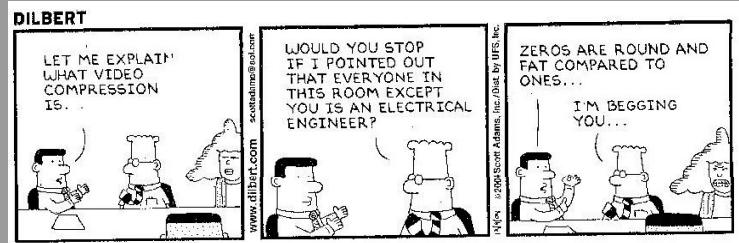


## Module 7

### Image Compression

- Lossless Image Coding
- Lossy Image Coding
- JPEG Image Compression Standards
- Wavelet Compression

1



4

### OBJECTIVES OF IMAGE COMPRESSION

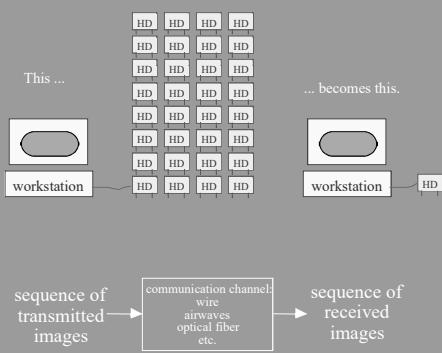
- Create a **compressed** image that "looks the same" (when decompressed) but can be stored in a fraction of the space.
- **Lossless compression** means the image can be exactly reconstructed.
- **Lossy compression** means that visually redundant information is removed. The decompressed image "looks" unchanged, but mathematically has lost information.
- **Image compression** is important for:
  - Reducing image storage space
  - Reducing image transmission bandwidth

2

### **Lossless Image Compression**

5

### Significance for Storage & Transmission



Compressed images can be sent at an increased rate: More images per second !

3

### Image Compression Measures

- **Bits Per Pixel (BPP)** is the **average** number of bits required to store the gray level of each pixel in  $I$ .
- **Uncompressed image** ( $K$  is gray scale range):
 
$$\text{BPP}(I) = \log_2(K) = B$$
- Usually
 
$$B = \log_2(256) = 8.$$

6

## Variable-Bit Codes

- The number of bits used to code pixels may **spatially vary**.
- Suppose that  $\hat{I}$  is a compressed version of  $I$ .
- Let  $B(i, j) = \#$  of bits used to code pixel  $I(i, j)$ . Then

$$BPP(\hat{I}) = \frac{1}{NM} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} B(i, j)$$

- If **total** number of bits in  $\hat{I}$  is  $B_{\text{total}}$ , then

$$BPP(\hat{I}) = \frac{1}{NM} B_{\text{total}}$$

7

## Methods for Lossless Coding

- Basically amounts to clever arrangement of the data.
- This can be done in many ways and in many domains (DFT, DCT, wavelet, etc)
- The most popular methods use **variable wordlength coding**.

10

## Compression Ratio

- The **Compression Ratio (CR)** is

$$CR = \frac{BPP(I)}{BPP(\hat{I})} > 1$$

- Both **BPP** and **CR** are used frequently.

8

## Variable Wordlength Coding (Huffman Code)

11

## LOSSLESS IMAGE COMPRESSION

- Lossless techniques achieve compression with **no loss of information**.
- The true image can be reconstructed **exactly** from the coded image.
- Lossless coding doesn't usually achieve **high compression** but has definite applications:
  - In **combination** with lossy compression, multiplying the gains
  - In applications where information loss is **unacceptable**.
- Lossless compression ratios **usually** in the range

$$2:1 \leq CR \leq 3:1$$

but this may **vary** from image to image.

9

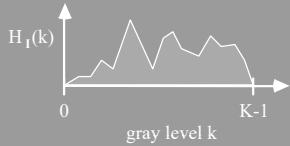
## Variable Wordlength Coding (VWC)

- **Idea:** Use **variable wordlengths** to code gray levels.
- Assign **short wordlengths** to gray levels that occur **frequently** (redundant gray levels).
- Assign **long wordlengths** to gray levels that occur **infrequently**.
- On the average, the **BPP will be reduced**.

