# Compressão de imagens utilizando o padrão H.264

Paulo Henrique F. T. Soares[1], Rafael Lemos Paes[2] e Marcelo Silva Pinho[1]

[1]ITA - Pça Mal Eduardo Gomes, 50 - Vila das Acácias - São José dos Campos - SP.

[2]IEAv - Rodovia dos Tamoios, km 5,5 - Bairro Putim - São José dos Campos - SP.

São José dos Campos, 10 de julho de 2011

**Abstract** - The H.264 video coding standard is used from applications which high definition is required, such as at Blue-Rays, to lower bit rate uses, like video conference. Its intra prediction outperforms any image compression standard, but the price is the computer complexity. In this paper, it will be described briefly a way to achieve a good rate compression for images without demanding too much of a processor.

**Keywords** - H.264, compression, optimization.

## 1 Introdução

Desde o início do século XIX, desejava-se transmitir imagens à distância. Porém, só em 1842 obteve-se a primeira transmissão telegráfica de uma imagem *fac-símile*, atualmente conhecido como fax. Em meados do século XX, começaram a ser transmitidos os primeiros sinais de vídeo preto e branco. A partir de então, as técnicas de transmissão de vídeos vem sendo aperfeiçoadas para responder à demanda de usuários por formatos com melhor qualidade.

Em particular, nos últimos dez anos, a compressão de vídeos tem sido o aspecto técnico imprescindível para a utilização em larga escala de vídeos digitais. Desde formatos com menor qualidade, que visam rápidas transmissões via internet, até os utilizados em DVDs e Blue-Rays; todos utilizam a compressão de vídeos.

No que tange as atividades operacionais, a compressão de imagens é necessária em atividades que envolvam a rápida transmissão de dados. Atualmente, são diversas as aeronaves que possuem sistemas imageadores, em especial na Força Aérea Brasileira, desde câmeras fotográficas até sensores de varredura e eletroópticos. Todavia, a transmissão destas informações deve ser objetiva, descartando dados redundantes. Neste contexto, o estudo de técnicas de compressão de imagens contribui tanto para a velocidade quanto para a confiabilidade deste fluxo de dados. Imagine, por exemplo, um vídeo de baixa resolução enviado por uma aeronave com 425x355 pixeis. Como normalmente se tratam de 30 *frames* por segundo e 24 bits por pixel no formato RGB ( 1 Byte para cada camada ) seriam necessários

$$N = 425\text{x}355\text{x}30\text{x}24 = 110{,}9 \text{ MBytes/segundo}$$

isto é, um vídeo com 10 segundos teria cerca de 1 GB. Ou seja, é inviável a transmissão de vídeos sem compressão, mesmo que eles tenham baixa resolução.

Estas técnicas tem se tornado, portanto, uma importante ferramenta para os decisores em vários níveis em cenários operacionais da Força Aérea Brasileira.

Por outro lado, é natural se imaginar que, como os vídeos são um conjunto de imagens estáticas, uma boa compressão de vídeo requer uma compressão eficiente de imagens. De fato, um dos motivos pelos quais o padrão de compressão de vídeos H.264 consegue superar os demais é a sua capacidade de comprimir com grande eficácia cada *frame*. Seu algoritmo de compressão de imagens divide cada um destes em blocos, unidades onde serão efetuadas as predições, e utiliza os blocos já codificados para estimar o atual. Os resíduos desta predição passam pela transformada DCT *(Discrete Cossine Tranform)* e depois de serem quantizados, ou seja, divididos por um passo de quantização $Q_{step}$ são passados para o codificador de entropia.

A saída deste codificador é uma sequência de bits que representa a imagem codificada. A Fig. 1 ilustra o diagrama de blocos .

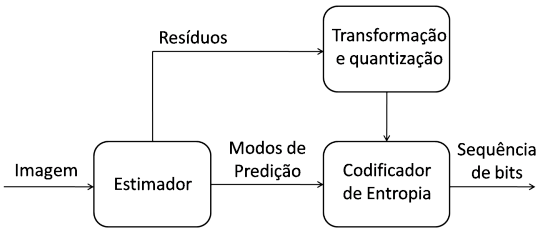


Figura 1: Diagrama de blocos do algoritmo de compressão de imagens utilizado pelo padrão H.264.

