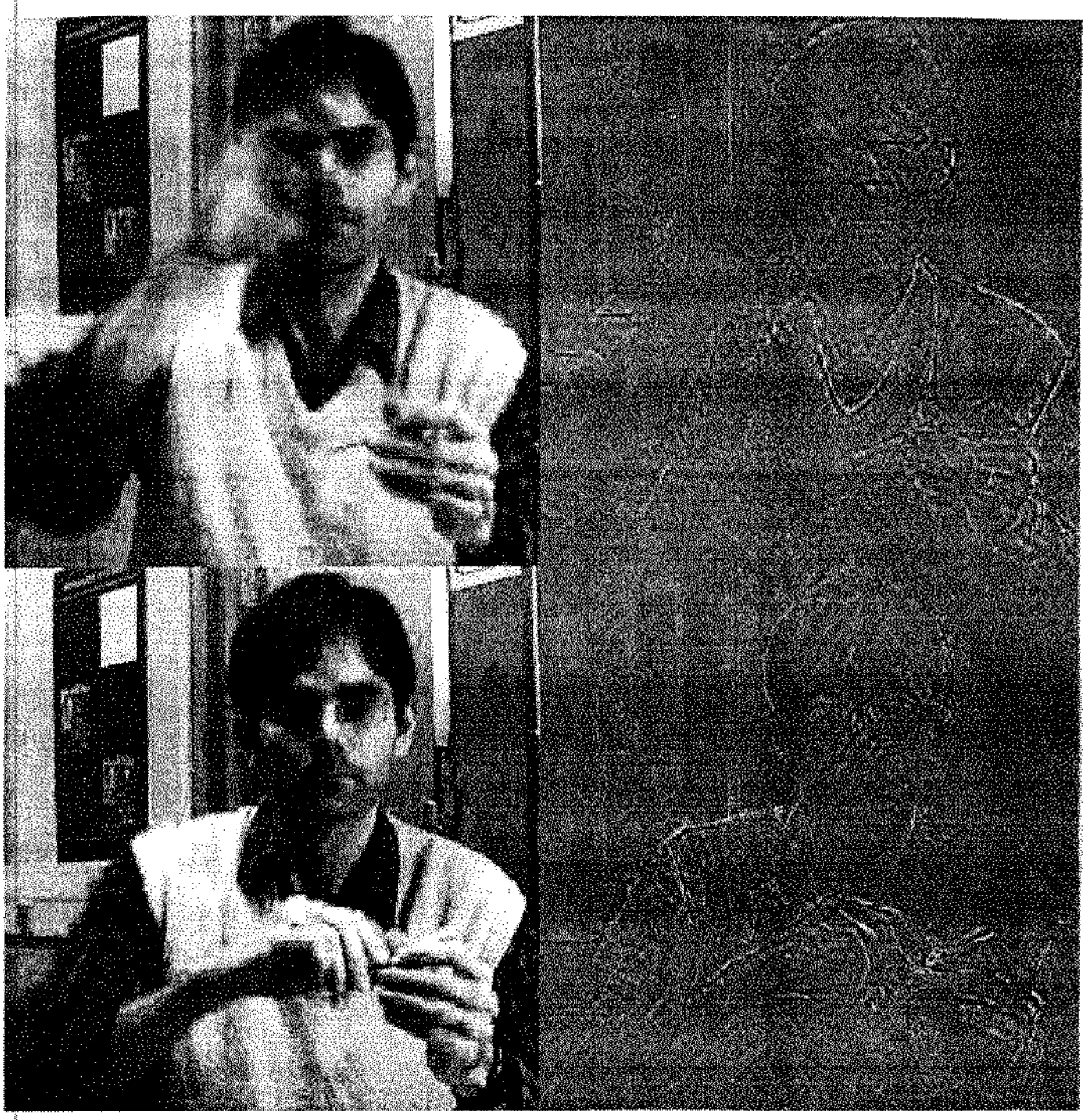
Tabela 6.4: Resultados obtidos com a Versão 3/4.

O nível de qualidade das imagens reproduzidas é bem maior do que o atingido com o *Block Eight* e com uma taxa de compressão que, na média, chega a ser um pouco superior, o que prova definitivamente a superioridade deste método em relação ao MDPT/VQ.

