UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE COMUNICAÇÕES

IMPLEMENTAÇÃO DO CODIFICADOR DE VÍDEO H.264 COM TRANSFORMADA FLICT

Autor
Kleber Teraoka

Orientador
Prof. Dr. Max Henrique Machado Costa

Banca Examinadora:
- Prof. Dr. Max Henrique Machado Costa (FEEC/UNICAMP)
- Prof. Dr. Dalton Soares Arantes (FEEC/UNICAMP)
- Prof. Dr. José Geraldo Chiquito (FEEC/UNICAMP)
- Prof. Dr. Sandro Adriano Fasolo (INATEL)
- Prof. Dr. Yuzo Iano (FEEC/UNICAMP)

Dissertação apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Engenharia Elétrica.

Campinas, Dezembro de 2003

Resumo

O padrão de codificação de vídeo H.264/AVC representa uma grande evolução em relação aos sistemas de codificação existentes, reduzindo substancialmente a taxa de bits. Assim como os padrões anteriores da família MPEG, o H.264 não define normas de implementação do sistema. Em vez disso, a recomendação define uma sintaxe em comum (*bitstream*) e o método de decodificação associado. Deste modo, sistemas de codificação com diferentes características podem ser desenvolvidos, garantindo interoperabilidade e diversidade entre produtos. O trabalho apresentado propõe uma implementação do codificador de vídeo H.264 com uma transformada alternativa denominada FLICT - *Floating/Integer Cosine Transform*. O sistema proposto utiliza precisão em ponto flutuante na transformada direta (etapa de quantização/normalização) e cálculo em ponto fixo na transformada inversa. Deste modo, os coeficientes transformados são obtidos com maior precisão, diminuido o erro médio das imagens reconstruídas e aumentando a relação sinal-ruído. O sistema proposto mantém compatibilidade com a recomendação H.264 e diferentemente de codificadores baseados na DCT, não apresenta descasamento entre codificador e decodificador.

Abstract

The H.264/AVC video coding standard represents a great improvement over existing video coding systems, providing significant bit-rate savings. Like the other standards from MPEG family, the H.264 standard does not define strict rules for its implementation. Instead, the recommendation defines a common bitstream and its associated decoding method. As a result, coding systems with different features can be developed, ensuring interoperability and diversity among products. The present work proposes an H.264 codec implementation using an alternative transform named FLICT (*Floating/Integer Cosine Transform*). The proposed system makes use of floating point precision in the forward transform (quantisation/normalisation step) and fixed point arithmetic in the inverse transform. As a result, the transform coefficients are obtained with improved accuracy, reducing the average error of the reconstructed images and improving the signal-to-noise ratio. The proposed codec maintains compliance with the H.264 recommendation and differently from others DCT-based codecs, it does not lead to a mismatch between encoder and decoder.

Aos meus pais,
Terumi e Eduardo

"L'essentiel est invisible pour les yeux."
- Antoine de Saint-Exupéry

Agradecimentos

Agradeço aos meus queridos pais Eduardo e Terumi e aos meus irmãos Kátia e Fabiano pelo apoio e suporte; Ao professor Max, que desde os tempos de graduação é meu orientador tanto para a ciência como para a vida; Aos professores Dalton Soares Arantes, José Chiquito, Sandro Fasolo e Yuzo Iano pelas valorosas sugestões. À Karla, pelo incentivo e companheirismo, sem os quais teriam tornado esta jornada muito mais árdua; Aos funcionários da FEEC pela prestatividade; À CAPES e ao convênio Ericsson/Unicamp pelo apoio financeiro.

Sumário

Lista de Figuras

Lista de Tabelas

Abreviaturas

ATSC:	*Advanced Television Systems Committee*
AVC:	*Advanced Video Coding*
BPP:	*Bit per Pixel*
CABAC:	*Context Adaptive Binary Arithmetic Coding*
CAVLC:	*Context Adaptive Variable Length Coding*
CDMA:	*Code Division Multiple Access*
CODEC:	*Coder-Decoder*
DC:	*Direct Current*
DiffServ:	*Differentiated Services*
DCT:	*Discrete Cosine Transform*
DSL:	*Digital Subscriber Line*
DST:	*Discrete Sine Transform*
DVB-T:	*Digital Video Broadcasting - Terrestrial*
EvDO:	*Evolution Data Only*
FFT:	*Fast Fourier Transform*
FLICT:	*Floating/Integer Cosine Transform*
ICT:	*Integer Cosine Transform*
IEC:	*International Electrotechnical Commission*
IntServ:	*Integrated Services*
ISDB-T:	*Integrated Services Digital Broadcasting - Terrestrial*
ISO:	*International Standardization Organization*
ITU-T:	*International Telecommunication Union - Telephony*
JPEG:	*Joint Photography Experts Group*
JVT:	*Joint Video Team*
KLT:	*Karhunen-Loève Transform*
MPEG:	*Moving Pictures Experts Group*

MPLS: *Multi Protocol Label Switching*
MSE: *Mean Squared Error*
NAL: *Network Abstract Layer*
PSNR: *Peak Signal to Noise Ratio*
QoS: *Quality of Service*
RMSE: *Root Mean Squared Error*
SAE: *Sum of Absolute Errors*
SBTVD: *Sistema Brasileiro de Televisão Digital*
SNR: *Signal-to-Noise Ratio*
VCEG: *Video Coding Experts Group*
VLC: *Variable Length Code*

1 INTRODUÇÃO

As técnicas de compressão de sinais contribuiram de forma decisiva para a disseminação das aplicações multimídia e para o consumo em massa de conteúdo audiovisual de formato digital. A utilização de tais técnicas aliada às recentes tecnologias de acesso e transporte de redes permite o surgimento de novos produtos e serviços de vídeo, reduzindo cada vez mais os custos de armazenamento e de transporte associados.

Dentre os vários padrões de compressão audiovisual destacam-se os do grupo MPEG (*Moving Pictures Experts Group*) [1]. As versões já desenvolvidas incluem o MPEG-1, MPEG-2 e recentemente o MPEG-4. Este último destaca-se pela codificação baseada em objetos e pela adição de interatividade. O perfil de codificação avançada do MPEG-4 (Parte 10), ainda em desenvolvimento, representa uma grande evolução em relação aos padrões precendentes, reduzindo substancialmente a taxa de bits necessária para representação do vídeo comprimido.

As técnicas de compressão utilizadas nos padrões MPEG são bastante conhecidas, tais como codificação por transformada, estimação, compensação de movimento e utilização de códigos de comprimento variável. Neste capítulo são apresentados algumas das ferramentas utilizadas para compressão de imagens, assim como conceitos

gerais sobre estas técnicas de compressão.

1.1 Compressão por transformadas

A compressão de imagens por transformadas consiste em transformar os valores de *pixels* de uma imagem em um conjunto menor de coeficientes transformados. Em imagens típicas, a energia do sinal se concentra em um pequeno número de coeficientes (baixas freqüências) e a energia dos coeficientes (momento ordinário de segunda ordem) tende a decair com o aumento de freqüência [2]. Assim, através da transformação da imagem do domínio espacial para o domínio da freqüência, é possível obter uma representação da imagem em uma forma mais apropriada ou mais compacta.

Existem diversas transformadas reportadas na literatura [3], como a DCT (*Discrete Cosine Transform*) [4] [5], DST (*Discrete Sine Transform*), KLT (*Karhunen-Loève Transform*), FFT (*Fast Fourier Transform*), Hadamard e a ICT (*Integer Cosine Transform*) [6]. Em compressão de imagens, a transformada mais utilizada é a DCT, por apresentar melhor relação entre eficiência na compactação de energia e complexidade computacional. A DCT é a transformada utilizada no padrão de compressão de imagens JPEG (*Joint Photography Experts Group*) [7] e em vários codificadores de vídeo como o MPEG-2 [8].

Após a transformação, a imagem é quantizada. Este passo introduz perdas na compressão (por isso é considerada *lossy compression*) e consiste em dividir os coeficientes transformados por uma matriz de quantização, do mesmo tamanho da imagem (divisão elemento-a-elemento). Se os elementos da matriz forem todos iguais, então a quantização é dita uniforme, uma vez que os todos os coeficientes freqüenciais serão divididos por um mesmo valor. Na quantização não-uniforme, a magnitude dos elementos da matriz de quantização são maiores nas altas freqüências e menores nas baixas. Este tipo de quantização privilegia as baixas freqüências em detrimento das freqüências mais altas, uma vez que o sistema visual humano é menos sensível a rápidas variações do sinal.

A maior parte da informação da imagem se concentra nas baixas freqüências, o que equivale a dizer que os valores dos coeficientes nesta região (da imagem transformada) são tipicamente maiores do que nas regiões de alta freqüência. A quantização resulta em uma matriz de coeficientes quantizados cujos valores mais significativos estão concentrados nas baixas freqüências. Após a quantização, a imagem quanti-

zada apresenta muitos zeros consecutivos ou valores pequenos nas freqüências mais altas, o que permite codificar estas seqüências para alcançar uma maior compressão.

A matriz de quantização pode ser definida de modo a controlar ou atingir uma taxa de bits desejada para uma determinada imagem. Tipicamente, esta matriz é multiplicada por fatores de escala constantes, dando origem a diferentes passos de quantização. Deste modo, a taxa de bits pode ser controlada através dos passos definidos.

Após a quantização, os valores quantizados são codificados (codificação de entropia). Tipicamente, o código utilizado é o código de Huffman [3], que garante o menor comprimento médio das palavras para uma fonte estácionária.

A Figura 1.1 ilustra o esquema de codificação de imagens. A imagem X é transformada nos coeficientes freqüenciais Y que são quantizados, resultando na matriz de coeficientes quantizados Z. A matriz Z é codificada utilizando o código apropriado. A reconstrução da imagem é realizada através da decodificação de entropia, quantização inversa e transformada inversa. É importante observar que a imagem reconstruída X* não é igual a X, pois a etapa de quantização introduz perdas (Y é diferente de Y*).

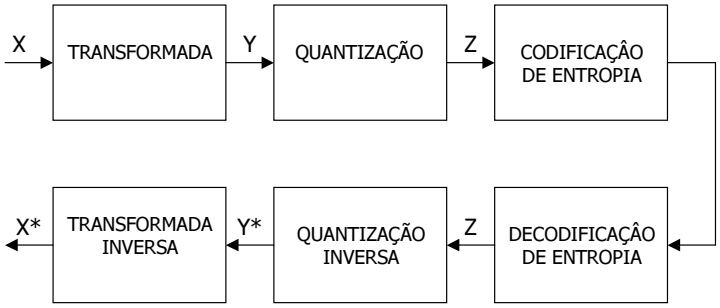


Figura 1.1: Esquema de codificação por transformada.

1.2 Estimação e compensação de movimento

Na compressão de seqüências de vídeo, utilizam-se dois tipos de codificação: intraquadro e interquadro. A codificação intraquadro é análoga à codificação descrita na seção anterior e elimina redundâncias espaciais. A codificação interquadro tem o objetivo de eliminar redundâncias temporais, isto é, as existentes de um quadro para outro.

Um vídeo é composto por seqüências de imagens que são muito similares entre si. Deste modo, ao invés de codificar cada quadro separadamente no modo *intraframe*, é possível comparar os quadros individualmente e codificar a diferença existente entre elas. Isto permite uma redução na taxa de bits necessária para representação da seqüência.

A estimação e compensação de movimento são técnicas que consistem em rastrear blocos de imagens de um quadro para outro e computar a diferença entre eles. Tendo um determinado quadro como referência, é possivel encontrar um bloco no quadro subseqüente que minimiza o erro em relação ao bloco de referência. A posição do bloco resulta num vetor de movimento que indica a direção e o sentido do movimento do bloco de um quadro para outro. A utilização de vetores de movimento e a codificação dos resíduos associados aos blocos permitem eliminar as redundâncias existentes entre quadros, aumentando a taxa de compressão.

1.3 Medidas de distorção

Como já mencionado anteriormente, a compressão por transformadas (*lossy compression*) introduz erros de reconstrução devido ao passo de quantização. Uma avaliação objetiva da qualidade da imagem reconstruída pode ser realizada calculando o erro médio existente entre a imagem original e a reconstruída. Uma medida bastante utilizada é o erro médio quadrático (MSE - *Mean Squared Error*), dado por

$$\mathbf{MSE} = \frac{1}{MN} \sum_{i,j=0}^{M-1,N-1} \left[x(i,j) - x^*(i,j)\right]^2, \tag{1.1}$$

onde M e N são os números de *pixels* da imagem nas coordenadas i e j respectivamente, $x(i,j)$ é o valor do *pixel* da imagem original na posição (i,j) e x^* é o valor do *pixel* da imagem reconstruída na posição (i,j). A raiz do erro MSE (RMSE - *Root Mean Squared Error* é definida por

$$\mathbf{RMSE} = \sqrt{\mathbf{MSE}} = \sqrt{\frac{1}{MN} \sum_{i,j=0}^{M-1,N-1} \left[x(i,j) - x^*(i,j)\right]^2}. \tag{1.2}$$

Outras medidas de distorção comumente utilizadas são a razão sinal-ruído (SNR - *Signal to Noise Ratio*) e a razão sinal-ruído de pico (PSNR- *Peak Signal to Noise*

Ratio). A relação SNR é dada por

$$\mathbf{SNR} = 10\log_{10}\left(\frac{\frac{1}{MN}\sum_{i,j=0}^{M-1,N-1}\left[x^*(i,j)\right]^2}{\mathbf{MSE}}\right)[dB]\,, \tag{1.3}$$

ao passo que para imagens com definição de 8 bits/*pixel* temos:

$$\mathbf{PSNR} = 10\log_{10}\left(\frac{255^2}{\mathbf{MSE}}\right) = 20\log_{10}\left(\frac{255}{\mathbf{RMSE}}\right)[dB]\,. \tag{1.4}$$

2

O PADRÃO DE COMPRESSÃO DE VÍDEO H.264

O H.264/AVC (*Advanced Video Coding*) [9] [10] [11] é o atual projeto de padronização de codificação de vídeo dos grupos ITU-T *Video Coding Experts Group* (VCEG) e ISO/IEC *Moving Pictures Experts Group* (MPEG). O grupo VECG iniciou os trabalhos de padronização em 1997. No fim de 2001, devido à notável qualidade do sistema em desenvolvimento, o grupo ISO/IEC MPEG uniu-se ao VECG formando o grupo *Joint Video Team* (JVT), tendo como objetivo o desenvolvimento de um padrão de vídeo unificado, resultando ao mesmo tempo, numa nova adição à família MPEG-4 (Parte 10) [12] e em uma recomendação ITU-T (H.264). Nesta data, os trabalhos do grupo JVT ainda estão em desenvolvimento e a oficialização do padrão é esperada para final de 2003.

O H.264 abrange uma ampla gama de aplicações de comunicação de vídeo, tanto bidirecionais (vídeo-telefonia) quanto unidirecionais (armazenamento, *streaming* [12] e difusão). O projeto do H.264 se beneficia de avanços em técnicas de compressão bem conhecidas (como codificação por transformadas, predição e estimação de movimento), resultando em um sistema de alto desempenho. Não há uma técnica de

codificação isolada que seja responsável pela melhora da eficiência do codificador, mas são os diversos avanços em seus blocos constituintes somados que resultam em um ganho significativo em relação aos padrões predecentes.

2.1 O Codificador/Decodificador H.264

Assim como em padrões anteriores, tais como o MPEG-1, MPEG-2 e MPEG-4, o H.264 não define um codificador/decodificador (*codec*). Alternativamente, o padrão define a sintaxe do vídeo codificado (*bitstream*) e o método associado para decodificá-lo. Deste modo, o padrão permite a existência de diferentes implementações, o que garante alternativas de diferenciação entre os desenvolvedores de *hardware* e *software*.

2.1.1 Codificador

Embora possam existir diferentes implementações, um codificador compatível com o H.264 é constituído pelos elementos funcionais da Figura 2.1.