A seção 2 irá explicar como funciona a estimação, a seção 3 irá explicar sucintamente a codificação de entropia, a seção 4 irá explicar a solução proposta para minimizar a exigência computacional do algoritmo, enquanto a seção 5 irá apresentar os resultados obtidos.

## 2 Estimador

Cada *frame* é dividido em porções menores denominadas macroblocos, os quais possuem 16x16 pixeis. Estes, por sua vez, são subdivididos em 16 microblocos de 4x4 pixeis. O estimador tem como objetivo realizar uma estimativa de cada macrobloco utilizando modos de predição pré-determinados. Há 9 modos de predição que são utilizados nos microblocos e outros 4 aplicados nos macroblocos. A Fig. 2 mostra os possíveis modos 4x4 para os microblocos. Os três primeiros modos de predição 16x16 são análogos aos modos 0, 1 e 2 usados para os microblocos, enquanto o quarto modo é um gradiente em diagonal [2].

Para cada um dos 16 microblocos escolhe-se a estimativa que produz o menor soma absoluta dos erros (SAE)

$$\sum_{i=1}^{16}\sum_{i=1}^{16}|p_{ij}-\hat{p_{ij}}|$$

isto é, a menor soma das diferenças em módulo entre pixeis estimados e os reais. As 16 estimativas com menor SAE são escolhidas e formam uma predição para o macrobloco que está sendo estimado. Esta predição juntamente com as outras 4 do tipo 16x16 formam um conjunto de 5 possíveis estimativas para este bloco. Por fim, todas estas cinco predições são avaliadas e é escolhida aquela que minimiza uma determinada função custo.

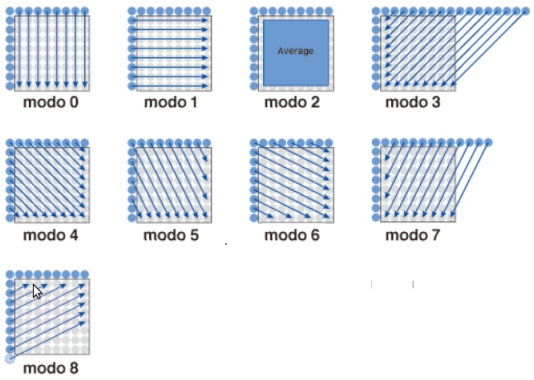


Figura 2: Modos de predição para blocos 4x4.

Para indicar o modo utilizado na predição, pode-se utilizar um bit de sinalização (*flag*) 0 caso o modo seja um dos 16x16 ou um *flag* 1 para indicar que a predição foi feita a partir das 16 melhores das predições 4x4. No primeiro caso, utilizam-se 2 bits para indicar qual foi o modo utilizado, uma vez que dois bits são suficientes para representar as 4 possibilidades[4].

Caso o modo escolhido para o macrobloco seja uma combinação de modos 4x4, então um esquema diferente é utilizado para indicar quais modos foram usados. Como blocos vizinhos na maioria dos casos tem modos iguais, o codificador define o modo mais provável para o microbloco atual como sendo o menor dos números que representam os modos do vizinho da esquerda e do de cima (caso nenhum dos blocos esteja disponível, então o modo 2 é usado como o mais provável). Por exemplo, se na figura abaixo os microblocos A e B já foram codificados com os modos 3 e 5, respectivamente então o modo mais provável para o microbloco atual, que é o C, será $min(3,5) = 3$

Caso este seja realmente o modo do bloco 4x4 atual, é enviado um *flag* 0 e a estimação de modos segue para o próximo microbloco. Se o modo do bloco 4x4 for diferente do o mais provável, envia-se o *flag* 1. Neste caso, para se indicar qual modo foi escolhido, há duas possibilidades

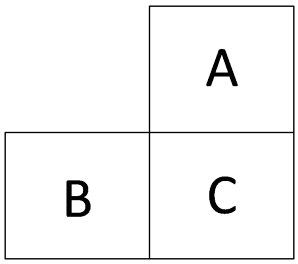


Figura 3: Imagem que ilustra um bloco a ser codificado (C) na vizinhança de outros dois já codificados (A e B).

1. o modo do bloco em questão é menor que o mais provável; ou

2. o modo do bloco em questão é maior que o mais provável.

No primeiro caso envia-se a representação binária de 3 bits do número que indica o modo observado. Enquanto no segundo, envia-se a representação do modo menos uma unidade [4].