6.4 *Differential MDPT*

No Capítulo 5, citamos uma variação de codificação por transformada chamada codificação híbrida, que combina transformadas com DPCM. Com base neste método, surgiu a idéia de tentar combinar DPCM com o MDPT/DCT, dando origem ao que podemos chamar de *Differential MDPT*. A idéia básica foi a de codificar por DPCM cada um dos coeficientes resultantes da transformada usando como predição os valores dos coeficientes reconstruídos no quadro anterior: como os erros de predição tendem a ter uma amplitude muito menor, eles poderiam ser codificados com um número menor de bits.

Em um primeiro teste com este esquema, transmitindo toda a informação de atualização em um único quadro, isto é, sem transmissão progressiva, obtivemos resultados bastante animadores, conseguimos diminuir pela metade o volume de dados transmitido para cada bloco sem que houvesse prejuízo visível na qualidade das imagens. Entretanto, quando incluímos a transmissão progressiva no método, começaram a surgir alguns problemas como podemos ver na Figura 6.10. Ela mostra



SNR: quadro 31 - 30,20 dB; quadro 63 - 29,82 dB.

Figura 6.9: Seqüência *hand3* codificada pela Versão 3/4.

quadros originais e reconstruídos da seqüência *hand*. Nos quadros reconstruídos é possível verificar que várias arestas do fundo continuam presentes durante a passagem da mão sobre elas, dando uma espécie de efeito de transparência que não existe na imagem original e que também não aparece quando toda a informação de atualização é transmitida de uma só vez. Então, o que estaria causando este efeito? A resposta é simples: quando estamos usando um esquema de transmissão progressiva, nas regiões em movimento são atualizados apenas o nível DC e alguns outros coeficientes de baixa freqüência espacial, ou seja, os coeficientes de alta freqüência espacial, que representam as arestas, não são atualizados, de modo que as arestas continuam presentes mesmo quando a mão passa diante delas.

Apesar de outras regiões da imagem terem sido bem reproduzidas este problema torna o uso desta idéia incompatível com a estratégia de transmissão progressiva, de modo que a abandonamos durante o restante desta pesquisa.

6.5 MDPT/DCT *versus* MDPT/VQ

Pelo que vimos até aqui, já é possível perceber que o MDPT/DCT tem um desempenho bastante superior ao do MDPT/VQ: foram obtidos ganhos significativos, tanto na taxa de compressão quanto na qualidade das imagens reproduzidas. Mas existem ainda outras vantagens do MDPT/DCT, tais como:

- Maior flexibilidade. O MDPT/DCT pode ser adaptado para diversos níveis diferentes de qualidade e taxa de compressão, sem sofrer nenhuma mudança estrutural; para isto basta modificar as alocações de bits.
- Maior simplicidade. O MDPT/DCT é muito mais simples e fácil de implementar, e esta simplicidade se reflete também na geração do código, que é muito mais conciso que o do MDPT/VQ.
- Economia de memória. Um dos problemas que surgem quando tentamos tornar o MDPT/VQ mais apurado é a necessidade de se utilizar dicionários muito grandes e que acabam consumindo muita memória. Este problema não ocorre com o MDPT/DCT.

Talvez a única desvantagem significativa do MDPT/DCT seja o grande custo computacional da transformada do cosseno, que é maior do que o da quantização vetorial[2], mas os avanços atuais da tecnologia nos levam a crer que muito em breve a codificação por transformadas em tempo real estará muito mais acessível, e mesmo com a tecnologia atual este problema pode muitas vezes ser resolvido com implementação em *hardware*.

6.6 Conclusões

Os experimentos com MDPT/DCT mostraram que é possível atingir resultados bastante satisfatórios com este método em diversos tipos de aplicação. Foi possível atingir um elevado padrão de qualidade com taxas de compressão superiores a 12:1, o que é suficiente para transmissão de seqüências de imagens de vídeo através de redes locais como *Ethernet* [Geu90]. Quando é necessário

[2]Quando é usado *tree-searched VQ* ou outro método de busca rápida.