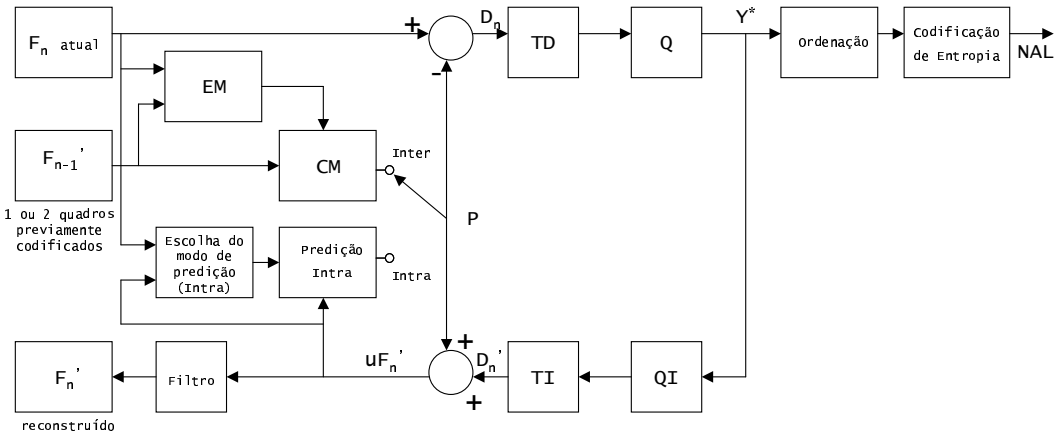


Figura 2.1: Codificador H.264.

O codificador recebe um quadro F_n que é particionado em unidades de macrobloco (16 x 16 *pixels*). Cada macrobloco é codificado no modo de predição *Intra* ou *Inter*, gerando um macrobloco de predição P, formado com base em um ou mais quadros reconstruídos.

No modo *Intra*, P é formado a partir de amostras do quadro atual F_n que foram codificados, decodificados e reconstruídos ($uF_n^{'}$). Existem até nove modos de predição *Intra* diferentes, e o modo escolhido é aquele que minimiza a soma dos erros absolutos (SAE - *Sum of Absolute Errors*), computados entre o quadro atual e o reconstruído ($uF_n^{'}$).

No caso de predição *Inter*, P é formado a partir de predições de estimação (EM) e compensação de movimento (CM) de um ou mais quadros de referência.

O macrobloco predito P é subtraído do macrobloco atual, resultando no resíduo ou macrobloco de diferença D_n. Este último é transformado através de uma transformada de bloco (TD) e quantizado (Q), resultando em um conjunto de coeficientes de transformação Y^*. A seguir, os coeficientes são reordenados e codificados (codificação de entropia), e junto com informações necessárias para decodificar o macrobloco, formam o *bitstream* comprimido. Por fim, o *bitstream* é passado à camada NAL-*Network Abstraction Layer* para transmissão ou armazenamento.

Os coeficientes quantizados X são decodificados para reconstruir quadros futuros. Assim, os coeficientes passam pela quantização inversa (QI) e pela transformada inversa (TI), resultando no macrobloco de resíduo $D_n^{'}$. Como o processo de quantização introduz perdas, o resíduo $D_n^{'}$ é diferente de D_n.

O macrobloco predito P é adicionado à $D_n^{'}$ para criar um macrobloco reconstruído $uF_n^{'}$. A seguir, um filtro é aplicado para reduzir os efeitos de bloco e um quadro de referência recuperado é criado de a partir de uma série de macroblocos $F_n^{'}$.

2.1.2 Decodificador

O decodificador da Figura 2.2 recebe um *bitstream* comprimido da camada NAL. Os elementos então são decodificados e reordenados para recuperar os coeficientes quantizados Y^*, que passam pela quantização e transformada inversa, resultando em $D_n^{'}$. O decodificador é capaz de criar o macrobloco P a partir de informações do cabeçalho. P é adicionado a $D_n^{'}$ para produzir $uF_n^{'}$ que é então filtrado para criar o macrobloco $F_n^{'}$.

O propósito do caminho de reconstrução do codificador é de garantir que o tanto o codificador quanto o decodificador usem quadros de referência idênticos para criar o quadro de predicão P. Caso as referências para recriar os macroblocos P não sejam idênticas, ocorrerá propagação de erro, que resulta no descasamento (*mismatch*) entre o codificador e o decodificador.

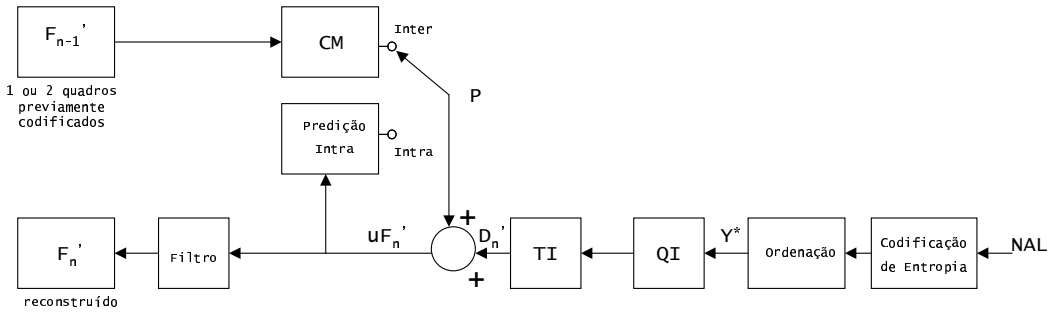


Figura 2.2: Decodificador H.264.

2.2 Codificação por transformada e quantização

Nesta etapa os macroblocos residuais são transformados e quantizados. A amostragem padrão dos quadros de entrada é 4:2:0 e os macroblocos são transmitidos na ordem ilustrada pela Figura 2.3.

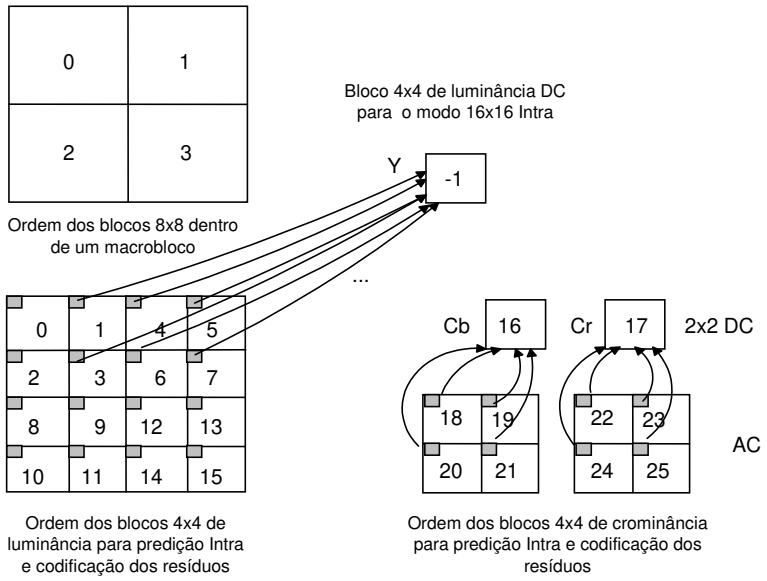


Figura 2.3: Ordem dos macroblocos.

Diferentemente de padrões precedentes, o H.264 não utiliza a transformada do coseno discreto (DCT) como transformada básica. O perfil *baseline* utiliza três transformadas diferentes, dependendo do tipo do resíduo a ser codificado:

- Transformada de Hadamard para a matriz 4x4 de coeficientes de luminância DC dos macroblocos (somente para o modo de predição *Intra* 16x16).
- Transformada de Hadamard para a matriz 2x2 de coeficientes de crominância DC dos macroblocos
- Transformada ICT (*Integer Cosine Transform*) 4x4 para todos os outros blocos residuais

2.2.1 Blocos 0-15, 18-25

Nestes blocos a transformada utilizada é a ICT, que é baseada na DCT mas que difere nos seguintes aspectos:

- É uma transformada inteira e todas as operações podem ser realizadas com aritmética inteira de 16 bits
- O núcleo da transformada é implementada sem multiplicações e somente com deslocamentos binários e adições
- A normalização e a quantização são implementadas numa etapa única, reduzindo o número total de multiplicações.
- A transformada inversa é especificada pelo padrão H.264 e, se implementada corretamente, não ocorre descasamento entre codificador e decodificador

2.2.2 Bloco -1

Quando um macrobloco é codificado no modo de predição 16x16 *Intra*, a componente 16x16 de luminância é predita a partir dos *pixels* vizinhos e cada bloco 4x4 é transformado utilizando a ICT. Os coeficientes DC de cada bloco 4x4 são transformados novamente utilizando a transformada de Hadamard 4x4.

2.2.3 Blocos 16 e 17

Cada componente de crominância em um macrobloco é formado a partir de quatro blocos 4x4. Os coeficientes DC de cada bloco 4x4 são agrupados em um bloco 2x2 e codificados utilizando uma transformada de Hadamard 2x2.

Um modo opcional denominado *adaptive block size transform* pemite o uso de outras transformadas escolhidas de acordo com o tamanho do bloco de compensação de movimento (4x8, 8x4, 8x8, 16x8, etc.).

2.3 Predição intraquadro

Quando um macrobloco é codificado no modo *Intra*, o bloco de predição P é subtraído do bloco atual antes da codificação. Para amostras de luminâncias, P pode ser formado para cada sub-bloco 4x4 ou para cada macrobloco 16x16. Há um total de nove modos de predição opcionais para cada bloco 4x4, quatro modos para cada bloco 16x16 e um modo que é sempre utilizado em cada bloco 4x4 de crominância.

2.3.1 Modos de predição *Intra* 4x4 para luminância

O bloco de predição P é calculado a partir das amostras A-Q disponíveis no quadro atual. Para preservar a decodificação independente dos quadros, a predição é realizada a partir dos *pixels* disponíveis no quadro. Os *pixels* a-p (Figura 2.4) são calculados a partir das amostras A-Q.

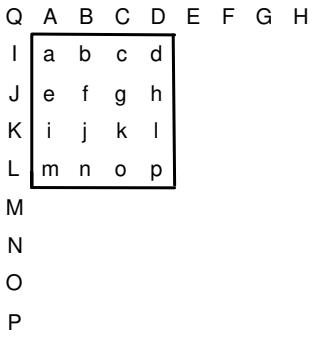


Figura 2.4: *Pixels* utilizados no modo de predição *Intra*.

Nove modos são definidos com diferentes direções (Figura 2.5):

- Modo 0 - *Vertical*:

  a=e=i=m=A
  b=f=j=n=B

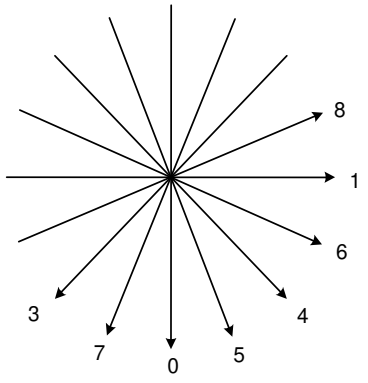


Figura 2.5: Direção das predições do modo *Intra*.

c=g=k=o=C
d=h=l=p=D

- Modo 1 - *Horizontal*:

  a=b=c=d=I
  e=f=g=h=J
  i=j=k=l=K
  m=n=o=p=L

- Modo 2 - *DC*:

  a-p são iguais a (A+B+C+D+I+J+K+L+4)$\gg$ 3, onde $\gg$ x denota deslocamento de x bits à direita

- Modo 3 - *Diagonal down-left*

  a=(A + 2B + C + I + 2J + K + 4) $\gg$ 3
  b=e=(B + 2C + D + J + 2K + L + 4)$\gg$ 3
  c=f=i=(C + 2D + E + K + 2L + M + 4) $\gg$ 3
  d=g=j=m=(D + 2E + F + L + 2M + N + 4) $\gg$ 3
  h=k=n=(E + 2F + G + M + 2N + O + 4) $\gg$ 3
  l=o=(F + 2G + H + N + 2O + P + 4) $\gg$ 3
  p=(G + H + O + P + 2)$\gg$ 2

- Modo 4 - *Diagonal down-right*

m=(J + 2K + L + 2) $\gg$ 2
i=n=(I + 2J + K + 2)$\gg$ 2
e=j=o=(Q + 2I + J + 2)$\gg$ 2
a=f=k=p=(A + 2Q + I + 2)$\gg$ 2
b=g=l=(Q + 2A + B + 2)$\gg$ 2
c=h=(A + 2B + C + 2)$\gg$ 2
d=(B + 2C + D + 2) $\gg$ 2

- Modo 5 - *Vertical-right*

  a=j=(Q + A + 1) $\gg$ 1
  b=k=(A + B + 1) $\gg$ 1
  c=l=(B + C + 1) $\gg$ 1
  d=(C + D + 1) $\gg$ 1
  e=n=(I + 2Q + A + 2) $\gg$ 2
  f=o=(Q + 2A + B + 2) $\gg$ 2
  g=p=(A + 2B + C + 2) $\gg$ 2
  h=(B + 2C + D + 2) $\gg$ 2
  i=(Q + 2I + J + 2) $\gg$ 2
  m=(I + 2J + K + 2) $\gg$ 2

- Modo 6 - *Horizontal-down*

  a=g=(Q + I + 1) $\gg$ 1
  b=h(I + 2Q + A+ 2)$\gg$ 2
  c=(Q + 2A + B+ 2) $\gg$ 2
  d=(A + 2B + C+ 2)$\gg$ 2
  e=k=(I + J + 1) $\gg$ 1
  f=l=(Q + 2I + J+ 2)$\gg$ 2
  i=o=(J + K + 1)$\gg$ 1
  j=p(I + 2J + K+ 2) $\gg$ 2
  m=(K + L + 1)$\gg$ 1
  n=(J + 2K + L + 2)$\gg$ 2

- Modo 7 - *Vertical-left*

  a=(2A + 2B + J + 2K + L + 4) $\gg$ 3
  b=i=(B + C + 1) $\gg$ 1

c=j=(C + D + 1)≫ 1
d=k=(D + E + 1) ≫ 1
l=(E + F + 1)≫ 1
e=(A + 2B + C + K + 2L + M + 4)≫ 3
f=m=(B + 2C + D + 2)≫ 2
g=n=(C + 2D + E + 2) ≫ 2
h=o=(D + 2E + F + 2)≫ 2
p=(E + 2F + G + 2) ≫ 2

- Modo 8 - *Horizontal-up*

  a=(B + 2C + D + 2I + 2J + 4) ≫ 3
  b=(C + 2D + E + I + 2J + K + 4)≫ 3
  c=e=(J + K+ 1)≫ 1
  d=f=(J +2K + L + 2) ≫ 2
  g=i=(K + L + 1) ≫ 1
  h=j=(K +2L + M + 2) ≫ 2
  l=n=(L + 2M + N + 2)≫ 2
  k=m=(L + M + 1) ≫ 1
  o=(M + N +1) ≫ 1
  p=(M + 2N + O +2)≫ 2

2.3.2 Modo de predição 16x16 para luminância

Este é um modo de predição alternativo, que segue as mesmas características da predição 4x4, porém utilizando o macrobloco inteiro. Quatro modos são definidos:

- Modo 0 - Vertical: extrapolação das amostras superiores
- Modo 1 - Horizontal: extrapolação das amostras à esquerda
- Modo 2 DC: média das amostras superiores e à esquerda
- Modo 2 Plano: uma função linear plana é obtida a partir das amostras superiores e à esquerda.

2.3.3 Modo de predição 8x8 para crominância

Este modo é semelhante ao modo 16x16 para luminância, com os mesmos quatro modos definidos. Cada bloco 8x8 de um macrobloco é predito a partir das amostras de crominância acima e/ou à esquerda daquelas já previamente codificadas e reconstruídas.

2.4 Predição interquadro

A predição interquadro permite que os quadros de uma seqüência de vídeo sejam codificadas a partir de um modelo de predição baseado em um ou mais quadros previamente codificados. O modelo é composto por amostras deslocadas espacialmente do quadro de referência (predição de compensação de movimento). O padrão H.264 utiliza compensação de movimento baseada em blocos, da mesma forma que padrões anteriores, mas com um número maior de tamanhos de bloco (até 4x4). Além disso, os vetores de movimento são calculados com precisão de 1/4 de *pixel* na componente de luminância.