Por exemplo, se o modo do bloco C for 3, então é enviado apenas o *flag* 0. Se o modo do bloco C for 5, que é maior que 3, então envia-se o *flag* 1 juntamente com a representação binária de $5-1 = 4 = (100)_2$, isto é, 1100. Se, por outro lado, o modo do bloco C for 1, que é menor que 3, envia-se o *flag* 1 juntamente com a representação binária de $1 = (001)_2$, ou seja, 1001.

Após a predição são gerados os resíduos, que são a diferença entre o macrobloco e sua estimativa. Os resíduos são transformados com a DCT 4x4, depois os resíduos são quantizados e, finalmente, enviados para o codificador de entropia.

# 3 Codificador de Entropia

Usar um número igual de bits para todos os símbolos só é uma boa alternativa quando estes são equiprováveis. Todavia, este nem sempre é o caso. No problema em questão, os resíduos transformados são em sua maioria zeros ou $\pm 1$, isto é, números de pequena amplitude. Portanto, faz sentido utilizar menos bits para números de menor amplitude. É o que faz o código de Golomb [4, 5], que será explicado abaixo. Todavia o código de Golomb não pode ser diretamente aplicado, pois na saída do quantizador podem haver números negativos.

Uma alternativa para fazer a codificação de entropia destes números é utilizar uma transformação em que os números negativos sejam mapeados em números ímpares positivos e os positivos em pares não negativos e, enfim, utilizar o código de Golomb. Isto é, fazer a transformação

$$T(n) = \begin{cases} 2n & \text{se } n \geq 0; \\ -2n-1 & \text{se } n < 0. \end{cases}$$

e depois utilizar o código de Golomb, que funciona da seguinte forma:

Primeiramente define-se $M$ como

$$M = \lfloor log_2(n+1) \rfloor$$

e INFO como

$$\text{INFO} = T(n) + 1 - 2^M$$

e a codificação resultante é dada pela sequência de M zeros, seguida de 1 e a representação binária de INFO, ou seja

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

Para codificar $n = -1$, por exemplo. Tem-se, após a transformação

$$T(n) = 1$$

daí segue que

$$M = 1$$

e

$$\text{INFO} = 0$$

portanto a representação de -1 fica 010.

Desta forma, após ser feita a codificação de entropia dos resíduos transformados e dos modos de predição, a imagem estará comprimida. Cabe ressaltar, porém, que o código de Golomb apesar de ser mais efetivo que utilizar um número igual de bits para todos os símbolos apresenta resultados bem inferiores a outros codificadores de entropia como o CAVLC[1] ou o CABAC[3], que são adaptativos [4].

# 4 Função Custo

Como mencionado, a escolha dos modos segue uma função custo. Normalmente, para escolher o melhor modo transformam-se os resíduos de cada uma das 5 estimativas e, para cada uma destas, estima-se quantos bits serão utilizados para representar tanto o modo quanto os resíduos codificados. A função custo a ser minimizada é, então, o número de bits utilizado em cada caso.

Note, porém, que observar cada estimativa do número de bits que todos os 5 conjuntos de resíduos do macrobloco produzem exige muito poder de processamento, especialmente se a imagem tiver grandes dimensões.

Uma alternativa a este modelo é a utilização de uma função custo Lagrangeana do tipo:

$$C = N.\lambda + \text{SAE}$$

onde SAE é a soma dos módulos das diferenças entre os 256 pixeis do bloco estimado e do real, N é o número de bits utilizado para indicar o modo (ou os modos) utilizados para se estimar o macrobloco e $\lambda$ é denominado fator de custo. Note que se for utilizado um *flag* para distinguir se o o modo de predição é um dos 16x16 ou se ele é predito com blocos 4x4, então para o primeiro caso utilizam-se $N = 3$ bits enquanto que no segundo requerem-se pelo menos $N = 17$ bits e, no máximo, $N = 49$.

De posse desta função, procurou-se um valor razoável para $\lambda$, a fim de que este valor escolhido seja bom para comprimir imagens níveis diferentes de detalhamento.