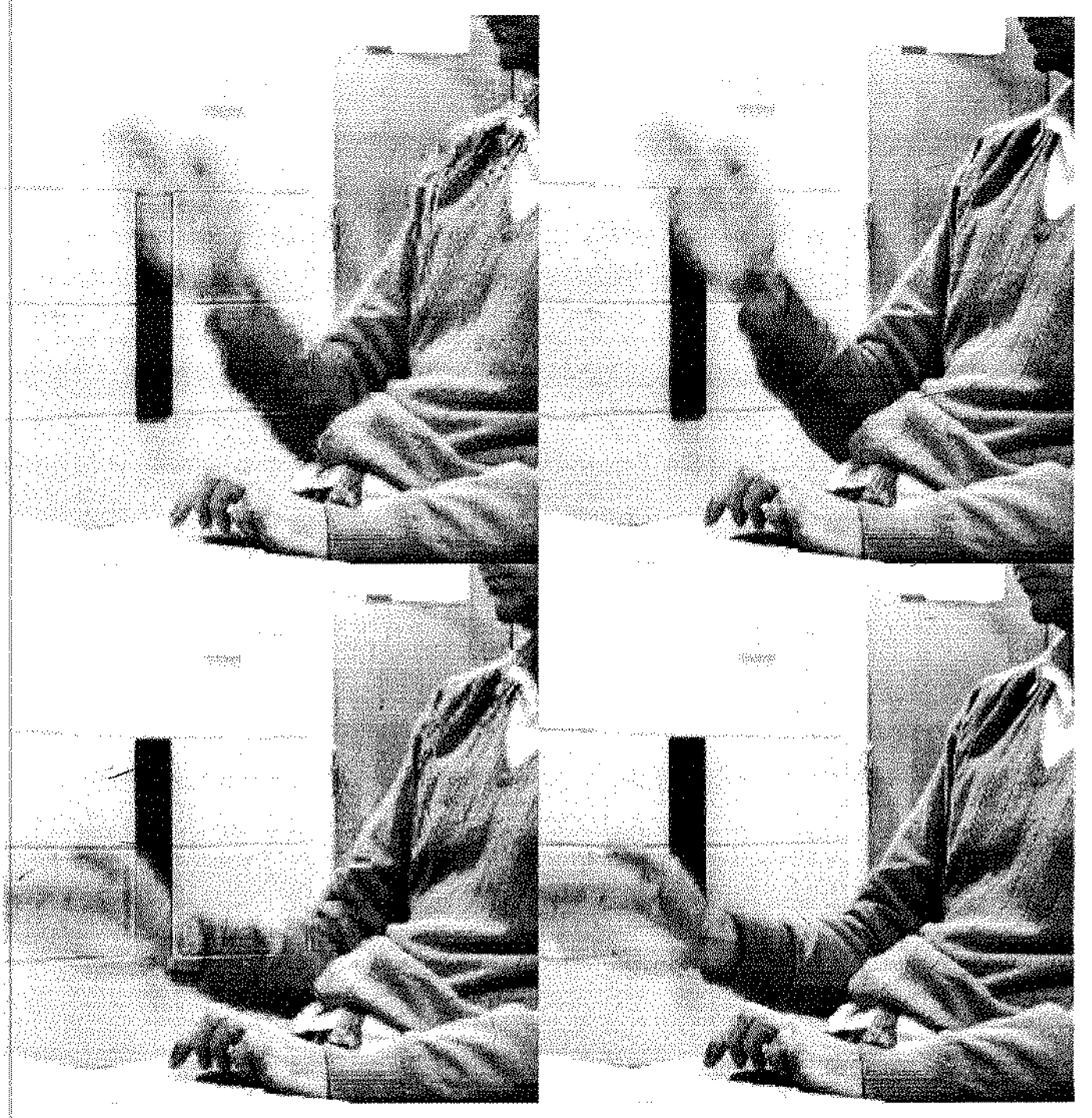


Figura 6.10: Seqüência *hand*: codificada por *Differential MDPT* (esquerda) e original (direita).

transmitir imagens de grande resolução, ou quando há outras transmissões simultâneas, fiá a necessidade de se atingir taxas maiores de compressão; para estes casos também conseguimos alguns resultados satisfatórios. Com um padrão de qualidade que pode ser considerado aceitável para muitas aplicações, conseguimos taxas de compressão superiores a 50:1, ou seja, mais que o dobro do que conseguimos com o MDPT/VQ, e com um nível de qualidade igual ou superior.

Este método tem mais um ponto forte, além do bom desempenho, que é sua simplicidade: permite com que ele seja facilmente integrável às aplicações, o que faz dele uma boa opção para sistemas que não necessitam de esquemas tão sofisticados quanto o MPEG.

Vale lembrar que o MDPT/DCT, como descrito neste capítulo, ainda não é um produto acabado e muito ainda há por fazer para torná-lo utilizável em aplicações comerciais. Entretanto o que apresentamos aqui pode ser o início do que em breve pode se tornar uma alternativa simples e eficiente para transmissão de seqüências de imagens digitais.