12

## Image Histogram and VWC

- Recall the **image histogram**  $H_I(k)$ :



- Let  $B(k) = \#$  of bits used to code gray-level  $k$ . Then

$$BPP(\hat{I}) = \frac{1}{NM} \sum_{k=0}^{K-1} B(k)H_I(k)$$

- This is the **common measure of BPP for VWC**.

13

## Maximum Entropy Image

- The entropy is **maximized** when

$$p_I(k) = \frac{1}{K}; k=0, \dots, K-1$$

corresponding to a **flat histogram**. Then ( $K = 2^B$ )

$$E[I] = -\sum_{k=0}^{K-1} \frac{1}{K} \log_2 \frac{1}{K} = B \cdot \sum_{k=0}^{K-1} \frac{1}{K} = B$$

- Generally, image entropy increases as the histogram is spread out.

16

## Image Entropy

- Recall the **normalized histogram** values

$$p_I(k) = \frac{1}{NM} H_I(k); k=0, \dots, K-1$$

so  $p_I(k)$  = probability of gray level  $k$

- The **entropy** of image  $I$  is then

$$E[I] = -\sum_{k=0}^{K-1} p_I(k) \log_2 p_I(k)$$

14

## Minimum Entropy Image

- The entropy is **minimized** when

$$p_I(k_n) = 1 \text{ for some } k_n; 0 \leq k_n \leq K-1$$

hence  $p_I(k_m) = 0$  for  $m \neq n$ .

- This is a **constant image** and  $E[I] = 0$ .

- In fact it is always true that

$$0 \leq E[I] \leq B$$

17

## Meaning of Entropy

- Entropy** is a **measure of information** with nice properties for designing VWC algorithms.
- Entropy**  $E[I]$  measures
  - Complexity
  - Information Content
  - Randomness

15

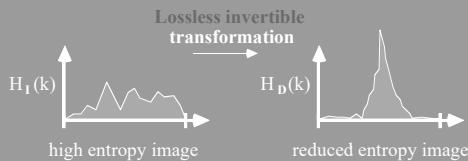
## Significance of Entropy

- An important theorem limits how much an image can be compressed by a lossless VWC:
- $$BPP(\hat{I}) \geq E[I]$$
- A VWC is assumed **uniquely decodable**.
  - The  $p_I(k)$  are assumed known at both transmitter and receiver.
  - Thus an image with a flat histogram:  $E[I] = B$  cannot be compressed by a VWC alone. Fortunately, we can often fix this situation by **entropy reduction**.
  - Moreover, a **constant image** need not be sent:  $E[I] = 0$ !
  - The entropy provides a compression lower bound as a **target**.

18

## Image Entropy Reduction

- **Idea:** Compute a new image  $\mathbf{D}$  with a more compressed histogram than  $\mathbf{I}$  but with no loss of information.



- Must be able to recover  $\mathbf{I}$  exactly from  $\mathbf{D}$  - no loss of information.
- Compressing the histogram (Module 3) won't work, since information is lost when two gray levels  $k_1$  and  $k_2$  are mapped to the same new gray level  $k_3$ .

19

## Entropy Reduction by Differencing (Simple DPCM)

- Differential pulse-code modulation (DPCM) is effective for lossless image entropy reduction.
- Images are mostly smooth: neighboring pixels often have similar values.
- Define a difference image  $\mathbf{D}$  using either 1-D or 2-D differencing:
  - (1-D)  $D(i, j) = I(i, j) - I(i, j-1)$
  - (2-D)  $D(i, j) = I(i, j) - I(i-1, j) - I(i, j-1) + I(i-1, j-1)$   
for  $0 \leq i \leq N-1, 1 \leq j \leq M-1$ .
- The new histogram  $H_D$  usually will be more compressed than  $H_I$ , so that

$$E[\mathbf{D}] < E[\mathbf{I}]$$

- This is a rule of thumb for images, not a math result.

20

## Reversing DPCM

- If (1) is used, then  
 $I(i, j) = D(i, j) + I(i, j-1)$
- The first column of  $\mathbf{I}$  must also be transmitted.
- If (2) is used, then  
 $I(i, j) = D(i, j) + I(i-1, j) + I(i, j-1) - I(i-1, j-1)$   
 where the first row and first column of  $\mathbf{I}$  must also be transmitted.
- The overhead of the first row and column is small, but they can be separately compressed.
- Hereafter we will assume an image  $\mathbf{I}$  has either been histogram compressed by DPCM or doesn't need to be.