# 5 Resultados

Existem alguns sistemas de imagens que são bastante utilizados. No sistema RGB, por exemplo, cada imagem possui três camadas: uma referente à cor vermelha, outra à cor verde e uma última relativa à cor azul. Outro sistema que é de interesse para a compressão de imagens é o YUV. Este sistema também possui três camadas: a primeira que é chamada de camada Y ou de camada de luminosidade, e outras duas denominadas camada U e camada V. A camada de luminosidade (Y) nada mais é que uma versão preta e branca da imagem original enquanto as outras duas (U e V) são formadas a partir de combinações lineares das camadas R, G e B. Os resultados deste artigo contemplam somente as camadas de luminosidade de cada imagem. Todavia, é fácil estender o algoritmo de compressão de uma camada para três.

Para se obter uma boa taxa de compressão, deve-se estimar um fator de custo $\lambda$ que seja razoável tanto para imagens homogêneas, quanto para imagens com poucas regiões suaves. Feita esta estimação, a função custo (*supra* citada) estará definida e, então, o algoritmo proposto poderá ser avaliado.

A fim de se descobrir como a taxa de compressão varia em função do fator de custo $\lambda$, foram testadas imagens com diferentes características. Para os experimentos, foi utilizado o código de Golomb como codificador de entropia nestas avaliações. Finalmente como parâmetro de quantização utilizou-se $QP = 20$. Este parâmetro define o passo de quantização e gera uma qualidade de imagem muito boa do ponto de vista da visão humana.

A Fig. 4 mostra uma imagem de sensoriamento remoto original, enquanto a Fig. 5 apresenta a curva dos bits utilizados por pixel para cada valor de $\lambda$.

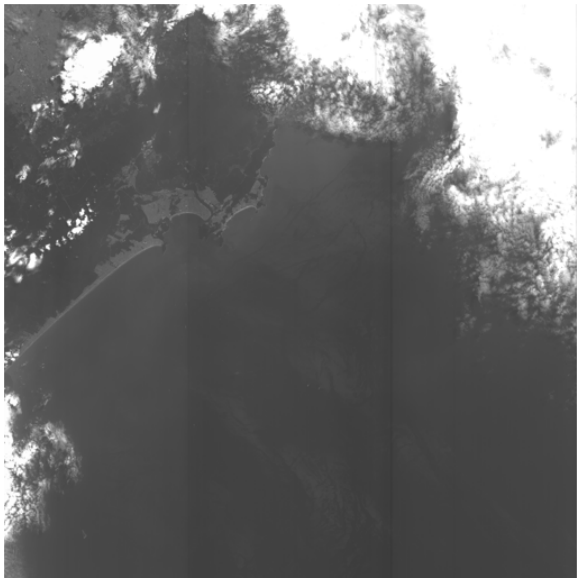

Figura 4: Imagem sensoriamento remoto original.

Neste ponto, cabe uma ligeira análise do gráfico contido na Fig. 5. Observe que se $\lambda = 0$ então a função custo se torna

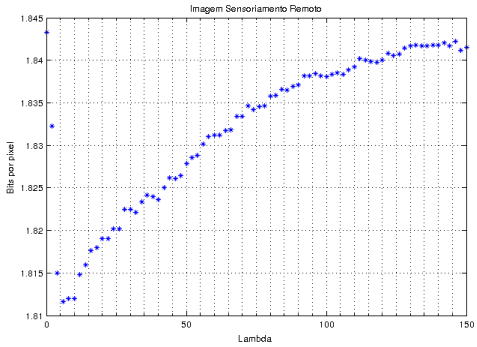


Figura 5: Curva bits por pixel em função de $\lambda$ para a imagem de sensoriamento remoto.

$$C(\lambda = 0) = \text{SAE}$$

isto é, como em geral estimações utilizando modos 4x4 geram erros menores, então a função custo $C(\lambda = 0)$ privilegia este tipo de predição. Todavia, esta escolha implica que muitos bits serão utilizados para sinalizar o tipo de predição. Isto faz com que a compressão não seja tão efetiva.

Por outro lado,

$$\lim_{\lambda \to \infty} \frac{C}{\lambda} = N$$

ou seja, quando $\lambda$ for suficientemente grande a predição irá privilegiar os modos que usem menos bits para indicá-los, uma vez que N é o número de bits necessários para indicar qual modo foi utilizado. Ou seja, ela será constituída em sua maioria por blocos 16x16. Como consequência a, predição não será tão eficiente pois os resíduos precisarão ser codificados com mais bits. Portanto, tendo em vista que os extremos de $\lambda$ não parecem ser favoráveis, então intuitivamente deve haver um ponto intermediário da reta $[0, \infty)$ em que $\lambda$ produza taxas de compressão razoáveis para diferentes tipos de imagens. De fato, isto é o que ocorre na Fig. 5: $\lambda$ tem torno de 10 produz a menor taxa de compressão.