Capítulo 7

Conclusão

O propósito inicial da pesquisa aqui desenvolvida foi o de fazer uma extensão ao trabalho desenvolvido por Geus adaptando novas técnicas às estratégias do MDPT. Uma das propostas iniciais era de melhorar o desempenho do MDPT/VQ usando, entre outras coisas, a quantização vetorial classificada. Este foi o tema principal do Capítulo 4 e apesar de termos obtido alguns resultados positivos, de uma maneira geral eles não foram plenamente satisfatórios. Melhoramos a qualidade das imagens reproduzidas pelo *Block Eight* mas houve uma queda, mesmo que pequena, na taxa de compressão. Não foi possível usar classificação em blocos de 8×8 pois o número de variações possíveis é muito grande e seria necessário criar um esquema excessivamente complexo que certamente teria um elevado custo computacional, o que impediu que usássemos esta estratégia no *Block Eight*. Para blocos de 4×4, conseguimos desenvolver um esquema de classificação e o adaptamos ao *Block Four*. Em geral, houve alguma melhora nos resultados mas em alguns casos o desempenho ainda ficou abaixo do esperado.

A outra linha de pesquisa a que nos dedicamos nos pareceu bem mais promissora: combinar as estratégias do MDPT com a transformada do cosseno. Com este objetivo, desenvolvemos um método, que chamamos de MDPT/DCT, que também usa estratégias de detecção de movimento e transmissão progressiva, mas com algumas diferenças fundamentais. O esquema de detecção de movimento usado foi o mesmo, mas tanto a descrição prévia do bloco quanto a transmissão progressiva foram baseadas na DCT; como resultado obtivemos um esquema muito mais simples e flexível que o MDPT/VQ e com resultados significativamente superiores, como mostram os resultados que listamos no Capítulo 6.

O desenvolvimento e os testes dos métodos citados acima constituíram a fase experimental desta pesquisa; antes porém foi necessário realizar uma pesquisa teórica. Inicialmente, estudamos técnicas de compressão de imagens em geral, o que deu origem ao Capítulo 2 desta dissertação e nos deu um embasamento suficiente para justificar o porquê da escolha das técnicas que foram usadas na elaboração dos MDPT's. Em seguida, fizemos um estudo um pouco mais aprofundado sobre quantização vetorial e transformada do cosseno, com o objetivo de compreender melhor o funcionamento destas técnicas e de suas variações. Este estudo deu origem a outros dois capítulos da dissertação: o Capítulo 3, que contém uma definição formal de quantização vetorial e um *survey* sobre as principais variações de métodos baseados nesta técnica, e o Capítulo 5 é uma pequena introdução à codificação por transformadas que começa dando os conceitos gerais sobre transformadas unitárias ortogonais e as definições e propriedades das transformadas de Fourier, do cosseno e de Karhunen-Loeve, e em seguida descreve um método básico de codificação por

transformadas, algumas de suas variações, e algoritmos rápidos.

Demos aqui apenas um pequeno sumário sobre nossa pesquisa; para conclusões mais detalhadas sobre o conteúdo da dissertação, o leitor pode consultar as seções **Conclusões** que finalizam cada capítulo.

7.1 Contribuições

Muitos dos resultados que obtivemos com este trabalho podem ser de interesse para outros pesquisadores interessados nesta área. A seguir são apresentadas as principais contribuições desta dissertação.

Técnicas de Compressão de Imagens. O Capítulo 2 descreve de maneira simples e concisa as técnicas mais comumente encontrados na literatura, além dos padrões JPEG e MPEG, e pode servir como uma introdução a esta área de pesquisa. Para aqueles que tiverem um maior interesse por quantização vetorial ou codificação por transformadas os Capítulos 3 e 5 também podem ser bastante úteis.

MDPT/VQ. Conseguimos melhorar este esquema em alguns aspectos, de modo que foi possível obter imagens com um padrão de qualidade que pode ser considerado aceitável para algumas aplicações.

Algoritmo de Classificação. Foi desenvolvido um algoritmo de classificação para blocos de 4×4 que mostrou um bom desempenho quanto a separação das classes e com um baixo custo computacional. Ele foi posteriormente adaptado ao *Block Four* com o qual foi possível obter alguns resultados positivos. Apesar deste algoritmo ter sido projetado visando quantização vetorial com blocos de 4×4, ele pode ser facilmente adaptado para blocos de outros tamanhos e utilizado para outros fins: quando tentamos utilizá-lo em quantização vetorial de blocos de 8×8 os resultados não foram muito satisfatórios, mas houve uma visível separação entre diferentes tipos de blocos. Uma outra possível utilização desta versão do algoritmo seria combiná-la com codificação por transformadas, de modo que haveria uma diferente alocação de bits para cada classe de blocos, o que provavelmente permitiria uma codificação mais eficiente; isto tornaria possível saber com antecedência se o bloco tem aresta ou não, e se tem, em que direção.