2.4.1 Compensação de movimento

O H.264 utiliza compensação de movimento com blocos de diferentes tamanhos, sendo o maior deles de tamanho 16x16 e o menor de 4x4. A componente de luminância de cada macrobloco pode ser particionada em quatro blocos menores. Cada um desses blocos pode ainda ser subdividido em mais quatro subpartições como mostrado na Figura 2.6.

O método de particionamento de macroblocos em sub-blocos de compensação de movimento de tamanho variável é conhecido como compensação de movimento em árvore estruturada (*tree structured motion compensation*).

Cada partição ou subpartição necessita de um vetor de movimento que é codificado e transmitido. Além disso, a escolha do tipo de particionamento deve ser informado no *bitstream*. Partições de tamanho maior, como 16x16, 16x8 ou 8x16 requerem menos bits de sinalização para informar a escolha do vetor de movimento e do tipo de partição utilizada. Em contrapartida, o resíduo resultante da compensação pode conter mais energia em áreas mais detalhadas. Para partições de tamanho menor, como 8x4, 4x4, etc. ocorre o inverso: o resíduo apesar de conter

menos energia, são necessários mais bits para informar os vetores de movimento e as partições utilizadas. Dessa forma, a escolha do tipo de partição tem um impacto importante na capacidade de compressão do codificador. De modo geral, partições de tamanho maior são melhores para áreas mais homogêneas dentro de um quadro e partições menores são mais indicadas para áreas com mais detalhes.

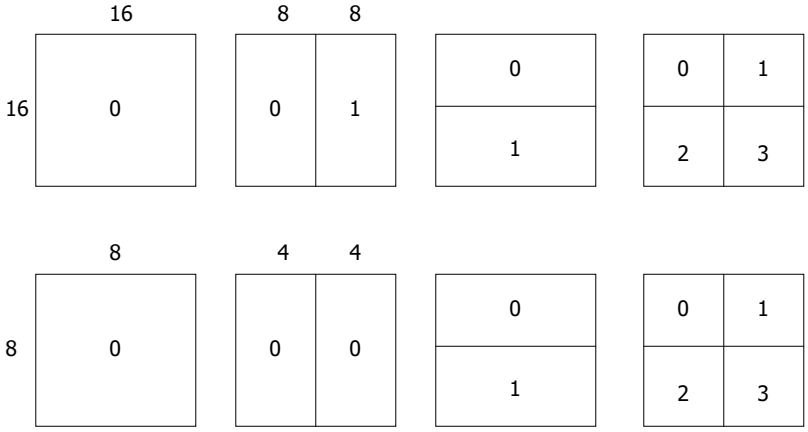


Figura 2.6: Blocos de luminância utilizados no modo de predição *Inter*.

Para as componentes de crominância, a resolução é a metade da componente de luminância. Cada bloco é particionado da mesma maneira que a luminância, sendo que os tamanhos dos blocos têm a metade dos *pixels*.

A implementação de referência [13] seleciona a partição que minimiza o resíduo e os vetores de movimento. Desse modo, o quadro codificado é subdividido em blocos de vários tamanhos, sendo que em áreas que há pouca mudança de um quadro para outro, os blocos maiores são escolhidos. Em áreas mais detalhadas, os blocos menores são os mais eficientes.

Dentro de um quadro, cada partição dentro de um macrobloco é predito a partir de uma área de mesmo tamanho no quadro de referência. O vetor de movimento tem uma resolução de 1/4 de *pixel* para a componente de luminância. Como as amostras de luminância e crominância não existem no quadro de referência, é necessário criá-los através de interpolação de *pixels* a partir de amostras vizinhas.

A Figura 2.7 ilustra uma subpartição 4x4 (a) que deve ser predita a partir de amostras vizinhas do quadro de referência. Caso a resolução do vetor de movimento seja inteiro (b), as amostras no bloco de referência existem (pontos cinzas) e o vetor de movimento é formado por números inteiros. No entanto, se um ou mais

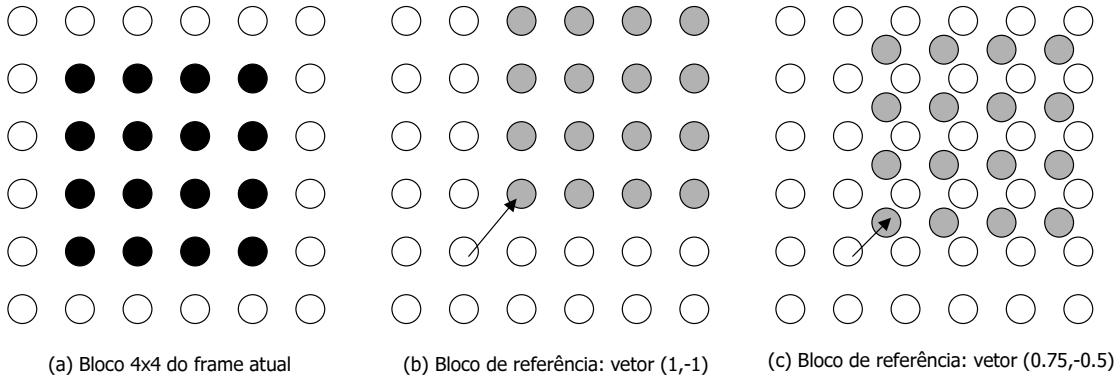


Figura 2.7: Predição inteira e *sub-pixel*.

componentes do vetor tem resolução fracionária (c), as amostras preditas devem ser geradas a partir da interpolação entre as amostras adjacentes do quadro de referência (pontos cinzas gerados pela interpolação dos pontos brancos).

A utilização de vetores de movimento com resolução fracionária melhora significativamente a taxa de compressão em relação à compensação com vetores de resolução inteira, porém a complexidade para implementação é maior, visto que para cada bloco ou sub-bloco predito é necessário realizar o cálculo da interpolação dos *pixels*.

2.5 Codificação de Entropia

Nesta etapa, os coeficientes transformados e quantizados são codificados para transmissão ou armazenamento. O padrão define dois tipos de codificação de entropia: CAVLC - *Context Adaptive Variable Length Coding* [14] e CABAC - *Context-based Adaptive Binary Arithmetic Coding* [15]. O perfil *Baseline* define o VLC como a codificação a ser utilizada enquanto que perfis mais avançados podem fazer uso da codificação CABAC.

Vários parâmetros devem ser codificados e informados ao decodificador para recuperação correta da seqüência de vídeo, tais como o método de predição para cada macrobloco, o parâmetro de quantização para cada quadro, o índice do *frame* de referência, os vetores de movimento, o padrão de codificação dos blocos, os coeficientes para cada bloco 4x4 ou 2x2 e os elementos de sintaxe de quadro, seqüência e *slice*. Dentre estes elementos, destaca-se o parâmetro *entropy coding mode* que define o

tipo de codificação a ser utilizado. Caso o modo seja configurado com "1", o código utilizado é o VLC. Caso o modo seja configurado com "0", o sistema de codificação aritmética CABAC é a utilizada para codificar e decodificar os elementos de sintaxe H.264.

2.5.1 Codificação VLC

Neste tipo de codificação, o resíduo é codificado utilizando um código de comprimento variável e adaptativo CAVLC - *Context-Adaptive Variable Length Coding*. Os outros elementos de sintaxe de comprimento variável são codificados utilizando o código de Exp-Golomb [16].

Os códigos Exp-Golomb são códigos de Golomb exponenciais que têm construção regular. Neste código, cada palavra é codificada da seguinte maneira:

$$[M \text{ zeros}]\ 1\ [INFO].$$

O campo $[M$ zeros$]$ é uma seqüência de M zeros, seguida do bit 1, seguida do campo $[INFO]$, que é um campo de M bits que carrega a informação. Cada palavra código pode ser construída baseada no índice *code_num* da seguinte maneira:

$$M = \log 2(code\_num + 1),$$
$$INFO = code\_num + 1 - 2^M.$$

A Tabela 2.1 apresenta as primeiras palavras do código Exp-Golomb.

Os resíduos referentes aos blocos 4x4 e 2x2 são codificados segundo um código adaptativo (CAVLC) que se beneficia de uma série de características dos blocos 4x4 quantizados:

- Após a predição, transformação e quantização, os blocos contém muitos zeros, especialmente nas altas freqüências. O CAVLC utiliza uma "corrida de zeros" para representar de forma compacta esta seqüência de zeros.

- Tipicamente, as componentes freqüenciais mais altas são compostas por seqüências de valor +/-1. O CAVLC sinaliza também uma "corrida de uns" (*Trailing 1s* ou *T1s*).

code_num	Palavra código
0	1
1	010
2	011
3	00100
4	00101
5	00110
6	00111
7	0001000
8	0001001
...	...

Tabela 2.1: Palavras do código Exp-Golomb.

- O número de coeficientes diferente de zero dos blocos vizinhos são correlatados. Assim, o número de coeficientes diferentes de zero são codificados através de uma tabela de *look-up*, que é escolhida de acordo com o número de coeficientes diferentes de zero dos blocos vizinhos.

- A magnitude dos coeficientes freqüenciais diferentes de zero tende a ser maior nas baixas freqüências, perto da componente DC e no começo do vetor ordenado. Nas altas freqüências ou no fim do vetor ordenado, a magnitude tende a ser menor. Desse modo, CAVLC aproveita esta característica e adapta a escolha da tabela de *look-up* ao nível do parâmetro *level* que expressa a magnitude dos coeficientes recém-codificados.

A codificação dos coeficientes com o CAVLC é formado pelas seguintes etapas:

1. Codificação do número de coeficientes diferentes de zero (*TotalCoeffs*) e "corrida de uns"(*T1s*):

   O valor de *TotalCoeffs* está entre 1 e 16 e o número de *T1s* entre 0 e 3. No caso de haver uma seqüência com mais de três "uns", somente os últimos três são codificados como *T1s*, sendo que os demais são tratados da mesma forma que os outros coeficientes. Há até três tabelas de código de comprimento

variável (Num-VLC0 até Num-VLC2) e uma fixa (Num-VLC4) que codificam o par *TotalCoeffs*/*T1s* (variável *coeff_token*). A escolha da tabela depende do número de coeficientes diferentes de zero nos blocos previamente codificados (referente aos blocos à esquerda (NL) e acima (NU) do bloco considerado). Se ambos os blocos estão disponíveis, então N=(NU+NL)/2, onde N é a variável de comparação para decidir qual tabela deve ser utilizada; se somente um deles está disponível, então o valor de N é o número de coeficientes diferentes de zero do bloco disponível. Caso não existam blocos à esquerda nem acima, então N=0. A Tabela 2.2 ilustra o critério utilizado para decisão no uso das tabelas.

N	Tabela utilizada
0,1	Num-VLC0
2,3	Num-VLC1
4,5,6,7	Num-VLC2
8 e acima	Num-VLC3

Tabela 2.2: Critério para utilização das tabelas Num-VLC.

2. Codificação do sinal de cada coeficiente em *T1*:

   O sinal de cada coeficiente em *T1* deve ser indicado em ordem reversa, isto é, começando pelo coeficiente de maior freqüência em *T1*.

3. Codificação dos níveis dos coeficientes diferentes de zero:

   Os níveis (sinal e magnitude) dos coeficientes diferentes de zero remanescentes também são codificados em ordem reversa. São utilizadas até 7 tabelas diferentes (Level_VLC0 até Level_VLC6), sendo que a escolha da tabela é adaptativa e depende da magnitude dos níveis sucessivamente codificados. As palavras da tabelas são adaptadas em ordem crescente de magnitude. A tabela Level_VLC0 é mais adaptada para codificar coeficientes de magnitudes menores enquanto que Level_VLC6 é mais eficiente para codificação dos coeficientes de magnitudes maiores.

   O limiar de decisão para mudança de tabela depende da magnitude do coeficiente diferente de zero de maior freqüência. Desse modo, à medida que

os blocos vão sendo codificados, a tabela utilizada muda, se adaptando ao nível dos coeficientes e contribuindo para maior compactação do *bitstream*. Os limiares de decisão estão definidos na Tabela 2.3.

Tabela utilizada	Limiar de decisão (nível do coeficiente de mais alta freqüência)
VLC0	0
VLC1	3
VLC2	6
VLC3	12
VLC4	24
VLC5	48
VLC6	-

Tabela 2.3: Critério para utilização das tabelas *Level_VLC*.

4. Codificação do número total de zeros antes do último coeficiente:

   O número total de zeros existente no vetor ordenado que precede o coeficiente diferente de zero de maior freqüência é codificado de forma separada, com uma tabela própria. O par *TotalZeros/TotalCoeff* define uma palavra código na tabela. A maioria dos blocos contém coeficientes diferentes de zero no começo do vetor ordenado. "Corridas de zero" que aparecem no começo do vetor não necessitam ser codificados, bastanto para isso informar o número total de zeros.

5. Codificação das "corridas de zeros":

   Cada coeficiente diferente de zero codifica o número de zeros imediatamente precedente a ele (variável *run_before*), a iniciar pelo coeficiente de mais alta freqüência. O par *TotalZeros/run_before* define uma palavra código na tabela.

2.5.2 Codificação CABAC

A codificação CABAC - *Context-adaptive Binary Arithmetic Coding* é o modo de codificação de entropia utilizado quando o parâmetro *entropy_coding_mode* é igual a 1. Este modo de codificação tem desempenho superior ao modo de codificação

CAVLC e é utilizado em perfis mais avançados que o perfil *baseline*. O CABAC faz uso de codificação aritmética [17] e seleciona um modelo de probabilidade para cada elemento de sintaxe de acordo com o contexto.

Codificação aritmética

O algoritmo da codificação aritmética pode ser entendido através do seguinte exemplo. Suponha que a seqüência de símbolos "FLICT" deva ser codificada. Cada símbolo nesta seqüência tem probabilidade de ocorrência igual a 1/5. Podemos mapear estas probabilidades em um intervalo de [0 a 1), segundo a Tabela 2.4.

Símbolo	Probabilidade	Intervalo
C	0,2	[0,0-0,2)
F	0,2	[0,2-0,4)
I	0,2	[0,4-0,6)
L	0,2	[0,6-0,8)
T	0,2	[0,8-1,0)

Tabela 2.4: Exemplo: Mapeamento das probabilidades no intervalo [0,0-1,0).

O primeiro símbolo a ser codificado (F) reside no intervalo de [0,2 a 0,4). O próximo passo do algoritmo consiste em expandir este intervalo, mapeando os intervalos de probabilidade dos símbolos seguintes no intervalo referente ao símbolo anterior. Por exemplo, o próximo símbolo (L) corresponde ao intervalo de [0,6 a 0,8), que deve ser mapeado no intervalo [0,2 a 0,4). Deste modo, o mapeamento corresponde ao intervalo de [0,32 a 0,36) (L), dentro de [0,2 a 0,4) (F). Para o símbolo (I), devemos mapear [0,4-0,6) dentro de [0,32-0,36) que resulta no intervalo [0,336-0,344). Os cálculos prosseguem até o que último símbolo seja codificado. A Tabela 2.5 ilustra todos os mapeamentos calculados.

A seqüência pode ser decodificada a partir do intervalo resultante do último símbolo codificado (T). O limite inferior 0,33728 e a distribuição de probabilidade são necessários para a decodificação, que é realizada da seguinte forma:

O limite inferior do último intervalo (0,33728) reside dentro do intervalo [0,2-0,4). Portanto, o primeiro símbolo é (F). Para decodificar o segundo símbolo, devemos subtrair o limite inferior do intervalo referente a (F) (0,2) de 0,33728 e dividir o resultado pelo valor do intervalo (0,2) referente a (F), ou seja, (0,33728-0,2)/0,2=0,6864.

Símbolo	Mapeamento em [0,0-1,0)	Mapeamentos subseqüentes
F	[0,2-0,4)	[0,2-0,4)
L	[0,6-0,8)	[0,32-0,36)
I	[0,4-0,6)	[0,336-0,344)
C	[0,0-0,2)	[0,336-0,3376)
T	[0,8-1,0)	[0,33728-0,3376)

Tabela 2.5: Exemplo: Mapeamentos subseqüentes de intervalos.