Em seguida, foi codificada a imagem *football* ilustrada Fig. 6. Foi obtida também a curva bits por pixel em função de $\lambda$, a qual está apresentada na Fig. 7. Note que as curvas bits por pixel da imagem de sensoriamento remoto e da imagem *football* tem formas semelhantes (ambas possuem um mínimo local em torno de $\lambda = 10$). Isto ocorre pois ambas as imagens são bastante homogêneas, apesar de apresentarem regiões com maiores níveis de detalhamento.

Figura 6: Imagem *football* original.

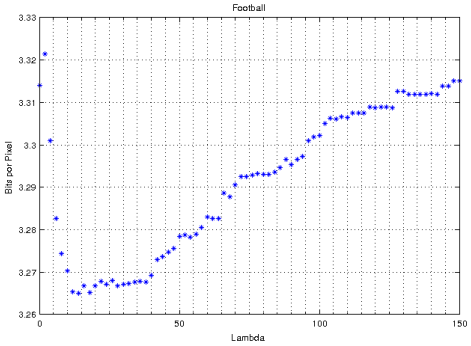


Figura 7: Curva bits por pixel em função de $\lambda$ para a imagem *football*.

Em seguida foi testada a imagem *carphone*, que está ilustrada na Fig. 8 e apresenta poucas regiões suaves. A Fig. 9 é a sua curva bits por pixel em função de $\lambda$.

Note que como a imagem *carphone* possui poucas regiões suaves, a imagem é melhor comprimida usando $\lambda = 0$ (privilegiando predições 4x4). Conforme mais modos 16x16 são utilizados, mais a eficiência da compressão é penalizada devido ao erro de predição se tornar cada vez maior.

Finalmente, foi testada a imagem *tempete* que praticamente não contém regiões suaves. A Fig. 10 mostra esta imagem.

De fato, a curva bits por pixel em função do fator de custo $\lambda$ ilustrada na Fig. 11 mostra que o

Figura 8: Imagem *carphone* original.

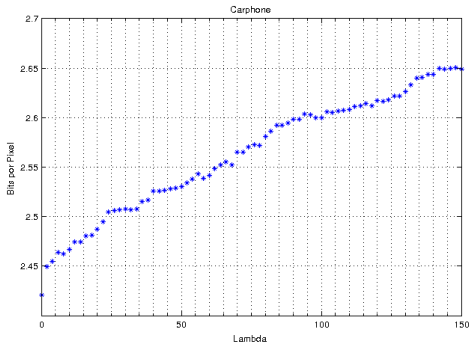


Figura 9: Curva bits por pixel em função de $\lambda$ para a imagem *carphone*.

Figura 10: Imagem *tempete* original.

resultado é bastante semelhante ao da imagem *carphone*, conforme esperado devido ao alto nível de detalhes apresentado por ambas.

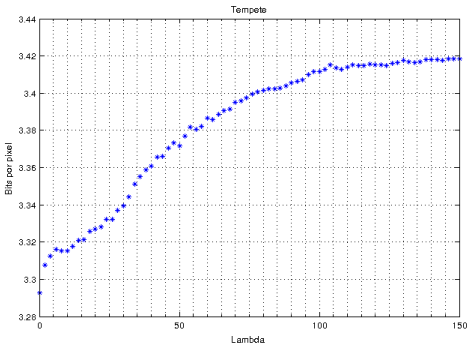


Figura 11: Curva bits por pixel em função de $\lambda$ para a imagem *tempete*.

Quanto ao custo computacional, o algoritmo foi implementado utilizando a técnica proposta e seu tempo total de execução foi calculado ($t_{prop}$). Este tempo foi comparado com o tempo de execução do algoritmo de referência[6] ($t_{ref}$). A Tab. 1 abaixo mostra os resultados obtidos simulados no *software* MATLAB.

| imagem | $t_{prop}$ | $t_{ref}$ |
|---|---|---|
| *carphone* | 3,36s | 3,91s |
| *football* | 11,86s | 12,10s |
| *tempete* | 13,89s | 14,41s |
| Sensoriamento Remoto | 35,21s | 37,27s |

Tab. 1 - Comparação entre os tempos de execução do algoritmo proposto e o de referência.