21

## Optimal Variable Wordlength Code

- Recall the theoretical lower bound using a VWC:

$$BPP(\hat{\mathbf{I}}) \geq E[\mathbf{I}] = -\sum_{k=0}^{K-1} p_I(k) \log_2 p_I(k)$$

- Observe: In any VWC, coding each gray-level  $k$  using wordlength  $L(k)$  bits gives an average wordlength:

$$BPP(\hat{\mathbf{I}}) = \sum_{k=0}^{K-1} p_I(k) L(k)$$

- Compare with the above. If  $L(k) = -\log_2 p_I(k)$  then an optimum code has been found - lower bound attained!

22

## Optimal VWC

- IF we can find such a VWC such that  
 $L(k) = -\log_2 p_I(k)$  for  $k = 0, \dots, K-1$   
 then the code is **optimal**.
- It is impossible if  $-\log_2 p_I(k) \neq$  an integer for **some**  $k$ .
- So: define an **optimal code**  $\hat{\mathbf{I}}$  as one that satisfies:
  - $BPP(\hat{\mathbf{I}}) \leq BPP(\hat{\mathbf{I}}')$  of any other code  $\hat{\mathbf{I}}'$
  - $BPP(\hat{\mathbf{I}}) = E[\mathbf{I}]$  if  $[-\log_2 p_I(k)] =$  integers

23

## The Huffman Code

- The **Huffman algorithm** yields an **optimum code**.
- For a set of gray levels  $\{0, \dots, K-1\}$  it gives a set of **code words**  $c(k); 0 \leq k \leq K-1$  such that

$$BPP(\hat{\mathbf{I}}) = \sum_{k=0}^{K-1} p_I(k) L[c(k)]$$

is the **smallest possible**.

24

## Huffman Algorithm

- Form a **binary tree** with branches labeled by the gray-levels  $k_m$  and their probabilities  $p_I(k_m)$ :

(0) Eliminate any  $k_m$  where  $p_I(k_m) = 0$ .

(1) Find 2 **smallest probabilities**  $p_m = p_I(k_m)$ ,  $p_n = p_I(k_n)$ .

(2) Replace by  $p_{mn} = p_m + p_n$  to form a **node**; reduce list by 1

(3) Label the branch for  $k_m$  with (e.g.) '1' and for  $k_n$  with '0'.

(4) Until list has only 1 element (root reached), return to (1).

- In step (3), values '1' and '0' are assigned to element pairs ( $k_m, k_n$ ), elements triples, etc. as the process progresses.

25

## Huffman Tree Example 2

- There are  $K = 8$  values  $\{0, \dots, 7\}$  to be assigned codewords:

$$p_I(0) = 0.4 \quad p_I(4) = 0.12$$

$$p_I(1) = 0.08 \quad p_I(5) = 0.08$$

$$p_I(2) = 0.08 \quad p_I(6) = 0.04$$

$$p_I(3) = 0.2 \quad p_I(7) = 0.00$$

28

## Huffman Tree Example 1

- There are  $K = 8$  values  $\{0, \dots, 7\}$  to be assigned codewords:

$$p_I(0) = 1/2 \quad p_I(4) = 1/16$$

$$p_I(1) = 1/8 \quad p_I(5) = 1/32$$

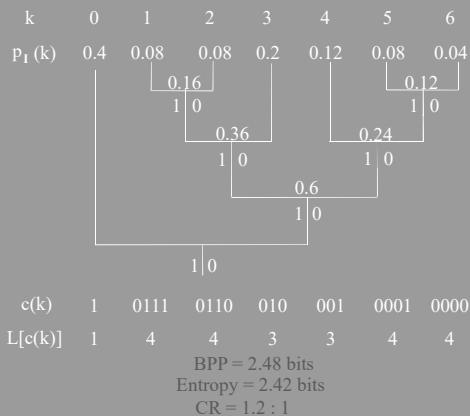
$$p_I(2) = 1/8 \quad p_I(6) = 1/32$$

$$p_I(3) = 1/8 \quad p_I(7) = 0$$

- The process creates a **binary tree**, with values '1' and '0' placed (e.g.) on the right and left branches at each stage:

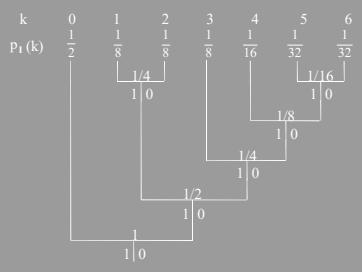
26

## Huffman Tree Example 2



29

## Huffman Tree Example 1



c(k)	1	0111	0110	001	0001	00000
L[c(k)]	1	3	3	3	4	5

BPP = Entropy = 2.1875 bits  
CR = 1.37 : 1

27

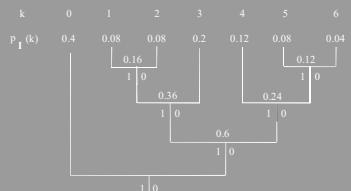
## Huffman Decoding

- The Huffman code is a **uniquely decodable** code. There is **only one interpretation** for a series of codewords (series of bits).
- Decoding progresses by **traversing the tree**.

30

## Huffman Decoding Example

- In the second example, this sequence is received:  
0001011010111000010000100010110111010
- It is sequentially examined until a codeword is identified. This continues until all are identified:  
0001 0110 1 0111 0000 010 0001 0001 0110 1 1 1 010
- The decoded sequence is: 5 2 0 1 6 3 5 5 2 0 0 0 3



31

## LOSSY IMAGE CODING

- Many approaches proposed.
- We will review a few popular approaches.
  - Block Truncation Coding (BTC) – simple, fast
  - Vector Quantization Coding (VQC) - costly but high-quality results
  - Discrete Cosine Transform (DCT) Coding and the JPEG Standard
  - Wavelet Image Coding and JPEG-2000

34

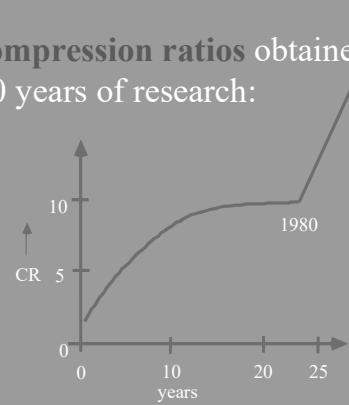
## Comments on Huffman Coding

- Huffman image compression usually  
 $2:1 \leq CR \leq 3:1$
- Huffman codes are quite noise-sensitive (much more so than an uncompressed image).
- Error-correction coding can improve this but increases increase the coding rate somewhat.
- Some Huffman codes aren't computed from the image statistics - instead assume "typical" frequencies of occurrence of values (gray-level or transform values) – as in JPEG.

32

## History of Image Compression Advances

- The compression ratios obtained over the first 30 years of research:



35

## Lossy Image Compression

33

## Goals of Lossy Coding

- To optimize and balance the three C's
  - Compression achieved by coding
  - Computation required by coding and decoding
  - Quality of the decompressed image

36



Artistic image coding with significant loss  
(Cezanne)



Artistic image coding with little loss  
(Degas)

37

## Why Block Coding?

- **Reason:** images are **highly nonstationary**: different areas of an image may have **different properties**, e.g., more high or low frequencies, more or less detail etc.
- Thus, **local coding is more efficient**. Wavelet methods provide localization without blocks.
- Typical block sizes: **4 x 4, 8 x 8, 16 x 16**.

40

## Broad Methodology of Lossy Compression

- There have been **many** proposed lossy compression methods.
- The successful broadly (and loosely) follow three steps.
  - (1) Transform the image to another domain and/or extract specific features.
  - (2) Quantize in this domain or those features.
  - (3) Efficiently organize and/or entropy code the quantized data.

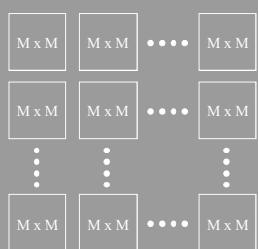
38

## **Block Truncation Coding (BTC)**

41

## Block Coding of Images

- **Most** lossy methods begin by partitioning the image into **sub-blocks** that are individually coded. Wavelet methods are an exception.