Como 0,6864 reside entre [0,6-0,8), o símbolo decodificado é (L). O terceiro símbolo pode ser decodificado fazendo (0,6864-0,6)/0,2=0,432, que é referente ao símbolo (I), pois 0,432 reside entre [0,4-0,6). Os símbolos seguintes podem ser decodificados de maneira análoga.

Seleção do modelo de contexto

A codificação CABAC é realizada através das seguintes etapas: binarização, seleção do modelo de contexto, codificação aritmética e atualização das probabilidades.

A binarização consiste em converter símbolos (coeficientes transformados, vetores de movimento, etc.) na forma binária antes da codificação aritmética. O processo é similar à conversão de um símbolo em um código VLC para transmissão posterior.

A seleção do modelo de contexto é um modelo de probabilidade para um ou mais *bins* do símbolo binarizado. O termo *bin* refere-se à posição dos bits do símbolo, sendo que o *bin 1* corresponde ao bit mais significativo, *bin 2* ao segundo bit mais significativo e assim por diante. A Tabela 2.6 ilustra os códigos binarizados utilizados para codificar os símbolos de 0 a 8 referentes ao vetor de movimento na direção x.

Para cada *bin* é escolhido um modelo de contexto. A probabilidade de ocorrência de um símbolo segue uma distribuição geométrica, pois é igual a probabilidade de ocorrência uma seqüência de bits “0” seguidos de um bit “1”, dentro de um mesmo *bin*. O modelo de contexto fornece ao codificador aritmético os dois intervalos referentes às probabilidades de ocorrência dos bits “0” ou “1” para um determinado *bin*. Deste modo, à medida que os símbolos são codificados, os modelos de contexto são atualizados incrementando-se os contadores de ocorrência dos bits “0” ou “1” e adaptando o código ao contexto.

Símbolo	Código
0	1
1	01
2	001
3	0001
4	00001
5	000001
6	0000001
7	00000001
8	000000001

Tabela 2.6: Códigos para o vetor de movimento na direção x (0 a 8).

2.6 Filtro de reconstrução

Nos processos de codificação e decodificação, os macroblocos reconstruídos são filtrados para reduzir os efeitos de bloco. O filtro é aplicado logo após a transformada inversa e ajuda a suavizar as bordas dos blocos, melhorando a aparência das imagens decodificadas.

A filtragem é aplicada nas bordas horizontais e verticais dos blocos 4x4 de luminância e dos blocos 2x2 de crominância. Cada filtragem afeta até três *pixels* de cada borda.

O filtro de reconstrução ajuda a reduzir a taxa de bits, pois o macrobloco filtrado é utilizado na compensação de movimento, o que contribui para reduzir os valores do macrobloco residual.

3
A TRANSFORMADA FLICT

A utilização de transformadas em algoritmos de compressão de imagens é amplamente conhecida. Em particular, a transformada do coseno discreto (DCT) é a mais utilizada em codificadores de imagens e vídeo, devido a sua grande capacidade de compactação de energia. No entanto, o cálculo da DCT requer o uso de aritmética de ponto flutuante, o que demanda o uso de processadores mais sofisticados e de maior custo. A transformada inteira do coseno (ICT) [6] se apresenta como uma alternativa à DCT, podendo ser calculada apenas com aritmética inteira e de forma mais rápida [18] [19]. Por ser uma aproximação da DCT, a ICT mantém a sua eficiência e pode ser implementada em processadores de ponto fixo e de custo reduzido [20] [21]. Neste capítulo apresentamos a transformada FLICT - *Floating/Integer Cosine Transform* no contexto do codificador de vídeo H.264 [9]. A FLICT é uma transformada que se baseia na ICT, utilizando aritmética de ponto fixo no cálculo da transformada inversa e operação em ponto flutuante na normalização dos coeficientes da transformada direta. Dessa forma, a FLICT melhora o desempenho da transformada ICT convencional, à medida que os coeficientes são calculados com maior precisão, ao custo de um pequeno aumento de complexidade.

3.1 A transformada inteira do coseno (ICT)

Em [6], W.-K. Cham apresenta um método para a conversão de transformadas DCTs em ICTs, que podem ser implementadas apenas com aritmética inteira. O desempenho destas transformada derivadas da DCT são muito semelhantes às originais e provêem uma liberdade de escolha adicional para os projetistas, que podem optar pela simplicidade de implementação da ICT em detrimento de um melhor desempenho propiciado pela DCT.

3.1.1 Desenvolvimento da transformada ICT

A DCT é a transformada comumente utilizada na codificação de imagens e vídeo porque é a que mais se aproxima da transformada de Karhunen-Loève, que é estatisticamente ótima. Existem outras transformadas de implementação mais simples, como as de Walsh ou Slant, mas ambas têm desempenho inferior à DCT.

A DCT mapeia um vetor $\mathbf{X}$ de tamanho $\mathbf{N}$ em um vetor $\mathbf{Y}$ de coeficientes freqüenciais através de uma transformação linear $\mathbf{Y} = \mathbf{AX}$. A matrix $\mathbf{A}$ é definida por

$$\mathbf{A}_{kn} = \mathbf{A}(k,n) = c_k \sqrt{\frac{2}{N}} \cos\left[(n+\frac{1}{2})\frac{k\pi}{N}\right], \tag{3.1}$$

onde $k, n = 0, 1, 2, ..., N-1$, e

$$c_k = \begin{cases} \sqrt{2} & \text{se } k = 0; \\ 1 & \text{se } k > 1. \end{cases} \tag{3.2}$$

A matrix DCT é ortogonal, ou seja, $\mathbf{X} = \mathbf{A}^{-1}\mathbf{Y} = \mathbf{A}^T\mathbf{Y}$. Além disso, os elementos que compõem a matrix $\mathbf{A}$ são números irracionais. Como conseqüência, quando se calcula a transformada direta e inversa em cascata, os coeficientes obtidos podem não ser os mesmos.

Deste modo, é desejável que a matriz $\mathbf{A}$ seja substituída por uma matriz ortogonal, composta exclusivamente por números inteiros. Um procedimento geral para obtenção de matrizes com estas características é sumarizada a seguir [6][22].

Passos para geração de matrizes ICT:

1. Gere uma matriz DCT $\mathbf{A}_{NxN}$ a partir da equação 3.1.

2. Construa uma matrix $\mathbf{B}_{NxN}$ substituindo os N possíveis valores de $\mathbf{A}$ com N símbolos, preservando o sinal dos elementos de $\mathbf{A}$.

3. Calcule $\mathbf{BB}^T$ e gere um conjunto de equações algébricas que forcem $\mathbf{BB}^T$ a ser uma matriz diagonal.

4. Encontre um conjunto de N números que satisfaçam um conjunto de equações algébricas geradas em (3).

Por exemplo, para a matrix DCT 4x4

$$\mathbf{A} = \begin{pmatrix} 0,5 & 0,5 & 0,5 & 0,5 \\ 0,6533 & 0,2706 & -0,2706 & -0,6533 \\ 0,5 & -0,5 & -0,5 & 0,5 \\ 0,2706 & -0,6533 & 0,6533 & -0,2706 \end{pmatrix}. \tag{3.3}$$

Podemos gerar a matrix $\mathbf{B}$ (passo 2)

$$\mathbf{B} = \begin{pmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{pmatrix}. \tag{3.4}$$

Se tomarmos $a = 1$, o produto $\mathbf{BB}^T$ é uma matriz diagonal (passo 3)

$$\mathbf{BB}^T = \begin{pmatrix} 4 & 0 & 0 & 0 \\ 0 & 2b^2 + 2c^2 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 2b^2 + 2c^2 \end{pmatrix}. \tag{3.5}$$

Por fim (passo 4), podemos gerar as matrizes ICT escolhendo b e c apropriadamente. Uma possível solução é:

$$\mathbf{C} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix}. \tag{3.6}$$

Esta matriz é idêntica à matriz de Hadamard, que apesar de bastante simples, não apresenta um ganho de codificação considerável. Um resultado melhor pode ser obtido fazendo $b = 2$ e $c = 1$:

$$\mathbf{C} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix}. \tag{3.7}$$

Uma outra forma mais simples de se obter a matriz ICT [23] é através da multiplicação da matriz DCT por um número α, seguido de arredondamento:

$$\mathbf{C} = \text{round}[\alpha \mathbf{A}], \tag{3.8}$$

onde round[.] é a função de arredondamento.

Uma matriz ICT inicialmente proposta para o padrão de compressão de vídeo H.264 [24] é obtida com $\alpha = 26$, resultando $a = 13$, $b = 17$ e $c = 7$. A escolha destes elementos torna a matriz ICT muito próxima à uma DCT escalonada, garantindo que todas as linhas da matriz tenham a mesma norma. No entanto, a escolha destes elementos resulta num ganho de alcance dinâmico igual a 52, visto que o valor máximo da soma dos valores absolutos das linhas desta matriz é $13 \times 4 = 52$ (primeira linha). Uma vez que transformada é bidimensional, ou seja, calculada nas linhas e colunas, o ganho total é $52^2 = 2704$. Como $\log_2 2704 = 11,4$, são necessários 12 bits a mais em relação ao vetor $\mathbf{X}$ de entrada para armazenar os coeficientes de transformação $\mathbf{Y}$.

Para contornar esta incoveniência, a matriz ICT adotada pelo padrão H.264 [23] é a mesma da equacão 3.7 ($a = 1$, $b = 2$ e $c = 1$), que também pode ser obtida através da equacão 3.8, fazendo $\alpha = 2,5$. Nesta matriz, a soma dos valores absolutos em qualquer linha de $\mathbf{C}$ é igual a 6, de forma que o ganho de alcance dinâmico para uma transformada bidimensional é $log_2 6^2 = 5,17$, ou seja, o armazenamento dos coeficientes de freqüência da matriz $\mathbf{Y}$ requer 6 bits a mais do que os da matriz de entrada $\mathbf{X}$. Para uma imagens com 256 níveis de resolução (8 bits/*pixel*), o resíduo necessita de 9 bits. Assim, são necessários $6+9 = 14$ bits e a ICT pode ser calculada usando aritmética de 16 bits.

Assim como a DCT, a ICT é implementada através de uma transformação linear. Os coeficientes de freqüenciais da matriz $\mathbf{Y}$ são obtidos multiplicando-se a imagem $\mathbf{X}$ pela matriz ICT $\mathbf{C}$, ou seja, $\mathbf{Y} = \mathbf{CX}$. Portanto, a implementação da transformada

requer $N^2(N-1)$ adições e N^3 multiplicações para cada bloco NxN da imagem a ser codificada.

No entanto, é possível implementar esta transformada de modo a reduzir consideravelmente o número de multiplicações [20].

Sendo a matriz ICT $\mathbf{C}$ ortogonal, temos que $\mathbf{C}\mathbf{C}^T = \mathbf{D}$, onde $\mathbf{D}$ é uma matriz diagonal. Seja $\mathbf{\Delta}$ a inversa de $\mathbf{D}$. Podemos fatorar a matriz identidade $\mathbf{I}$ como sendo $\mathbf{I} = \sqrt{\mathbf{\Delta}}\mathbf{C}\mathbf{C}^T\sqrt{\mathbf{\Delta}}$. A matriz $\mathbf{M} = \sqrt{\mathbf{\Delta}}\mathbf{C}$ é uma matriz ortonormal, com $\mathbf{M}^T = \mathbf{C}^T\sqrt{\mathbf{\Delta}} = \mathbf{M}^{-1}$ e representa a matriz ICT normalizada.

Notando que

$$\mathbf{C}^{-1} = \mathbf{C}^T\mathbf{\Delta}, \tag{3.9}$$

podemos recuperar $\mathbf{X}$ a partir de $\mathbf{Y}$, pois

$$\mathbf{X} = \mathbf{C}^{-1}\mathbf{Y} = \mathbf{C}^{-1}\mathbf{C}\mathbf{X} = (\mathbf{C}^T\sqrt{\mathbf{\Delta}})(\sqrt{\mathbf{\Delta}}\mathbf{C})\mathbf{X}. \tag{3.10}$$

A ICT bidimensional é separável, ou seja, pode ser calculada aplicando a ICT unidimensional nas colunas da imagem, seguido das linhas do resultado intermediário. Na forma matricial, isto é equivalente a pré-multiplicar o bloco de dados pela matriz ICT direta e pós-multiplicá-lo pela matriz ICT transposta. Assim,

$$\mathbf{Y} = \sqrt{\mathbf{\Delta}}\mathbf{C}\mathbf{X}\mathbf{C}^T\sqrt{\mathbf{\Delta}} \tag{3.11}$$

e

$$\mathbf{X} = \mathbf{C}^T\sqrt{\mathbf{\Delta}}\mathbf{Y}\sqrt{\mathbf{\Delta}}\mathbf{C}. \tag{3.12}$$

Uma simplificação adicional pode ser feita notando que $\mathbf{C}\mathbf{X}\mathbf{C}^T$ e $\mathbf{Y}$ são pré-multiplicados e pós-multiplicados pela matriz diagonal $\sqrt{\mathbf{\Delta}}$.

Dado duas matrizes diagonais, $\mathbf{D_1}$ e $\mathbf{D_2}$ e uma matriz qualquer $\mathbf{A}$, o produto $\mathbf{D_1}\mathbf{A}\mathbf{D_2}$ pode ser calculado de forma mais eficiente multiplicando-se termo a termo todos os elementos de $\mathbf{A}$ pelos elementos do produto $\mathbf{D_1}\mathbf{1}\mathbf{D_2}$, onde $\mathbf{1}$ é uma matriz do mesmo tamanho de A, composta exclusivamente por números "1". Este procedimento reduz o número de multiplicações pela metade. Assim, temos

$$\mathbf{Y} = \mathbf{C}\mathbf{X}\mathbf{C}^T\#\mathbf{N} \tag{3.13}$$

e

$$\mathbf{X} = \mathbf{C}^T(\mathbf{Y}\#\mathbf{N})\mathbf{C}, \tag{3.14}$$

onde o símbolo "#" denota o produto termo a termo e $\mathbf{N}$ é a matriz de normalização definida como

$$\mathbf{N} = \sqrt{\mathbf{\Delta}} 1 \sqrt{\mathbf{\Delta}}. \tag{3.15}$$

Ou seja, a transfomada DCT

$$\mathbf{Y} = \mathbf{AXA}^T = \begin{pmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{pmatrix} X \begin{pmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{pmatrix}, \tag{3.16}$$

onde $a = 0,5$, $b = 0,6533$ e $c = 0,2706$ pode ser fatorada da seguinte forma

$$\mathbf{Y} = \mathbf{CXC}^T \# \mathbf{N} = \mathbf{CXC}^T \# \begin{pmatrix} a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \\ a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \end{pmatrix}, \tag{3.17}$$

onde $a = 0,5$, $b = \sqrt{2/5}$ e $c = 0,5$. Os valores de b e c são ligeiramente modificados para garantir a ortogonalidade da transformada.

Os coeficientes de $\mathbf{Y}$ são quantizados através da divisão termo a termo de seus elementos por uma matriz de quantização $\mathbf{H}$. Assim, temos a matriz de transformação quantizada $\mathbf{Y}^*$, expressa por

$$\mathbf{Y}^* = \mathbf{Y}(\#)\mathbf{H} = (\mathbf{CXC}^T)\#\mathbf{N}(\#)\mathbf{H}, \tag{3.18}$$

onde o símbolo $(\#)$ denota a operação de arredondamento para o inteiro mais próximo.

Podemos combinar a operação de normalização e quantização em uma matriz única $\mathbf{Q}$, de forma que

$$\mathbf{Y}^* = (\mathbf{CXC}^T)(\#)\mathbf{Q}, \tag{3.19}$$

onde $\mathbf{Y}^*$ representa a matriz dos coeficientes quantizados e $\mathbf{Q} = \mathbf{N}\#\mathbf{H}$.