MDPT/DCT. Esta certamente é a principal contribuição desta dissertação. Este método é superior ao MDPT/VQ em vários aspectos e com diferentes versões conseguimos atingir taxas de compressão que vão deste 12:1 até 100:1, com diversos níveis de qualidade. Estes resultados são bastante satisfatórios, principalmente levando-se em conta a simplicidade do método e acreditamos que, havendo continuidade nesta pesquisa, em futuro próximo poderemos ter uma alternativa bastante eficiente para transmissão de seqüências de imagens digitais.

Compressão de Imagens Isoladas. Apesar do tema principal desta dissertação ter sido compressão de seqüências de imagens, também fizemos alguns testes com esquemas simples de compressão de imagens isoladas: um deles usando *mean/shape VQ* com DPCM para codificação da média, e outro baseado em DCT. Os resultados que obtivemos, que foram listados nos Capítulos 4 e 6, serviram para testar os quantizadores vetoriais e escalares que foram

construídos e o algoritmo de alocação de bits para DCT. Os métodos que implementamos, apesar de simples, permitiram atingir alguns resultados razoáveis e podem ser úteis para aplicações que lidem com compressão de imagens isoladas.

7.2 Pesquisas Futuras

Tanto o MDPT/VQ quanto o MDPT/DCT têm diversos pontos que ainda podem ser melhorados, entretanto este último nos parece ser um campo bem mais promissor para posterior investigação, de modo que elaboramos alguns temas para pesquisas futuras que podem eventualmente torná-lo um esquema mais completo e eficiente. Abaixo seguem algumas sugestões:

- O tipo de código que sugerimos para o MDPT/DCT não é ótimo e certamente pode ser melhorado. Utilizando técnicas de codificação estatística, como código de Huffman ou codificação aritmética, podemos aumentar ainda de maneira significativa a taxa de compressão.

- Todos os blocos detectados com movimento são tratados de maneira idêntica pelo MDPT/DCT; em um esquema um pouco mais sofisticado, poderia ser dado um tratamento diferenciado para diferentes tipos de bloco. Uma possibilidade seria fazer diferentes alocações de bits para diferentes classes de blocos, como já sugerimos antes, e uma outra possibilidade seria codificar apenas as diferenças quando o bloco não possuir componentes de alta freqüência pois, pelos resultados que obtivemos com o *Differential MDPT* (ver Capítulo 6), podemos observar que este tipo de bloco pode ser codificado de maneira bastante eficiente usando esta estratégia.

- Toda a codificação do MDPT baseia-se em reconstruções a partir de uma única imagem base que corresponde à primeira imagem da seqüência. Isto pode causar problemas como propagação de erros e dificulta o acesso a imagens em posições aleatórias da seqüência. Em um sistema mais completo a reconstrução deveria se dar a partir, não de uma, mas de diversas imagens base distribuídas entre intervalos regulares ao longo da seqüência. Para estas imagens, deveria ser desenvolvido um esquema eficiente de compressão intraquadros, isto é, que não dependa de outras imagens da seqüência. A elaboração de tal esquema é algo que não chegamos a discutir nesta dissertação.

Enfim, não foi nosso objetivo desenvolver um sistema completo para transmissão de imagens em tempo real que fosse imediatamente utilizável em aplicações comerciais. Isto demandaria um volume de pesquisa muito maior do que o que foi desenvolvido até aqui. Entretanto, esperamos que este trabalho possa ser útil e servir de incentivo a pesquisadores que tenham interesse por esta área.

Apêndice A

Medidas de Distorção

Um critério quantitativo freqüentemente utilizado para medir a qualidade de imagens reconstruídas com relação à imagem original é o *erro médio quadrático* ou MSE (*mean square error*). Para duas imagem u e u^* de tamanho $M \times N$, esta medida de distorção é dada por,

$$\mathrm{MSE} = E[(u(m,n) - u^*(m,n))^2] \tag{A.1}$$

onde $E[\cdot]$ indica expectância matemática. Em geral, quando não se conhece as densidades de probabilidade para $u(m,n)$ e $u^*(m,n)$, esta quantidade pode ser estimada por,

$$\mathrm{MSE}' = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (u(i,j) - u^*(i,j))^2, \tag{A.2}$$

que é também conhecida como *erro quadrático integral*.