39

## Block Truncation Coding (BTC)

- **Fast**, but limited compression.
- Uses **4 x 4 blocks** each containing  $16 \cdot 8 = 128$  bits.
- Each  $4 \times 4$  block is coded identically so no need to index them.
- Consider the coding of a **single block** which we will denote  $\{I(1), \dots, I(16)\}$ .

42

## BTC Coding Algorithm

- (1) Quantize block sample mean:

$$\bar{I} = \frac{1}{16} \sum_{p=1}^{16} I(p)$$

and transmit/store it with  $B_1$  bits.

- (2) Quantize block sample standard deviation:

$$\bar{\sigma}_I = \sqrt{\frac{1}{16} \sum_{p=1}^{16} [I(p) - \bar{I}]^2}$$

and transmit/store it with  $B_2$  bits.

43

## BTC Coding Algorithm

- Compute a 16-bit binary block:

$$b(p) = \begin{cases} 1 & ; \text{ if } I(p) \geq \bar{I} \\ 0 & ; \text{ if } I(p) < \bar{I} \end{cases}$$

121	114	56	47
37	200	247	255
16	0	12	169
43	5	7	251

$\bar{I} = 98.75$

1	1	0	0
0	1	1	1
0	0	0	1
0	0	0	1

which requires 16 bits to transmit/store.

44

## BTC Quantization

- The quality/compression of BTC-coded images depends on the **quantization** of mean and standard deviation.
- If  $B_1 = B_2 = 8$ , 32 bits / block are transmitted:  
 $CR = 128 / 32 = 4:1$
- OK quality if  $B_1 = 6$ ,  $B_2 = 4$  (26 bits total):  
 $CR = 128 / 26 \approx 5:1$
- Using  $B_1 > B_2$  is OK: the eye is quite sensitive to the **presence** of variation, but **not to the magnitude** of variation:



Mach Band Illusion

45

## BTC Block Decoding

- To form the "decoded" pixels  $J(1), \dots, J(16)$ :

(1) Let  $Q = \text{number of '1's}$ ,  $P = \text{number of '0's}$  in binary block.

(2) If  $\begin{cases} b(p)=1 & ; \text{ set } J(p) = \bar{I} + \bar{\sigma}_I / A \\ b(p)=0 & ; \text{ set } J(p) = \bar{I} - \bar{\sigma}_I \cdot A \end{cases}$

where

$$A = \sqrt{\frac{Q}{P}}$$

- It is possible to show that this forces  $\bar{J} \approx \bar{I}$  and  $\bar{\sigma}_J \approx \bar{\sigma}_I$

46

## BTC Block Decoding Example

- In our example:  $Q = 7$ ,  $P = 9$ ,  $A = 0.8819$

$$\bar{I} = \text{INT}[98.75 + 0.5] = 99$$

$$\bar{\sigma}_I = \text{INT}[92.95 + 0.5] = 93$$

- If  $\begin{cases} b(p)=1 & ; \text{ set } J(p) = 99 + \text{INT}[93 \cdot 0.882 + 0.5] = 204 \\ b(p)=0 & ; \text{ set } J(p) = 99 - \text{INT}[93 \cdot 0.882 + 0.5] = 17 \end{cases}$

DEMO

204	204	17	17
17	204	204	204
17	17	17	204
17	17	17	204

Here  $\bar{J} \approx 98.8$

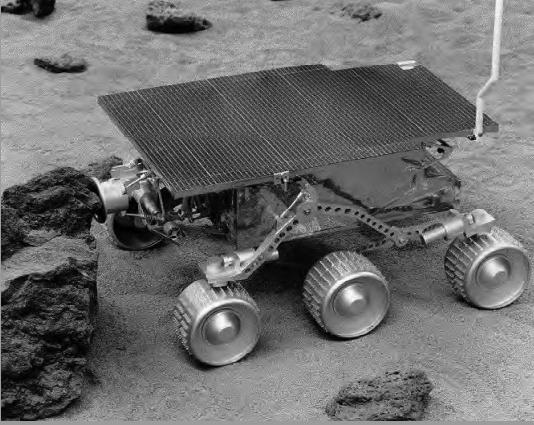
$\bar{\sigma}_J \approx 77.3$

47

## Comments on BTC

- Attainable compressions by simple BTC in the range 4:1 – 5:1.
- Combining with **entropy coding** of the quantized data, in the range 10:1.
- Popular in **low-bandwidth, low-complexity** applications. Used by the Mars Pathfinder!