A imagem reconstruída $\mathbf{X}^*$ pode ser obtida por

$$\mathbf{X}^* = \mathbf{C}^T(\mathbf{Y}^*\#\mathbf{Q}^*)\mathbf{C}, \tag{3.20}$$

onde $\mathbf{Q}^*$ e $\mathbf{Q}$ satisfazem

$$\mathbf{Q}^* \# \mathbf{Q} = \mathbf{\Delta 1 \Delta}. \tag{3.21}$$

O padrão H.264 [9] define a transformada inversa da seguinte forma

$$\mathbf{X}^* = \mathbf{C_i}^T (\mathbf{Y} \# \mathbf{N_i}) \mathbf{C_i}, \tag{3.22}$$

onde

$$\mathbf{C_i} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1/2 & -1/2 & -1 \\ 1 & -1 & -1 & 1 \\ 1/2 & -1 & 1 & -1/2 \end{pmatrix} \tag{3.23}$$

e

$$\mathbf{N_i} = \begin{pmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{pmatrix}, \tag{3.24}$$

com $a = 0,5$, $b = \sqrt{2/5}$ e $c = 0,5$.

Na transformada inversa, os coeficientes de $\mathbf{Y}$ são pré-multiplicados pela matriz de normalização $\mathbf{N_i}$. Assim, a multiplicação por fatores de 1/2 e -1/2 pode ser implementada com deslocamentos binários sem perda de precisão.

3.1.2 A transformada ICT simplificada

Uma versão simplificada da ICT [20] [21] proposta por M. Costa e K. Tong pode ser implementada de forma bastante rápida. Através da aproximação dos fatores combinados de normalização e quantização por potências de dois, pode-se diminuir o tempo computacional requerido para o cálculo da transformada. Em algoritmos convencionais, a etapa de quantização requer um número de divisões igual ao número de coeficientes de transformação. Ao aproximar os fatores por potências de dois, as divisões podem ser realizadas através de simples deslocamentos na representação binária dos coeficientes. Instruções de deslocamento binário dependem do número de bits deslocados, mas são mais velozes que operações de divisão e sua utilização permite a implementação de algoritmos ICT bastante rápidos.

A ICT simplificada aproxima a matriz de quantização $\mathbf{Q}$ por uma matriz $\mathbf{Q_a}$ formada exclusivamente por potências negativas de dois. Como regra de aproximação, os elementos de $\mathbf{Q_a}$ escolhidos são as potências de dois mais próximas, privilegiando os fatores menores em caso de proximidade idêntica. Ao substituir $\mathbf{Q}$ por $\mathbf{Q_a}$, a quantização é implementada através de deslocamentos binários seguido da adição de 0,5 e truncamento.

A aproximação introduz um erro no resultado obtido, mas pode ser compensado desde que a matriz de reconstrução $\mathbf{Q_a^*}$ satisfaça

$$\mathbf{Q_a^*} \# \mathbf{Q_a} = \mathbf{\Delta 1 \Delta}. \tag{3.25}$$

Esta simplificação pode reduzir o tempo de execução em mais de 50% quando comparado ao tempo requerido pela operação de divisões inteiras. O tempo exato depende também da arquitetura do processador utilizado, mas de forma geral, como instruções de divisão são implementadas através de deslocamentos repetidos e subtrações, a divisão através de deslocamentos utiliza apenas uma fração do tempo de execução requerido por divisões convencionais.

3.2 Desenvolvimento da transformada FLICT

A transformada FLICT - *Floating/Integer Cosine Transform* difere da ICT convencional na etapa de normalização/quantização, que é realizada com precisão em ponto flutuante. Nesta seção trataremos o desenvolvimento da FLICT no contexto do padrão de compressão de vídeo H.264 [9].

Na transformada direta, um total de 52 passos de quantização (Q_{step}) são definidos e indexados por um parâmetro QP (Tabela 3.1). O valor de Q_{step} dobra de tamanho a cada incremento de 6 em QP, sendo que o passo aumenta aproximadamente em 12,5% para cada incremento de 1 em QP. Esta variação nos passos de quantização possibilita um controle mais preciso e flexível do compromisso existente entre a taxa de bits e a qualidade.

Os valores de QP são diferentes para a luminância e as crominâncias. Ambos variam de 0 a 51, mas QP_{chroma} é obtido a partir de QP_Y de forma que os passos nas crominâncias sejam menores que os valores da luminância para QP_Y acima de 30. Opcionalmente, é possível definir de maneira diferenciada a relação entre QP_Y e QP_{chroma}, que deve ser sinalizada apropriadamente através do parâmetro *Picture*

Parameter Set.

QP	0	1	2	3	4	5	6	7	8	...	51
Q_{step}	0,625	0,6875	0,8125	0,875	1,00	1,125	1,25	1,375	1,625	...	224

Tabela 3.1: Passos de quantização do codificador H.264[9].

O cálculo da transformada direta é implementado [13] a partir da equação 3.17. A etapa de quantização é realizada em conjunto com a normalização segundo a equação 3.19 com

$$\mathbf{Q} = \mathbf{N}\#\mathbf{H} = \frac{1}{Q_{step}} \begin{pmatrix} a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \\ a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \end{pmatrix}. \tag{3.26}$$

A fim de se evitar instruções de divisão, o *software* de referência implementa a quantização/normalização utilizando uma matriz de fatores multiplicativos **MF**, onde

$$\mathbf{Q} = \mathbf{N}\#\mathbf{H} = \frac{1}{2^{qbits}}\mathbf{MF}, \tag{3.27}$$

ou seja,

$$\mathbf{MF} = \frac{2^{qbits}}{Q_{step}} \begin{pmatrix} a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \\ a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \end{pmatrix}, \tag{3.28}$$

com $qbits = 15 + \lfloor QP/6 \rfloor$

A matriz **MF** depende somente do índice QP desejado, e, portanto, varia de acordo com o passo de quantização. No entanto, para $QP > 5$, os elementos de **MF** permanecem inalterados pois o divisor $Qstep$ aumenta por um fator de 2 a cada incremento de 6 em QP, assim como 2^{qbits}.

Por exemplo, no intervalo $6 \leq QP \leq 11$, $qbits = 16$; para $12 \leq QP \leq 17$, $qbits = 17$. Assim, $QP = 6$ utiliza a mesma matriz **MF** de $QP = 0$ e $QP = 17$ utiliza a mesma de $QP = 5$, visto que o $6\%6 = 0$ e $17\%6 = 5$ (operador $\%6$ = resto da divisão por 6).

Dessa forma, a matriz **MF** é fixa e não necessita ser recalculada para todo QP. Dependendo da implementação, **MF** pode ser armazenada em memória não volátil.

QP	Posições (0,0),(2,0),(2,2),(0,2)	Posições (1,1),(1,3),(3,1),(3,3)	Outras
0	13107	5243	8066
1	11916	4660	7490
2	10082	4194	6554
3	9362	3647	5825
4	8192	3355	5243
5	7282	2893	4559

Tabela 3.2: Matriz de fatores multiplicativos **MF**[22].

A Tabela 3.2 é a implementada no *software* de referência do H.264. Note que alguns valores das duas últimas colunas são ligeiramente diferentes dos valores calculados pela equação 3.28. Esta modificação tem o objetivo de melhorar a qualidade subjetiva no decodificador, e não resulta em uma incompatibilidade, visto que somente a decodificação (transformação e quantização inversa) é padronizada.

O processo de decodificação visa recuperar a imagem original a partir dos coeficientes freqüenciais quantizados e codificados. Após o processo de decodificação de entropia, os coeficientes são recuperados a partir da equação

$$\mathbf{Y}' = \mathbf{Y}^* \# \mathbf{Q}^* = 64 Q_{step} \mathbf{Y}^* \# \mathbf{N_i}. \tag{3.29}$$

A equação acima é multiplicada por um fator constante igual a 64 para evitar erros de arredondamento.

Por fim, a imagem original pode ser reconstruída a partir de

$$\mathbf{X}' = \text{round} \left[\frac{1}{64} \mathbf{C_i}^T \mathbf{Y}' \mathbf{Ci} \right]. \tag{3.30}$$

A implementação de referência não especifica os valores de Q_{step} e $\mathbf{N_i}$ diretamente. Alternativamente, a matriz V é definida para $0 \leq QP \leq 5$, sendo que

$$\mathbf{V} = 64 Q_{step} \mathbf{N_i}. \tag{3.31}$$

Assim,

$$\mathbf{Y}^{'} = 2^{\lfloor QP/6 \rfloor}\mathbf{Y}^{*}\#\mathbf{V}. \tag{3.32}$$

QP	Posições (0,0),(2,0),(2,2),(0,2)	Posições (1,1),(1,3),(3,1),(3,3)	Outras
0	10	16	13
1	11	18	14
2	13	20	16
3	14	23	18
4	16	25	20
5	18	29	23

Tabela 3.3: Matriz de fatores multiplicativos $\mathbf{V}$[13].

Do mesmo modo que a matriz de fatores multiplicativos **MF**, a matriz $\mathbf{V}$ do decodificador também é fixa (Tabela 3.3) e o termo $2^{\lfloor QP/6 \rfloor}$ na equação 3.32 multiplica a saída por um fator de 2 a cada incremento de 6 em QP.

O processo de codificação pode ser melhorado se calcularmos a transformada com um pouco mais de precisão na etapa de quantização/normalização. A matriz de quantização inversa $\mathbf{Q}^{*}$ do decodificador pode ser calculada da seguinte forma

$$\mathbf{Q}^{*} = \frac{2^{\lfloor QP/6 \rfloor}}{64}\mathbf{V}\#\mathbf{R}, \tag{3.33}$$

onde

$$\mathbf{R} = \begin{pmatrix} 1 & 1/2 & 1 & 1/2 \\ 1/2 & 1/4 & 1/2 & 1/4 \\ 1 & 1/2 & 1 & 1/2 \\ 1/2 & 1/4 & 1/2 & 1/4 \end{pmatrix}. \tag{3.34}$$

Definindo

$$\mathbf{V_{FLICT}} = \frac{1}{64}\mathbf{V}\#\mathbf{R}, \tag{3.35}$$

temos

$$\mathbf{Q}^{*} = 2^{\lfloor QP/6 \rfloor}\mathbf{V_{FLICT}}. \tag{3.36}$$

A matriz multiplicativa $\mathbf{R}$ e o fator divisivo 64 são necessários para reescalonar o resultado, visto que a equação 3.20 utiliza $\mathbf{C}$ e não $\mathbf{C_i}$ e que o fator de 64 foi introduzido para o cálculo de $\mathbf{V}$.

É importante notar que se a equação 3.21 for satisfeita, então podemos calcular uma nova matriz de quantização $\mathbf{Q}$ do codificador que utilizado na transformada direta, permitirá obter os coeficientes freqüenciais com mais precisão do que a transformada ICT estabelecida no padrão H.264.

Para a matriz ICT 4x4 da equação 3.7 temos

$$\mathbf{D} = \mathbf{C}\mathbf{C}^T = \begin{pmatrix} 4 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 10 \end{pmatrix}, \tag{3.37}$$

e a inversa de $\mathbf{D}$

$$\mathbf{D}^{-1} = \mathbf{\Delta} = \begin{pmatrix} 1/4 & 0 & 0 & 0 \\ 0 & 1/10 & 0 & 0 \\ 0 & 0 & 1/4 & 0 \\ 0 & 0 & 0 & 1/10 \end{pmatrix}. \tag{3.38}$$

O produto $\mathbf{\Delta 1 \Delta}$ vale

$$\begin{aligned} \mathbf{\Delta 1 \Delta} &= \begin{pmatrix} 1/4 & 0 & 0 & 0 \\ 0 & 1/10 & 0 & 0 \\ 0 & 0 & 1/4 & 0 \\ 0 & 0 & 0 & 1/10 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1/4 & 0 & 0 & 0 \\ 0 & 1/10 & 0 & 0 \\ 0 & 0 & 1/4 & 0 \\ 0 & 0 & 0 & 1/10 \end{pmatrix} \\ &= \begin{pmatrix} 0,0625 & 0,025 & 0,0625 & 0,025 \\ 0,025 & 0,01 & 0,025 & 0,01 \\ 0,0625 & 0,025 & 0,0625 & 0,025 \\ 0,025 & 0,01 & 0,025 & 0,01 \end{pmatrix}. \end{aligned} \tag{3.39}$$

Como $\mathbf{Q}$ deve satisfazer $\mathbf{Q}^* \# \mathbf{Q} = \mathbf{\Delta 1 \Delta}$,

$$\mathbf{Q} = \frac{1}{2^{\lfloor QP/6 \rfloor}} \mathbf{\Delta 1 \Delta} \# \mathbf{W_{FLICT}}, \tag{3.40}$$

onde $\mathbf{W_{FLICT}}$ é a matriz composta pelas inversas dos elementos de $\mathbf{V_{FLICT}}$.

$$\mathbf{W_{FLICT}}(i,j) = \mathbf{V^{-1}_{FLICT}}(i,j) \text{ para i,j=0,1,2,3.} \tag{3.41}$$

Podemos definir o produto $\mathbf{\Delta 1 \Delta \# W_{FLICT}}$ como

$$\mathbf{MF_{FLICT}} = \mathbf{\Delta 1 \Delta \# W_{FLICT}}, \tag{3.42}$$

e a matriz $\mathbf{Q}$ final como

$$\mathbf{Q} = \frac{1}{2^{\lfloor QP/6 \rfloor}} \mathbf{MF_{FLICT}}. \tag{3.43}$$

Desse modo, os coeficientes quantizados da transfomada FLICT direta são obtidos através da equação 3.19 ($\mathbf{Y} = \mathbf{CXC}^T(\#)\mathbf{Q}$), com $\mathbf{Q}$ calculado a partir da equação 3.43.

QP	Posições (0,0),(2,0),(2,2),(0,2)	Posições (1,1),(1,3),(3,1),(3,3)	Outras
0	0,4	0,16	0,2462
1	0,3636	0,1422	0,2286
2	0,3077	0,1280	0,2000
3	0,2857	0,1113	0,1778
4	0,2500	0,1024	0,1600
5	0,2222	0,0883	0,1391

Tabela 3.4: Matriz de fatores multiplicativos $\mathbf{MF_{FLICT}}$.

Assim como $\mathbf{MF}$, a matriz $\mathbf{MF_{FLICT}}$ é fixa e não necessita ser recalculada para todo QP (Tabela 3.4).

A transformada FLICT proposta apresenta-se como uma alternativa para a implementação da transformada ICT no codificador H.264. A utilização de cálculo em ponto flutuante na etapa de quantização/normalização permite que os coeficientes de freqüência quantizados sejam obtidos com mais precisão.

É importante notar também que o codificador H.264 que implementa a FLICT permanece compatível com o padrão e assim como a ICT padronizada, não resulta em descasamento ou *mismatch* entre o codificador e o decodificador, que é uma característica de sistemas de codificação que utilizam a DCT como transformada.

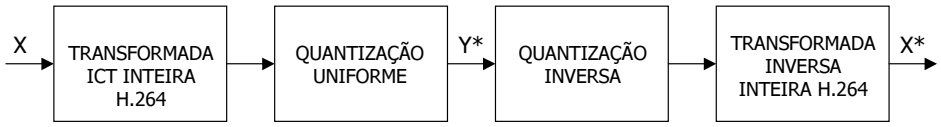
4
IMPLEMENTAÇÃO DA TRANSFORMADA FLICT

A transformada FLICT desenvolvida no capítulo anterior foi implementada no codificador de vídeo H.264 do software de referência disponibilizado pelo grupo JVT. A implementação consiste de um codificador que comprime seqüências de vídeo no formato YUV com amostragem do tipo 4:2:0 (4 amostras de luminância para 2 de crominância) e de um decodificador que é capaz de reconstruir uma seqüência de vídeo comprimida de acordo com o padrão H.264, resultando numa seqüência de vídeo no formato YUV.

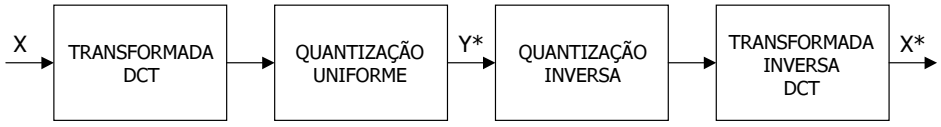
O objetivo da utilização da transformada FLICT no codificador H.264 é de melhorar a qualidade do vídeo comprimido através de um cálculo mais preciso dos coeficientes de freqüência quantizados. Embora a implementação do software de referência utilize a transformada ICT com aritmética inteira no codificador, o uso da transformada FLICT com aritmética em ponto flutuante (na etapa de quantização/normalização) não torna o codificador imcompatível com o padrão, visto que somente a decodificação é definida no H.264.