Em muitos casos o erro é representado em termos da *relação sinal/ruído* ou SNR (*signal-to-noise ratio*) que é definida em decibeis (dB) como,

$$\mathrm{SNR} = 10 \log_{10} \frac{\sigma^2}{\mathrm{MSE}} \tag{A.3}$$

onde σ^2 é a variância da imagem original, que pode também, por simplicidade, ser substituída pelo quadrado da diferença entre o valor máximo e o valor mínimo de brilho de uma imagem de referência. No caso de imagens de 256 tons de cinza a relação sinal/ruído pode ser representada por

$$\mathrm{SNR} = 10 \log_{10} \frac{255^2}{\mathrm{MSE}} \tag{A.4}$$

Em geral, quanto maior for esta quantidade melhor será a qualidade da imagem reproduzida, entretanto existe a possibilidade de que uma imagem com SNR maior tenha uma qualidade visual inferior. Isto ocorre, por exemplo, quando a distorção se concentra em algumas regiões específicas da imagem; quando o erro é distribuído de maneira uniforme por toda a imagem a qualidade visual tende a ser superior, mesmo que o SNR seja baixo.

Apêndice B

O Quantizador de Lloyd-Max

O quantizador de Lloyd-Max [Max60] minimiza o erro médio quadrático para um dado número de níveis de quantização. Seja $\mathbf{x}$ uma variável aleatória contínua definida em um intervalo $[a, b]$ e com densidade de probabilidade $p(x)$. Desejamos encontrar os níveis de transição t_k e os níveis de reconstrução r_k do quantizador Q de L níveis, tais que o erro médio quadrático

$$\epsilon = E[(\mathbf{x} - Q(\mathbf{x}))^2] = \int_a^b p(x)(x - Q(x))^2 dx \tag{B.1}$$

seja minimizado. Podemos definir ϵ como sendo uma função dos níveis de transição t_k e reconstrução r_k, ou seja, fixando $t_1 = a$ e $t_{L+1} = b$, temos,

$$\epsilon(t_1, \ldots, t_{L+1}, r_1, \ldots, r_L) = \sum_{j=1}^{L} \int_{t_j}^{t_{j+1}} p(x)(x - r_j)^2 dx \tag{B.2}$$

Para encontrar o ponto de mínimo de ϵ devemos derivá-la em relação a cada uma de suas variáveis, com exceção de t_1 e t_{L+1}, que são fixas, e igualar as expressões resultantes a zero. Supondo que $p(t_k) \neq 0$ para todo $k = 2, \ldots, L$, temos,

$$\begin{aligned}
\frac{\partial \epsilon}{\partial t_k} = p(t_k)(t_k - r_{k-1})^2 - p(t_k)(t_k - r_k)^2 &= 0 \\
\Longrightarrow \qquad (t_k - r_{k-1})^2 &= (t_k - r_k)^2 \\
\Longrightarrow \qquad -2t_k r_{k-1} + r_{k-1}^2 &= -2t_k r_k + r_k^2 \\
\Longrightarrow \qquad t_k = \frac{r_k^2 - r_{k-1}^2}{2(r_k - r_{k-1})} &= \frac{r_k + r_{k-1}}{2}
\end{aligned} \tag{B.3}$$

ou seja, os níveis de transição ótimos são dados pelas médias aritméticas entre níveis de reconstrução consecutivos; por outro lado, supondo que $t_k < t_{k+1}$ para todo $k = 1, \ldots, L$, temos,

$$\begin{aligned}
\frac{\partial \epsilon}{\partial r_k} = 2 \int_{t_k}^{t_{k+1}} p(x)(x - r_k) dx &= 0 \\
\Longrightarrow \qquad \int_{t_k}^{t_{k+1}} x p(x) dx - r_k \int_{t_k}^{t_{k+1}} p(x) dx &= 0 \\
\Longrightarrow \qquad r_k = \frac{\int_{t_k}^{t_{k+1}} x p(x) dx}{\int_{t_k}^{t_{k+1}} p(x) dx} &
\end{aligned} \tag{B.4}$$

o que significa que os níveis de reconstrução ótimos são dados pelos centros de massa em relação à função de densidade nos intervalos definidos por níveis de transição consecutivos. Juntas, as expressões B.3 e B.4 formam um sistema de equações não lineares que, na prática, pode ser resolvido por meio de métodos iterativos.