48



49

## VQ Codebook

- The image  $\mathbf{I}$  is broken into  $4 \times 4$  blocks  $\mathbf{b}_m$ ;  $m = 1, \dots, MN/16$ .
- From these, compute a **codebook** of  $P$  "typical"  $4 \times 4$  blocks  $\mathbf{t}_p$ ,  $p = 1, \dots, P$ .
- This is done using a **clustering algorithm** which divides a space of points into likely groups according to a distance criterion.
- Usually  $256 \leq P \leq 1024$ .
- Simplest: Use K-Means Clustering with  $K=P$ .

52

## Vector Quantization Image Compression

50

### K-Means: Simple Clustering Method

- Given  $MN/16$  16-D  $4 \times 4$  blocks  $\mathbf{b}_m$ ;  $m = 1, \dots, MN/16$ .
- Partition  $\mathbf{b}_m$  into  $K$  16-D clusters  $C = \{C_1, C_2, \dots, C_K\}$  by minimizing the within-cluster sum-of-squares:
$$\arg \min_C \sum_{k=1}^K \sum_{b_j \in C_k} \|b_j - \bar{b}_k\|$$

where the  $\bar{b}_k$  are the **block means** (centroids).

- Lloyd's Algorithm:** Select initial means  $\{\bar{b}_1^{(0)}, \bar{b}_2^{(0)}, \dots, \bar{b}_K^{(0)}\}$  (e.g. randomly from the  $\mathbf{b}_m$ ). Then iterate between two steps:
  - Assign  $\mathbf{b}_m$ ;  $m = 1, \dots, MN/16$  to cluster  $q$  with mean  $\bar{b}_q^{(0)}$  closest to  $\mathbf{b}_m$ .
  - Calculate new cluster means  $\bar{b}_k^{(r+1)} = \frac{1}{\text{card}[C_k^{(r)}]} \sum_{b_j \in C_k^{(r)}} b_j$
- Complexity:**  $O\left[\left(\frac{MN}{16}\right)^{16P+1} \log\left(\frac{MN}{16}\right)\right]$

53

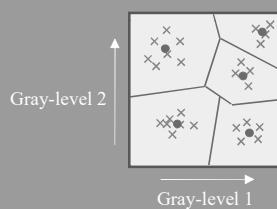
## Vector Quantization Image Coding

- A very light overview!
- Actually, there are **many variations** of this technique.
- Coding via VQ is very computation-intensive.** **Decoding** is not.
- VQ algorithms also (usually) use  **$4 \times 4$  blocks** – each of which is coded identically.

51

## $2 \times 1$ Codebook Example

- If  $2 \times 1$  blocks were used, the space might look like this:



- The cluster **centroids** are the coordinates of the "typical" blocks!
- In VQ image compression, the space is **16-D**, rather than 2-D.

54

## Codebook Computation

- Computation of the codebook directly from the image is very time-consuming.
- Unrealistic for real-time encoding.
- Approximate (sub-optimal) algorithms exist which are faster – e.g., Equitz Algorithm.
- A "universal" codebook can be used in applications where the images are always similar. Compute the codebook once from many images, then don't change it.

55

## VQ Comments

- VQ is an excellent example of an **unbalanced codec** (coder-decoder).
- Great for apps where coding time is less important but **decoding time is very important**: e.g., image libraries.
- Compression ratios CR = 15:1 - 20:1 are **typical**.

58

## Codebook Search

- If codebook is computed from **I**, finding the centroid of the cluster that block **b** belongs to is **simple**.
- If a universal codebook is used, then the closest centroid must be **searched for**.
- The typical block  $t_{q^*}$  is found that is closest to **b**:

$$q^* = \arg \min_q \sqrt{\frac{1}{16} \sum_{p=1}^{16} [b(p) - t_q(p)]^2}$$

- The index  $q^*$  is the **code** for **b**! This is what is stored or transmitted.