4.1 Comparação entre ICT e FLICT

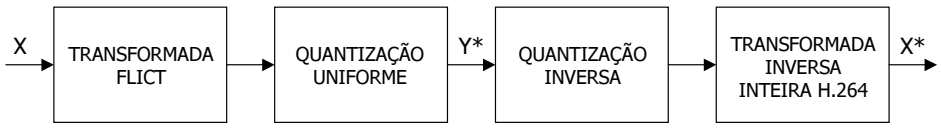
Com o objetivo de avaliar o desempenho da FLICT em um sistema de codificação, algumas imagens foram comprimidas e reconstruídas utilizando as transformadas DCT, ICT e FLICT. Embora o objetivo seja comparar a desempenho da FLICT com relação a ICT do padrão H.264, é interessante incluir a DCT neste primeiro ensaio porque esta transformada é o limite superior em termos de desempenho para ambas as transformadas. O diagrama da Figura 4.1 ilustra o ensaio realizado. As imagens de entrada foram particionadas em blocos de tamanho 4x4 (X), transformadas e quantizadas, resultando nos coeficientes de freqüência (Y^*). A quantização utilizada é uniforme e os passos de quantização são os mesmos especificadas pelo padrão (Tabela 3.1) para todos os esquemas em (a), (b) e (c), com o índice QP variando de 0 a 51.



(a) Compressão e reconstrução utilizando ICT (padrão H.264)



(b) Compressão e reconstrução utilizando DCT



(c) Compressão e reconstrução utilizando FLICT

Figura 4.1: Ensaio comparativo entre DCT, ICT e FLICT.

Na seqüência, os coeficientes quantizados Y^* passam pela quantização inversa e pelas respectivas transformadas inversas, originando os blocos de imagem reconstruídos (X^*).

A imagem reconstruída contém erros devido à quantização e o desempenho de cada esquema foi avaliado calculando a razão sinal-ruído de pico (PSNR) para os diferentes passos de quantização.

A seguir, temos o gráfico de desempenho comparativo para a imagem "Lena"-256x256 (Figura 4.2). Podemos observar que a FLICT tem um desempenho comparável à DCT, porém não é melhor que a mesma.

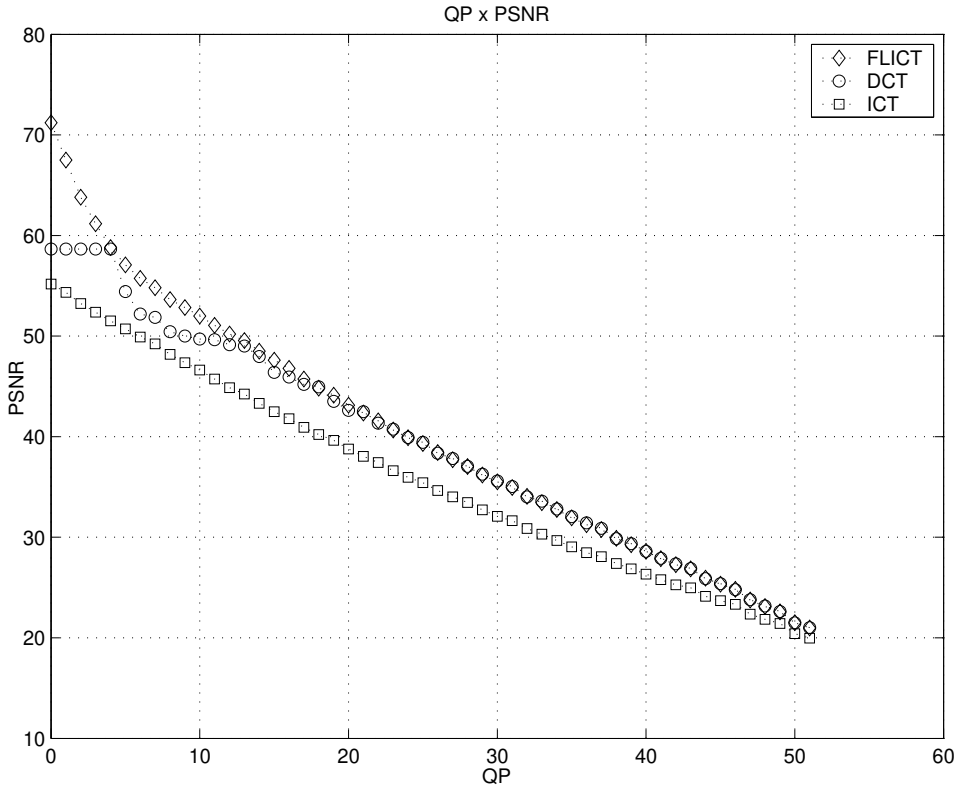


Figura 4.2: Curva QP x PSNR para imagem Lena: DCT, ICT e FLICT.

Em alguns pontos, o desempenho da FLICT é superior, porém é conveniente lembrar que no desenvolvimento da matriz MF no capítulo anterior, alguns coeficientes do software de referência foram alterados pelo grupo de padronização de forma a diminuir o efeito da quantização e melhorar a qualidade subjetiva das imagens reconstruídas pelo decodificador. Como o ensaio com a DCT considera os mesmos passos de quantização, a FLICT apresenta desempenho superior em alguns casos. A transformada FLICT e a ICT são derivações da transformada DCT e, portanto,

o desempenho em relação a ela será inferior. No entanto, é sempre desejável implementar sistemas de codificação de imagens e vídeo de maneira simplificada. Neste trabalho, o objetivo é implementar a FLICT de forma mais simplificada que a DCT, mas obtendo a mesma eficiência (ou muito próxima dela) em termos de compactação de energia.

Com relação aos passos de quantização, a transformada FLICT tem um desempenho melhor que a transformada ICT implementada pelo padrão H.264. As curvas de desempenho são praticamente lineares com o passo de quantização e a FLICT apresenta para esta imagem um ganho de aproximadamente 4,5 dB em média, para os diferentes passo de quantização. Quando o passo de quantização é pequeno, entre 0 a 5, o ganho da FLICT em relação à ICT é de pouco mais de 15 dB para o menor passo e de aproximadamente 8 dB para QP=5.

As Figuras 4.3 e 4.4 ilustram duas imagens que foram comprimidas e recuperadas para o passo de quantização QP=32, a primeira com transformada ICT e a segunda com transformada FLICT.

Figura 4.3: Imagem Lena recuperada - ICT, QP=32.

Figura 4.4: Imagem Lena recuperada - FLICT, QP=32.

Para a imagem “Cameraman”-256x256, o desempenho da FLICT em relação à ICT também é bastante semelhante, como podemos observar na Figura 4.5. Da mesma forma que a imagem “Lena”, a transformada FLICT apresenta um desempenho melhor, sendo praticamente linear com QP e com ganhos mais significativos para passos de quantização menores ($0 \leq QP \leq 5$). A FLICT mantém um ganho de aproximadamente 4 dB em relação à ICT para os diversos passos de quantização.

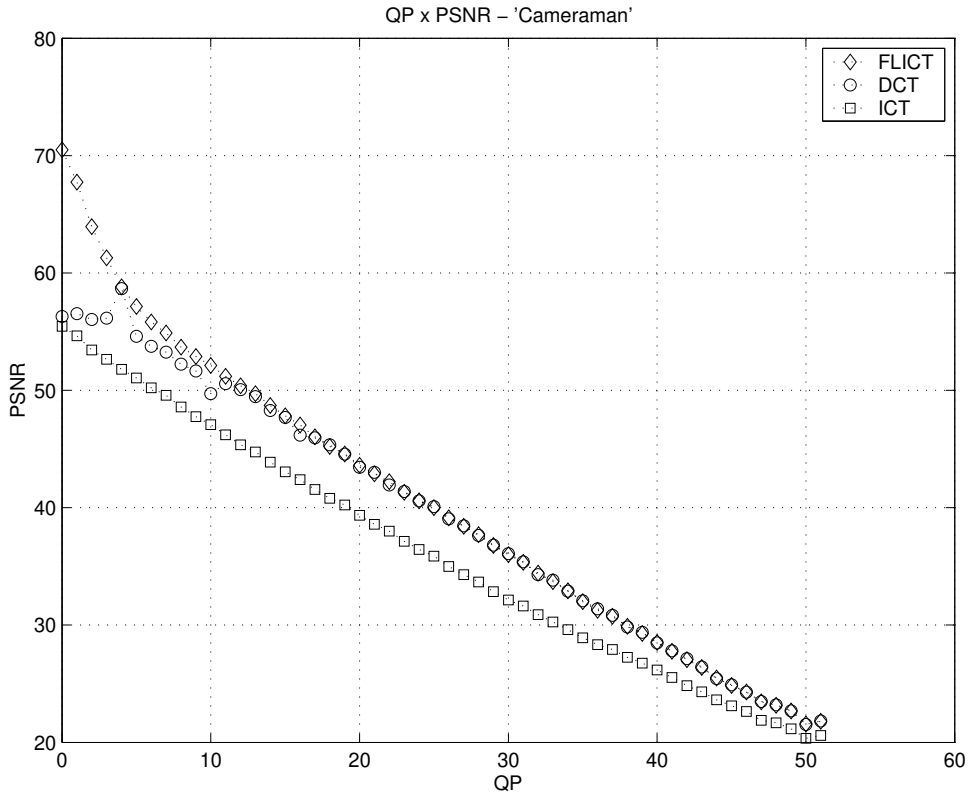


Figura 4.5: Curva QP x PSNR para imagem Cameraman: DCT, ICT e FLICT.

As Figuras 4.6 e 4.7 ilustram duas imagens que foram comprimidas e recuperadas para o passo de quantização QP=32, a primeira com transformada ICT e a segunda com transformada FLICT.

Figura 4.6: Imagem Cameraman recuperada - ICT, QP=32.

Figura 4.7: Imagem Cameraman recuperada - FLICT, QP=32.

Em termos de erro médio, podemos observar pela Figura 4.8 que praticamente não existe erro de reconstrução até o passo de quantização referente a QP=10 para ambas transformadas. A partir disso, o erro é crescente, alcançando o valor máximo de aproximadamente MSE=33,0 para ICT, no último passo de quantização. Para a FLICT, o máximo valor do erro médio é de aproximadamente MSE=13,5. Podemos observar que o erro de reconstrução da transformada FLICT é bem menor do que a ICT, sendo que para $QP > 15$ a diferença aumenta consideravelmente.

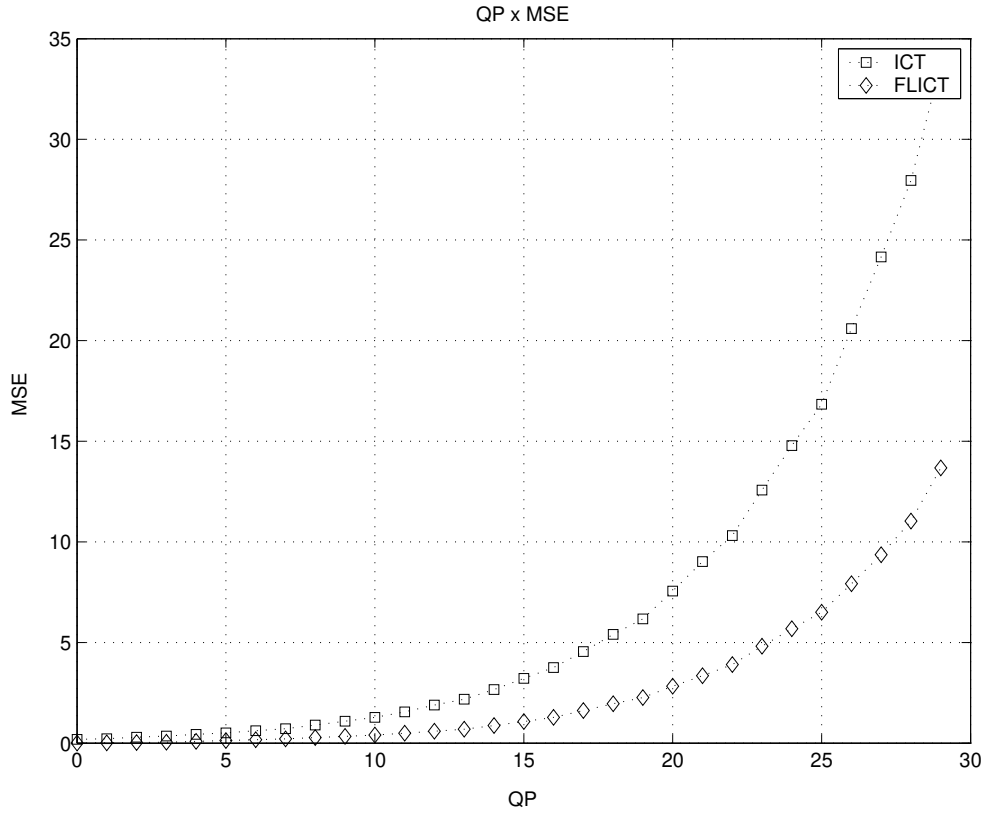


Figura 4.8: Curva QP x MSE para imagem Cameraman: ICT e FLICT.

4.2 A transformada FLICT no codificador H.264

Neste ensaio, a transformada ICT direta implementada no H.264 foi substituída pela transformada FLICT desenvolvida no capítulo anterior. A Figura 4.9 ilustra o diagrama de blocos da implementação.

Com este sistema, as seqüências de imagens de formato 4:2:0 foram comprimidas e avaliadas. As seqüências foram comprimidas utilizando um GOP (*Group of Pictures*) típico com IBPBPB. Cada quadro I, P e B foi quantizado variando os passos de quantização de 0 a 51 e a razão sinal-ruído de pico foi calculada tendo como referência o quadro original para o cálculo do erro. A etapa de codificação de entropia é realizada com o código CAVLC.

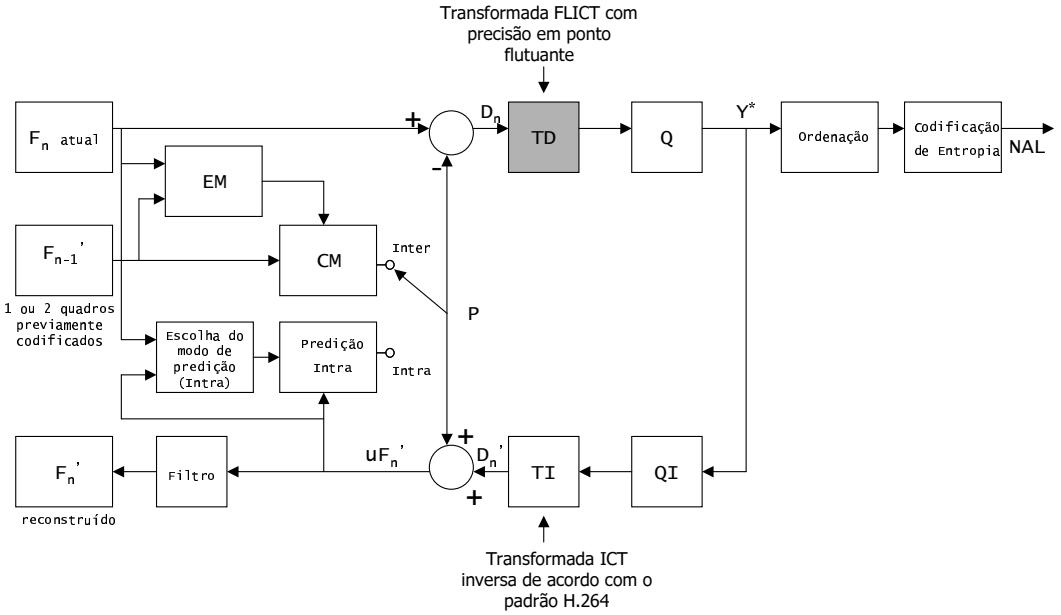


Figura 4.9: Codificador H.264 com transformada FLICT.