Finalmente, seguem as quatro imagens comprimidas nas Fig. 12, 13, e 14, onde suas respectivas taxas de compressão utilizando-se $\lambda = 10$ se encontram discriminadas na Tab. 2. Além disso, é mencionada nas figuras a PSNR (*Peak Signal to Noise Ratio*) obtida. Esta medida de qualidade da figura comprimida é definida como

$$PSNR = 10log_{10}\left[\frac{A_{max}}{E[|e|^2]}\right]$$

onde $A_{max}$ é a amplitude máxima de um pixel da figura original e $|e|$ é a diferença absoluta entre os pixeis da figura original e a reconstruída.

Figura 12: Imagem *carphone* comprimida com $\lambda = 10$ e PSNR = 43,27.

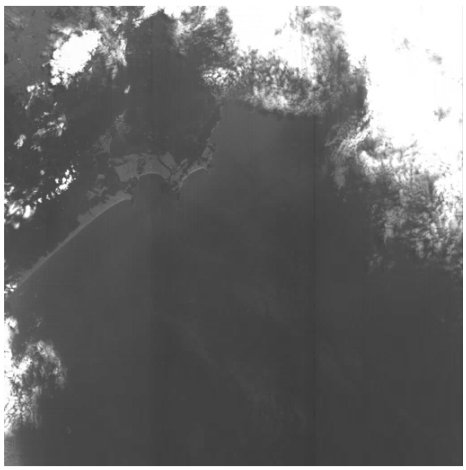

Figura 13: Imagem sensoriamento remoto comprimida com $\lambda = 10$ e PSNR = 46,34.

Figura 14: Imagem *football* comprimida com $\lambda = 10$ e PSNR = 42,58.

Figura 15: Imagem *tempete* comprimida com $\lambda = 10$ e PSNR = 42,45.

| imagem | taxa (bits por pixel) |
|---|---|
| *carphone* | 2,45 |
| *football* | 3,27 |
| *tempete* | 3,32 |
| Sensoriamento Remoto | 1,81 |

Tab. 2 - Comparação entre as taxas de compressão.

Portanto, foram obtidas imagens com taxas de compressão razoáveis, sem grandes distorções e com um algoritmo menos complexo que o algoritmo de referência.

# 6 Conclusão

Dentro das condições expostas, viu-se que usando $\lambda$ em torno de $\lambda = 10$ se consegue uma boa taxa de compressão para imagens mais homogêneas. Por outro lado, nas imagens com menos regiões suaves observou-se que $\lambda$ em torno de 10 produz uma compressão razoável se comparada com aquela obtida com $\lambda = 0$, que é melhor nestes casos. Logo, para o fator de quantização $QP = 20$, $\lambda = 10$ é uma boa escolha tanto para imagens com poucas regiões suaves como para imagens mais homogêneas.

Além disso, com a função custo apresentada, observou-se que o algoritmo conseguiu codificar as imagens com maior rapidez que o algoritmo de referência. Ou seja, com esta função foi possível

obter um algoritmo de menor complexidade. Isto se deve ao fato de que, com o algoritmo proposto, não haver a necessidade de transformar e quantizar os resíduos para enfim calcular o número total de bits utilizado para cada modo de estimação.

## Bibliografia

[1] G. Bjøntegaard and K. Lillevold, *Context-adaptive vlc coding of coefficients*, (2002).

[2] Y.-L. S. Lin, *Vlsi design for video coding: H.264/avc encoding from standard specification to chip*, Springer, 2010.

[3] H. Schwarz Marpe and T. Wiegand, *Context-based adaptive binary arithmetic coding in the h.264 / avc video compression standard*, (2003).

[4] I. E. Richardson, *The h.264 advanced video compression standard*, Wiley and Sons, 2010.

[5] K. Sayood, *Introdution to data compression*, Morgan Kaufman Publishers, 2000.

[6] T. Wiegand, G. J. Sullivan, and A. Luthra, *Draft ITU-T recommendation and final draft international standard of joint video specification*, (2003).