56

## Simple VQ Example 1



16 : 1

32 : 1

64 : 1

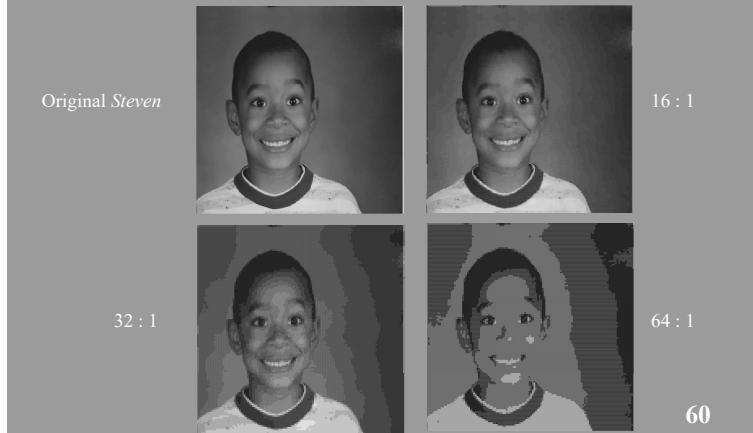
59

## VQ Decoding

- While codebook construction and search are very time consuming ....
- Decoding is **very fast!** Just a codebook look-up!
- Of course the codebook is **known by the decoder** (communications scenario).
- Given a code (index)  $q$  for block **b**, the typical code  $t_q$  represents **b** in the decompressed image.

57

## Simple VQ Example 2



16 : 1

32 : 1

64 : 1

60

## Classified VQ

- One effective variation is **classified VQ** where the block means and standard deviations are computed, and the blocks normalized by them:

$$\beta(p) = \frac{b(p) - \bar{b}}{\bar{\sigma}_b}$$

- The moments  $\bar{b}$  and  $\bar{\sigma}_b$  are quantized and stored/transmitted separately.
- VQ is applied to the normalized blocks  $\beta$  to form a “classified codebook” which can be much smaller or more accurate.

61

## JPEG

62

## JPEG

- The **JPEG Standard** remains the most widely-used method of compressing images.
- JPEG = **Joint Photographic Experts Group**, an international standardization committee.
- Standardization allows device and software manufacturers to have an **agreed-upon format** for compressed images.
- In this way, everyone can “talk” and “share” images.
- The overall JPEG Standard is quite complex, but the core algorithm is based on the **Discrete Cosine Transform (DCT)**.

63

## Discrete Cosine Transform (DCT)

- The **DCT** of an  $N \times M$  image or sub-image:

$$\tilde{I}(u, v) = \frac{4C_N(u)C_M(v)}{NM} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} I(i, j) \cos\left[\frac{(2i+1)u\pi}{2N}\right] \cos\left[\frac{(2j+1)v\pi}{2M}\right]$$

- The **IDCT**:

$$I(i, j) = \sum_{u=0}^{N-1} \sum_{v=0}^{M-1} C_N(u)C_M(v) \tilde{I}(u, v) \cos\left[\frac{(2i+1)u\pi}{2N}\right] \cos\left[\frac{(2j+1)v\pi}{2M}\right]$$

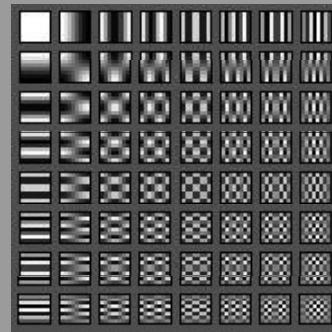
where

$$C_N(u) = \begin{cases} \frac{1}{\sqrt{2}} & ; u = 0 \\ 1 & ; u = 1, \dots, N-1 \end{cases}$$

64

## DCT Basis Functions

- Displayed as  $8 \times 8$  images:



65

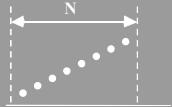
## DCT vs. DFT

- JPEG is based on quantizing and re-organizing block DCT coefficients.
- The DCT is very similar to the DFT. **Why use the DCT?**
- First,  $O(N^2 \log N^2)$  algorithms exist for DCT - slightly faster than DFT: all real, **integer-only** arithmetic.
- The DCT yields **better-quality compressed images** than the DFT, which suffers from more serious **block artifacts**.
- Reason:** The DFT implies the image is  $N$ -periodic, whereas the DCT implies the **once-reflected image** is  $2N$ -periodic.