Apêndice C

Implementação dos Métodos e Análise das Imagens

Os métodos que foram descritos nos Capítulos 4 e 6 foram implementados em estações de trabalho Sun Sparcstation sob o sistema operacional SunOS (baseado em UNIX) utilizando a linguagem C++ com programação orientada a objetos. As implementações tiveram o objetivo apenas de medir as taxas de compressão atingidas e gerar as imagens reconstruídas para uma posterior análise visual. Não houve a preocupação de desenvolver sistemas que fossem capazes de codificar e decodificar as imagens em tempo real e gerar código para transmissão, pois para isto precisaríamos de *hardware* específico para executar algumas operações, como a transformada do cosseno, de maneira suficientemente rápida.

Para analisar as imagens reconstruídas e compará-las com as originais, nós usamos uma ferramenta gráfica denominada Vide que foi desenvolvida no Departamento de Ciência da Computação da UNICAMP em um projeto realizado paralelamente ao que deu origem a esta dissertação, e que teve o objetivo de criar um ambiente de apoio ao estudo de técnicas de compressão de imagens. O programa foi implementado com o mesmo equipamento e linguagem citados acima e com interface gráfica baseada em X-Windows [XW90], utilizando as bibliotecas de programação XLib e XView e seguindo as especificações do padrão OpenLook [OL90]. O projeto foi dirigido especialmente ao processamento de imagens de até 256 tons de cinza (8 bits por pixel), mas também permite a visualização de imagens coloridas. Dentre as facilidades oferecidas pelo Vide temos:

- Conversão de imagens coloridas para monocromáticas (tons de cinza).
- Lente de aumento: permite o aumento de uma região indicada pelo cursor do mouse por um fator entre 2 e 32 vezes; permite também verificar numericamente o valor de brilho de cada pixel.
- Filtros: contém diferentes filtros para extração de ruído, entre eles: filtro de média, filtro de mediana e sigma.
- SNR: permite medir a relação sinal/ruído entre duas imagens ou duas seqüências de imagens; no caso de seqüências há também a possibilidade de traçar gráficos indicando a variação de distorção.
- Imagens de erro: permite gerar imagens com as diferenças, pixel a pixel, entre duas imagens.

- Métodos de compressão de imagens: permite codificar imagens por diferentes métodos, entre eles: DPCM, subamostragem com interpolação, quantização vetorial e codificação por DCT.

O programa ainda está em fase de desenvolvimento e futuramente deve ser acrescido de outros módulos voltados para compressão ou processamento de imagens em geral. Entretanto, mesmo no estado atual já constitui uma ferramenta bastante útil para visualização e comparação de imagens geradas por diferentes métodos de compressão de imagens.

Bibliografia

[ANR74] Ahmed, N., Natarajan, T., and Rao, K. R. Discrete cosine transform. *IEEE Trans. Computers*, 23:90–93, January 1974.

[BGGM80] Buzo, A., Gray Jr., A. H., Gray, R. M., and Markel, J. D. Speech coding based on vector quantization. *IEEE Trans. Acoust., Speech and Signal Proc.*, 28(5):562–574, October 1980.

[CL91] Cho, Nam Ik and Lee, Sang Uk. Fast algorithm and implementation of 2-D discrete cosine transform. *IEEE Trans. Circuits and Systems*, 38(3):297–305, March 1991.

[CSF77] Chen, W.-H., Smith, H., and Fralick, S. C. A fast computational algorithm for the discrete cosine transform. *IEEE Trans. Communications*, 25(9):1004–1009, September 1977.

[DM79] Delp, E. J. and Mitchell, O. R. Image compression using block truncation coding. *IEEE Trans. Communications*, 27(9):1335–1342, September 1979.

[Dre87] Dreizen, Howard M. Content-driven progressive transmission of grey-scale images. *IEEE Trans. Communications*, 35(3):289–296, March 1987.

[Fig87] Figueiredo, Djairo G. *Análise de Fourier e Equações Diferencias Parciais*. Instituto de Matemática Pura e Aplicada - CNPq, Rio de Janeiro, 1987.

[GBS86] Goldberg, M., Boucher, P. R., and Shlien, S. Image compression using adaptive vector quantization. *IEEE Trans. Communications*, 34(2):180–187, February 1986.

[Ger82] Gersho, Allen. On the structure of vector quantizers. *IEEE Trans. Inf. Theory*, 28(2):157–166, March 1982.

[Geu90] Geus, Paulo L. *Approaches to Live Image Trasmission Between Workstations Over Limited-Bandwidth Networks*. PhD thesis, University of Manchester, Manchester, 1990.

[Gra84] Gray, Robert M. Vector quantization. *IEEE ASSP Magazine*, pages 4–29, April 1984.