As curvas de desempenho foram calculadas tanto para a luminância (Y) como para as duas componentes de crominância (U e V). Podemos observar pela Figura 4.10 que a seqüência comprimida com o transformada FLICT mantém uma melhora em relação à seqüência comprimida com a implementação original para a componente de luminância. A média do ganho é de aproximadamente 1,80 dB para os diferentes passos de quantização.

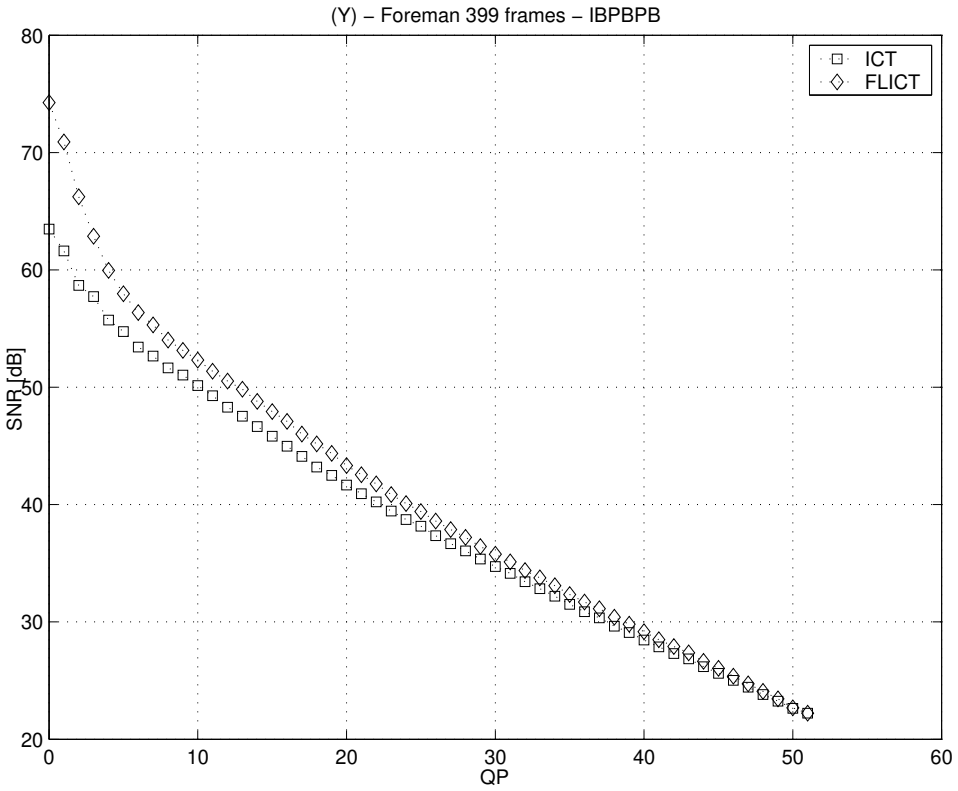


Figura 4.10: Curva QP x PSNR para a seqüência Foreman - (Y).

Para as componentes de crominância U e V, o desempenho da implementação proposta também apresenta melhora, como se pode observar nas Figuras 4.11 e 4.12. A média do ganho é de 1,75 dB para a componente U e de 1,70 dB para a componente V.

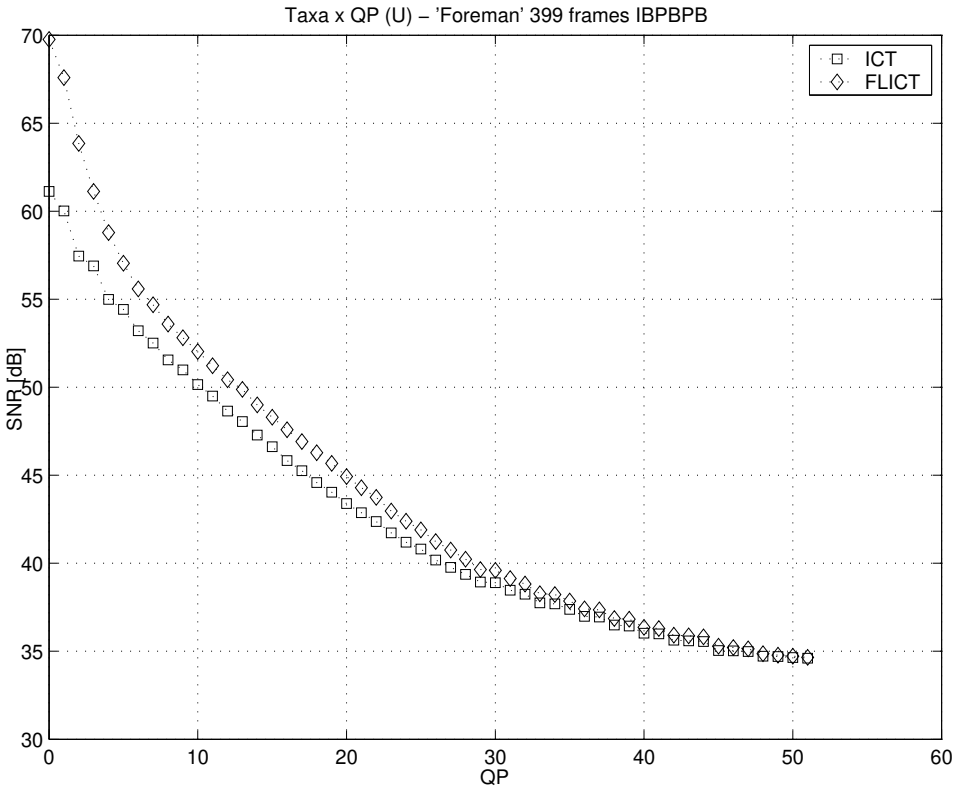


Figura 4.11: Curva QP x PSNR para a seqüência Foreman - (U).

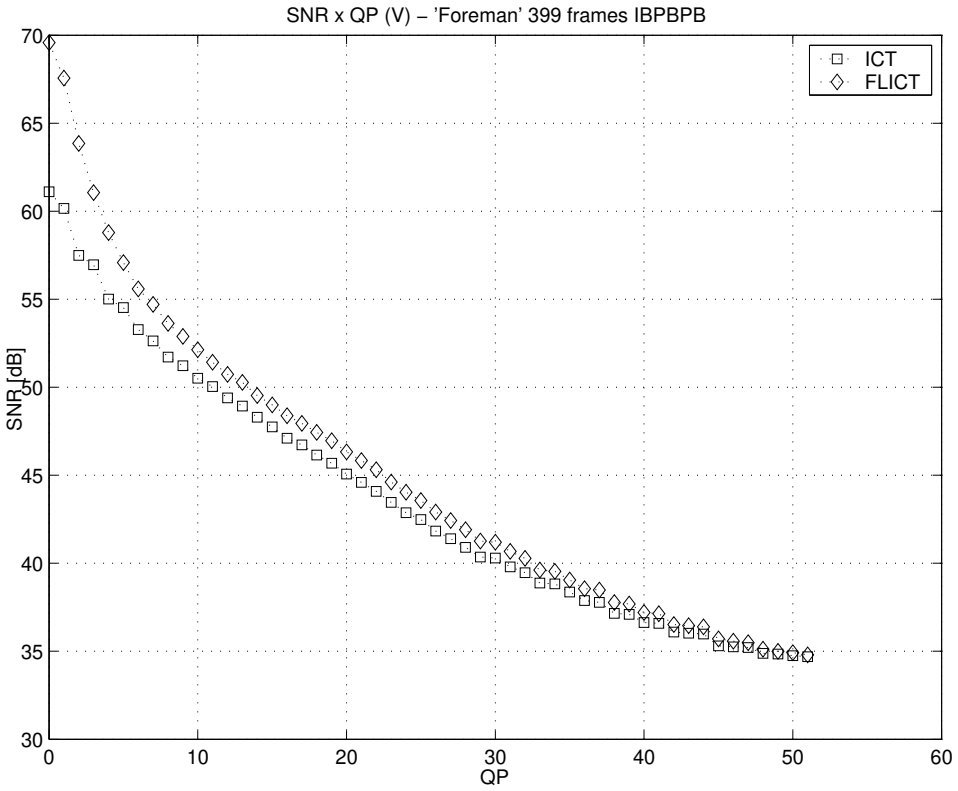


Figura 4.12: Curva QP x PSNR para a seqüência Foreman - (V).

As Figuras 4.13 e 4.14 ilustram dois quadros da seqüência 'Foreman' que foram comprimidos e recuperados para o passo de quantização QP=35 e GOP igual a IBPBPB, a primeira com transformada ICT e a segunda com transformada FLICT.

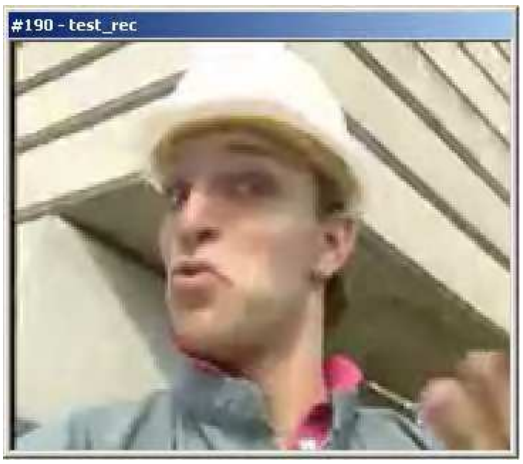


Figura 4.13: Quadro #190 recuperado da seqüência Foreman - ICT, QP=35.

Figura 4.14: Quadro #190 recuperado da seqüência Foreman - FLICT, QP=35.

O tempo médio requerido para codificação da seqüência é igual a 0,93 seg/quadro na ICT e 1,15 seg/quadro para a FLICT. O valor referente à FLICT reflete o tempo requerido para a realização da quantização/normalização em ponto flutuante. Estes valores, tanto para a ICT quanto para a FLICT, são bastante elevados, o que im-

possibilita qualquer operação do codificador em tempo real. O codificador foi implementado totalmente em linguagem C e há vários pontos nos algoritmos passíveis de serem otimizados. Por exemplo, os métodos de predição e de estimação de movimento foram implementados através de buscas e comparações exaustivas, o que contribui para a maior parcela do alto tempo de codificação apresentado.

Em [25] os autores apresentam uma implementação do decodificador H.264 em um processador de propósito geral. Verifica-se que a maior parte do tempo de processamento se concentra na compensação de movimento (55%) e na transformada inversa e quantização (12%). A implementação referida do decodificador consegue ser três vezes mais veloz que o *software* de referência, sendo capaz de decodificar até 48 quadros/seg. Embora a decodificação seja mais simples que a codificação, este artigo ilustra a dificuldade atual em se desenvolver produtos com o padrão H.264, visto que a implementação só foi possível através do uso de instruções com extensões multimídia e dedicadas, além de utilizar um processador com freqüência de 2,4 GHz, que na data atual é uns dos mais velozes disponíveis comercialmente.

É interessante também avaliar o desempenho do codificador em relação à taxa média de bits necessária para codificar a seqüência. Nos gráficos das Figuras 4.15, 4.16 e 4.17, podemos observar que o desempenho do codificador com a transformada FLICT é inferior ao codificador implementado como referência.

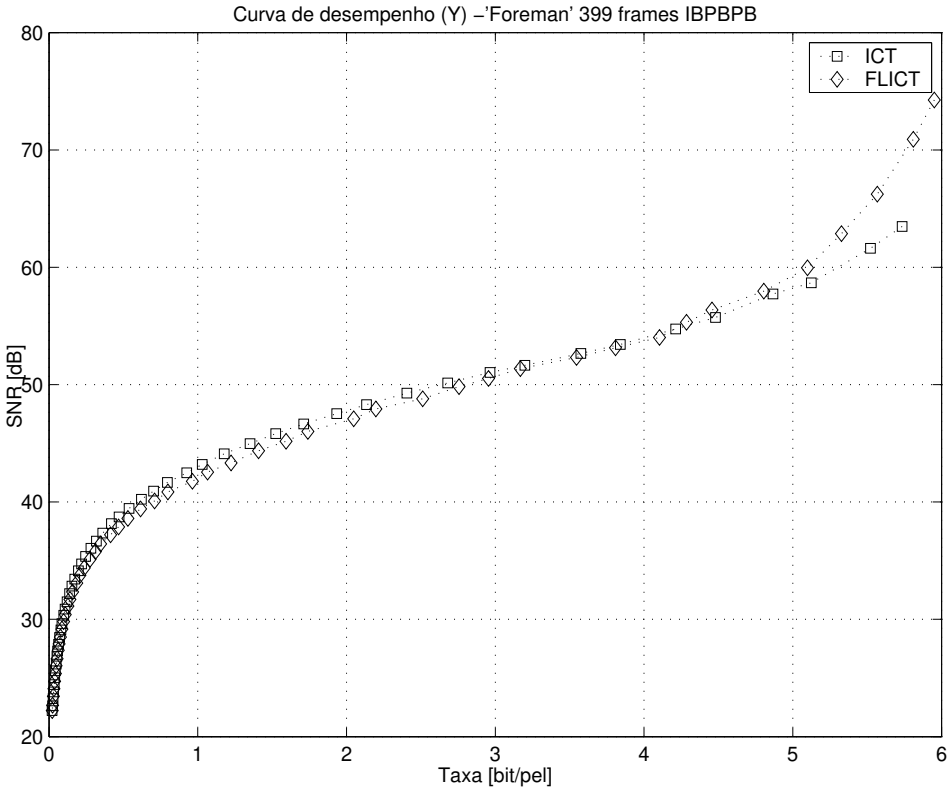


Figura 4.15: Curva BPP x PSNR para a seqüência Foreman - (Y).

Para um dado valor de PSNR, o codificador com transformada FLICT necessita de mais bits do que o codificador que utiliza a ICT padrão. Em relação à componente de luminância, o desempenho do sistema proposto é superior apenas nos primeiros passos de quantização, para $QP \leq 7$, onde a taxa média é superior a 4,4 bits/*pixel*. Para a componente de crominância U, o desempenho da FLICT é superior apenas para $QP \leq 6$, onde a taxa média é superior a 4,25 bits/*pixel*. Para a componente de crominância V, o comportamento é o mesmo, sendo superior apenas para $QP \leq 6$, onde a taxa média é superior a 4,25 bits/*pixel*.

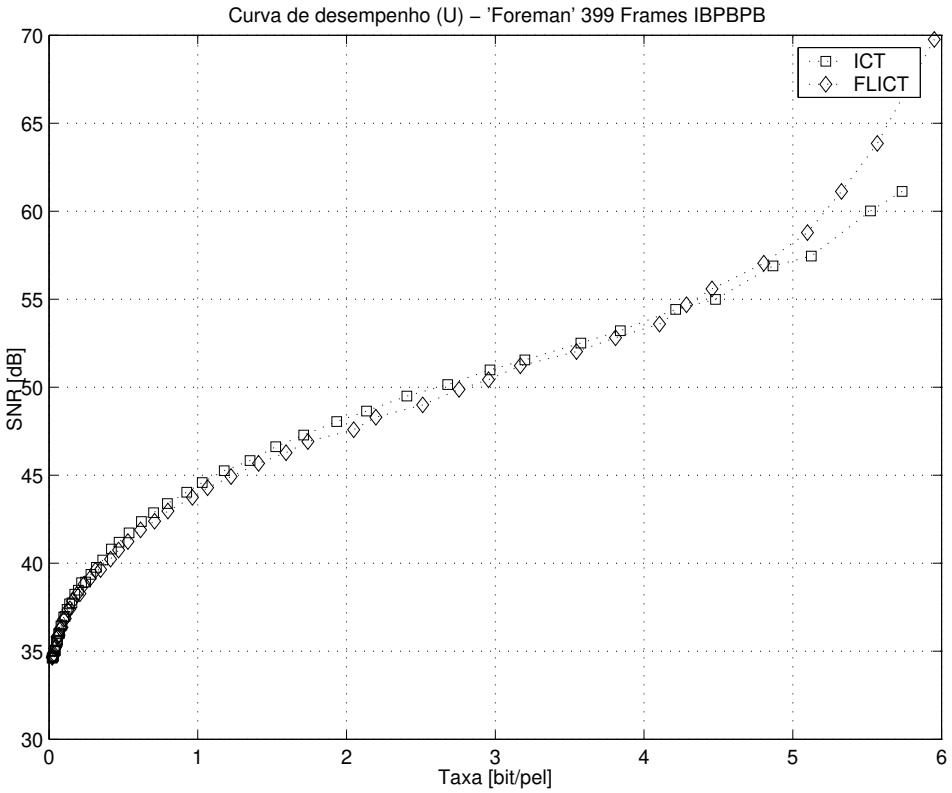


Figura 4.16: Curva BPP x PSNR para a seqüência Foreman - (U).

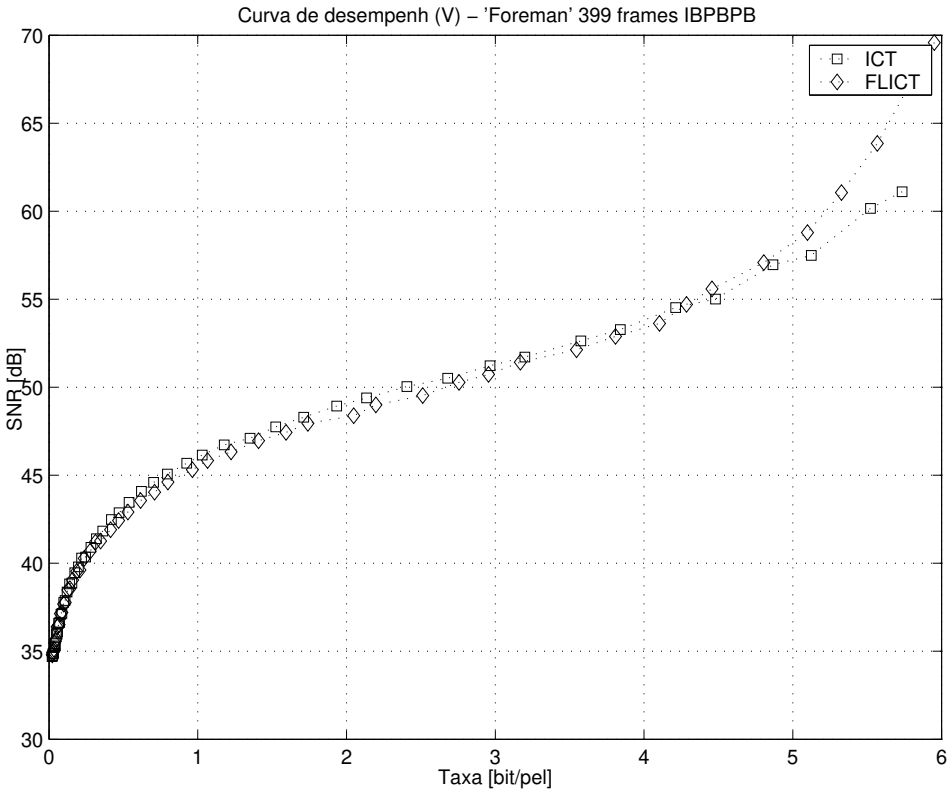


Figura 4.17: Curva BPP x PSNR para a seqüência Foreman - (V).

Embora o codificador com transformada FLICT proposto tenha desempenho inferior ao sistema original em relação à taxa média de bits, o seu desempenho ainda é comparável, visto que as duas curvas são muito próximas.

Outra consideração importante para uma avaliação mais imparcial é de que todas as tabelas de códigos de comprimento variável utilizados neste ensaio são ajustados às estatísticas dos coeficientes quantizados que resultam da transformada ICT original.

Como estamos utilizando uma transformada diferente, as estatísticas desta nova transformada não são iguais às da original e portanto, uma avaliação mais adequada deveria incluir um código que fosse adaptado às estatísticas dos coeficientes calculados com a transformada FLICT. Embora a utilização de um código mais adequado seja desejável, isto implica em alterações nas tabelas de código utilizadas no padrão, resultando em incompatibilidade com as especificações. Entretanto, isto não impede que a utilização da FLICT juntamente com seu código associado possa ser considerada em futuras versões ou evoluções do padrão.

Analisando os valores dos ganhos médios apresentados com o uso da transformada FLICT, tanto para imagens quanto para seqüências de imagens, observa-se que o ganho médio obtido para a FLICT é bastante significativo, o que encoraja a utilização desta transformada juntamente com um código apropriado. A implementação da FLICT, embora imponha o uso de cálculo em ponto flutuante, necessita de apenas 16 multiplicações por bloco (para blocos de tamanho 4x4), sendo que o desempenho alcançado é bastante próximo da DCT.

5
CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho analisamos o padrão de codificação de vídeo H.264 (MPEG-4 Part 10). Uma implementação do codificador foi proposta com o objetivo de melhorar a qualidade subjetiva das seqüências de vídeo reconstruídas. A utilização da transformada FLICT no codificador H.264 melhora a relação sinal-ruído das imagens reconstruídas, ao mesmo tempo que evita a ocorrência de *mismatch* ou descasamento entre o codificador e o decodificador.

A utilização da transformada FLICT aumenta o número de coeficientes quantizados de valor significativo (maiores que zero), o que resulta num aumento da taxa de bits necessária para a representação do vídeo comprimido.

O ganho obtido com o uso da FLICT em relação à transformada original é significativo para os diversos passos de quantização do padrão. Entretanto, o código utilizado pelo padrão não se adapta bem às características da transformada proposta. Um código mais adaptado seria mais conveniente, mas implicaria em uma não compatibilidade com a recomendação do H.264. Apesar deste inconveniente, a utilização da transformada FLICT na compressão de imagens e seqüências apre-

sentou margem para ganhos significativos em relação ao sistema implementado pelo grupo JVT, o que encoraja a busca de um código mais apropriado e que seja mais eficiente na compressão dos coeficientes transformados.

O padrão de compressão MPEG-4 AVC/H.264 representa um grande avanço em relação aos padrões precedentes, como o MPEG-2 e MPEG-4 *Simple Profile*. Embora o grau de complexidade também aumente, a redução na taxa de bits é de pelo menos 33% (em relação ao MPEG-4), chegando a 50% (em relação ao MPEG-2) [11].

A continuidade deste trabalho poderá abordar a busca de um código mais adaptado à transformada FLICT, assim como implementações do codificador H.264 em tempo real (*hardware/software*) e para o transporte de vídeo H.264 na Internet.

Um codificador que utiliza H.264 demanda um grande poder de processamento, mesmo para os processadores de propósito geral disponíveis atualmente no mercado. Atualmente, a implementação do codificador em *software* e em tempo real é bastante difícil, sendo necessárias otimizações específicas e dedicadas aos recursos de cada processador. Embora isto seja um inconveniente, espera-se que a tecnologia de processadores também evolua de maneira rápida, se tomarmos como base os avanços alcançados nesta área no passado recente. A crescente demanda por processadores mais rápidos indica que num futuro próximo, processadores de propósito geral poderão codificar e decodificar *bitstreams* H.264 a velocidades aceitáveis para as diversas aplicações.

Espera-se em breve que novos produtos multimídia utilizem o codificador H.264. A redução da taxa de bits necessária para a transmissão de vídeo permite o surgimento de serviços de vídeo digital de maior qualidade que utilizam a Internet. Ao mesmo tempo que a tecnologia de compressão de vídeo apresenta avanços ao longo do tempo, aumenta-se também a capacidade das diversas tecnologias de acesso à rede Internet, como a tecnologia DSL (*Digital Subscriber Line*) e o acesso através da rede de TV a cabo. O mesmo avanço é observado em relação às redes sem-fio [26] , onde tecnologias de terceira geração como o CDMA/EvDO (*Code Division Multiple Access/Evolution Data Only*) [27] podem alcançar taxas de até 2,4 Mbits/s no acesso.

À medida que a rede Internet evolui para uma rede com qualidade de serviço (QoS) [28], através da utilização de tecnologias como o MPLS (*Multi Protocol Label Switching*) [29] [30], DiffServ (*Differentiated Services*) [28] e IntServ (*Integrated Services*) [28], espera-se que serviços como vídeo sob demanda e videotelefonia se

tornem cada vez mais comuns. De fato, o tráfego de vídeo na rede Internet tende a aumentar consideravelmente, e no futuro espera-se que a maior parte do tráfego na Internet seja gerada por serviços de vídeo.

Um outro tema de interesse emergente é o estudo sobre a utilização do codificador H.264 no SBTVD (Sistema Brasileiro de Televisão Digital) [31]. A utilização do codificador H.264 também pode ser uma alternativa interessante no sistema brasileiro. Os sistemas atuais como ATSC (*Advanced Television Systems Committee*, americano) [32] DVB-T (*Digital Video Broadcasting - Terrestrial*, europeu) [33], ISDB-T (*Integrated Services Digital Broadcasting - Terrestrial*, japonês) [34] utilizam codificadores baseados em MPEG-2. A utilização do H.264 em sistemas de TV digital permitiria uma grande redução de custos, uma vez que aumentaria a eficiência na utilização do espectro de freqüência (uma redução de 50% em relação ao MPEG-2 significaria reduzir pela metade o número de faixas necessárias, com a mesma qualidade de imagem).

Nesta data, à medida que se desenvolvem estudos técnicos sobre a viabilidade e a implantação do SBTVD encomendado pelo Ministério das Comunicações às Universidades brasileiras, a utilização do codificador H.264 no sistema brasileiro é uma opção a ser considerada. Embora a maior parte do conteúdo esteja em formato MPEG-2 (devido à predominância deste padrão nos sistemas atuais), a redução de custos associados à utilização do H.264 é um fator que contribui favoravelmente para sua adoção. Ao definir um sistema de codificação de vídeo para o SBTVD, é importante que a tecnologia escolhida não se torne obsoleta em um curto espaço de tempo, e evite a necessidade de uma atualização prematura do sistema.

Além do ponto de vista técnico, o SBTVD a ser desenvolvido também deve considerar as questões comerciais, sociais e estratégicas, o que torna o assunto bastante complexo. Incentivo às exportações, desenvolvimento da indústria nacional de eletrônica de consumo e semicondutores, democratização do acesso à Internet e contrapartidas comerciais são alguns dos pontos a considerar em relação ao estabelecimento do SBTVD.

Referências Bibliográficas

[1] Mpeg, "Website oficial." http://www.mpeg.org.

[2] N. S. Jayant e P. Noll, *Digital Coding of Waveforms*. Prentice Hall, 1984.

[3] R. C. Gonzales e R. E. Woods, *Digital Image Processing*. Addison-Wesley Publishing Company, Setembro 1993.

[4] N. Ahmed, T. Natarajan e K. R. Rao, "Discrete Cosine Transform," *IEEE Transactions on Computers*, vol. C.23, pp. 90–93, Janeiro 1974.

[5] K. R. Rao e P. Yip, *Discrete Cosine Transform: Algorithms, Advantages and Applications*. Academic Press, Inc, 1990.

[6] W.-K. Cham, "Development of integer cosine transforms by the principle of dyadic symmetry," *IEE Proceedings, Part 1*, vol. 136, pp. 276–282, Agosto 1989.

[7] W. Pennebacker e J. Mitchell, *JPEG Still Image Compression Standard*. Van Nostrand Reinhold, 1993.

[8] J. Watkinson, *MPEG-2*. Focal Press, Maio 1999.

[9] ITU-T Rec. H.264/ISO/IEC 11496-10, "Advanced Video Coding," *Final Committee Draft, Documento JVT-F100*, Dezembro 2002.

[10] I. Richardson, "H.264 tutorials." http://www.vcodex.com, Março 2003.

[11] R. Schäfer, T. Wiegand and H. Schwarz, "The emerging H.264/AVC standard," *European Broadcasting Union Technical Review*, Janeiro 2003.

[12] H. Kalva, *Delivering MPEG-4 Based Audio-Visual Services*. Kluwer Academic Publishers, 2001.

[13] Joint Video Team, "H.264 Reference Software." ftp://ftp.imtc-files.org/jvt-experts/reference_software, Dezembro 2002.

[14] G.Bjontegaard e K. Lillevold, "Context-adaptive VLC coding of coefficients," *JVT Document JVT-C028*, Maio 2002.

[15] D. Marpe, G. Blattermann e T. Wiegand, "Adaptive codes for H.26L," *JVT Document JVT-L13*, Janeiro 2001.

[16] S. W. Golomb, "Run-length encodings," *IEEE Trans. Inf. Theory*, vol. IT-12, pp. 399–401, Julho 1966.

[17] P. G. Howard e J. S. Vitter, "Analysis of arithmetic coding for data compression," *Information Processing and Management*, vol. 28, no. 6, pp. 749–764, 1992.

[18] L. Ekroot, S. Dolinar e K.-M. Cheung, "Integer cosine transform compression for galileo at jupiter: A preliminary look," *TDA Progress Report 42-115, Communications Systems and Research Section, Jet Propulsion Laboratory, NASA, California*, vol. Nov 1993, pp. 110–123, Novembro 1993.

[19] K.-M. Cheung, M. Belongie e K. Tong, "End-to-end system consideration of the galileo image compression system," *TDA Progress Report 42-126, Communications Systems and Research Section, Jet Propulsion Laboratory, NASA, California*, vol. Ago 1996, pp. 1–11, Agosto 1996.

[20] M. H. M. Costa e K. Tong, "A Simplified Integer Cosine Transform and Its Application in Image Compression," *TDA Progress Report 42-119, Communications Systems and Research Section, Jet Propulsion Laboratory, NASA, California*, vol. Nov 1994, pp. 129–139, Novembro 1994.

[21] K. Teraoka e M. H. M. Costa, "Compressão de vídeo usando a transformada inteira do co-seno," *VII Congresso de Iniciação Científica, Campinas, SP*, Setembro 1999.

[22] K.-M. Cheung, F. Pollara e M. Shahshahani, "Integer cosine transform for image compression," *TDA Progress Report 42-105, Communications Systems and Research Section, Jet Propulsion Laboratory, NASA, California*, vol. Nov 1991, pp. 45–53, Maio 1991.

[23] H. Malvar, A. Hallapuro, M. Karczewicz e L. Kerofsky, "Low-complexity transform and quantization with 16-bit arithmetic for H.26L," *IEEE International Conference on Image Processing*, Setembro 2002.

[24] A. Hallapuro, M. Karczewicz e H. Malvar, "Low-complexity transform and quantization - Part 1:Basic Implementation," *Documento JVT-B038*, Fevereiro 2001.

[25] Xiaosong Zhou, Eric Q. Li e Yen-Kuang Chen, "Implementation of H.264 Decoder on General-Purpose Processors with Media Instructions," *SPIE Conference on Image and Video Communications and Processing*, Janeiro 2003.

[26] T. Stockhammer, M. M. Hannuksela e T. Wiegand, "H.264/AVC in Wireless Environments," *IEEE Transactions on Circuits and systems for Video Technology*, vol. 13, pp. 657–673, Julho 2003.

[27] Qualcomm 1xEV-DO Web Papers, "1XEV-DO system architecture." http://www.qualcomm.com/cdma/1xEV/media/web_papers/wp_system_arch.pdf.

[28] M. Magalhães e E. Cardozo, *Qualidade de Serviço na Internet*. DCA-FEEC-UNICAMP, 1999.

[29] B. Davie, Y. Rekhter, *MPLS Technology and Applications*. Morgan Kaufmann, 2000.

[30] M. Magalhães e E. Cardozo, *Introdução à Comutação IP por Rótulos Através de MPLS*. DCA-FEEC-UNICAMP, 2001.

[31] Sistema Brasileiro de Televisão Digital, "Proposta de debate - Ministério das Comunicações." http://www.mc.gov.br/tv_digital1.htm.

[32] Advanced Television Standard Committee, "Website oficial." http://www.atsc.org/standards.html.

[33] Digital Video Broadcasting, "Website oficial." http://www.dvb.org.

[34] Integrated Services Digital Broadcasting , "Website oficial." http://www.dibeg.org.