[Hab71] Habibi, Ali. Comparison of nth-order DPCM encoder with linear transformation and block quantization techniques. *IEEE Trans. Communications Tech.*, 19(6):948–956, December 1971.

[Huf52] Huffman, David A. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, September 1952.

[Jai81] Jain, Anil K. Image data compression: A review. *Proceedings of the IEEE*, 69(3):349–389, March 1981.

[Jai89] Jain, Anil K. *Fundamentals of Digital Image Processing.* Prentice Hall Inc., New Jersey, 1989.

[KIK85] Kunt, M., Ikonomopoulos, A., and Kocher, M. Second-generation image-coding techniques. *Proceedings of the IEEE*, 73(4):549–574, April 1985.

[Lan84] Langdon Jr., Glen G. An introduction to arithmetic coding. *IBM J. Res. Develop.*, 28(2):135–149, March 1984.

[LBG80] Linde, Y., Buzo, A., and Gray, R. M. An algorithm for vector quantizer design. *IEEE Trans. Communications*, 28(1):84–95, January 1980.

[Le 91] Le Gal, Didier. MPEG: A video compression standard for multimedia applications. *Communications of the ACM*, 34(4):46–58, April 1991.

[Lee83] Lee, Jong-Sen. Digital image smoothing and the sigma filter. *Comp. Vision, Graphics and Image Process.*, 24:255–269, 1983.

[Lee84] Lee, Byeong Gi. A new algorithm to compute the discrete cosine transform. *IEEE Trans. Acoust., Speech and Signal Proc.*, 32(6):1243–1245, December 1984.

[Mak80] Makhoul, John. A fast cosine transform in one and two dimensions. *IEEE Trans. Acoust., Speech and Signal Proc.*, 28(1):27–34, February 1980.

[Man89] Manber, Udi. *Introduction to Algorithms: A Criative Approach.* Addison-Wesley, New York, 1989.

[Max60] Max, J. Quantizing for minimum distortion. *IRE Trans. Inform. Theory*, 6:7–12, 1960.

[MPG85] Musmann, H. G., Pirsch, P., and Grallert, H.-J. Advances in picture coding. *Proceedings of the IEEE*, 73(4):523–548, April 1985.

[NK88] Nasrabadi, N. M. and King, R. A. Image coding using vector quantization: A review. *IEEE Trans. Communications*, 36(8):957–971, August 1988.

[NL80] Netravali, A. N. and Limb, J. O. Picture coding: A review. *Proceedings of the IEEE*, 68(3):366–406, March 1980.

[NP78] Narasimha, M. J. and Peterson, A. M. On computation of the discrete cosine transform. *IEEE Trans. Communications*, 26(6):934–936, June 1978.

[Nus82] Nussbaumer, Henri J. *Fast Fourier Transform and Convolution Algorithms.* Springer-Verlag, Berlin, 1982.

[OL90] *Open Look: Graphical User Interface Application Style Guidelines.* Sun Microsystems, 1990.

[Pra78] Pratt, William K. *Digital Image Processing.* John Wiley & Sons, New York, 1978.

[RG86] Ramamurthi, B. and Gersho, A. Classified vector quantization of images. *IEEE Trans. Communications*, 34(11):1105–1115, November 1986.

[RL79] Rissanen, J. and Langdon Jr., G. G. Arithmetic coding. *IBM J. Res. Develop.*, 23(2):149–162, March 1979.

[Sam84] Samet, Hanan. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, June 1984.

[SB66] Schwartz, J. W. and Baker, R. C. Bit-plane encoding: A technique for source encoding. *IEEE Trans. Aerospace Electron. Syst.*, 2(4):384–392, July 1966.

[SG84] Sabin, M. J. and Gray, R. M. Product code vector quantizers for waveform and voice coding. *IEEE Trans. Acoust., Speech and Signal Proc.*, 32(3):474–488, June 1984.

[SN77] Sharma, D. K. and Netravali, A. N. Design of quantizers for DPCM coding of picture signals. *IEEE Trans. Communications*, 25(11):1267–1274, November 1977.

[Wal91] Wallace, Gregory K. The JPEG still picture compression standard. *Communications of the ACM*, 34(4):30–45, April 1991.

[Wel84] Welch, Terry A. A technique for high performance data compression. *IEEE Computer*, 17(6):8–19, June 1984.

[XW90] *The X Window System Series*, volume 1, 2 e 7. O'Reilly & Associations, 1990.

[ZL77] Ziv, J. and Lempel, A. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory*, 23(3):337–343, May 1977.

Índice

Q

R

S

T

V