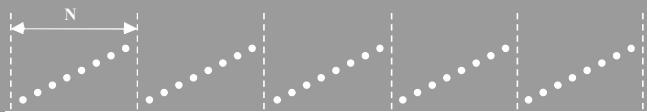
66

## Periodicity Implied by DFT

- A length-N (1-D) signal:



- Periodic extension implied by the DFT:



67

## Periodicity Implied by DCT

- Periodic extension of reflected signal implied by the DCT:



- In fact (good exercise): The DFT of the length- $2N$  reflected signal yields (two periods of) the DCT of the length- $N$  signal.

68

## Overview of JPEG

- The commercial industry standard - formulated by the CCIT Joint Photographic Experts Group (JPEG).
- Uses the DCT as the central transform.
- Overall JPEG algorithm is quite complex. Three components:
  - JPEG baseline system (basic lossy coding)
  - JPEG extended features (12-bit; progressive; arithmetic)
  - JPEG lossless coder

70

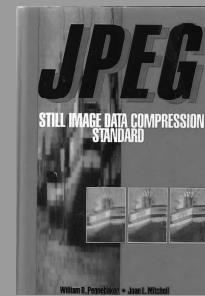
## Quality Advantage of DCT Over DFT

- The periodic extension of the DFT contains high frequency discontinuities.
- These high frequencies must be well-represented in the code, or the edge of the blocks will degrade, creating visible blocking artifacts.
- The DCT does NOT have meaningless discontinuities to represent - so less to encode, and so, higher quality for a given CR.

69

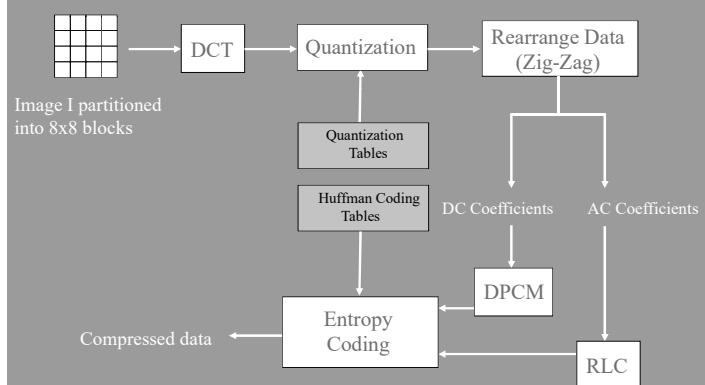
## JPEG Standard Documentation

- The written standard is gigantic. Best resource (Pennebaker and Mitchell):



71

## JPEG Baseline Flow Diagram



## JPEG Baseline Algorithm

- (1) Partition image into **8 x 8 blocks**, transform each block using the DCT. Denote these blocks by:  $\tilde{\tilde{I}}_k(u, v)$
- (2) Pointwise divide each block by an **8 x 8 user-defined normalization array Q(u, v)**. This is **stored / transmitted** as part of the code. Usually designed using sensitivity properties of **human vision**.
- (3) The JPEG committee performed a human study on the **visibility of DCT basis functions** under a specific viewing model.

73

## JPEG Baseline Algorithm

- (3) Uniformly quantize the result

$$\tilde{\tilde{I}}_k^Q(u, v) = \text{INT} \left[ \frac{\tilde{\tilde{I}}_k(u, v)}{Q(u, v)} + 0.5 \right]$$

yielding an **integer array with many zeros**.

74

## JPEG Quantization Example

- A block DCT (via integer-only algorithm)

$$\tilde{\tilde{I}}_k = \begin{bmatrix} 1260 & -1 & -12 & -5 & 2 & -2 & -3 & 1 \\ -23 & -17 & -6 & -3 & -3 & 0 & 0 & -1 \\ -11 & -9 & -2 & 2 & 0 & -1 & -1 & 0 \\ -7 & -2 & 0 & 1 & 1 & 0 & 0 & 0 \\ -1 & -1 & 1 & 2 & 0 & -1 & 1 & 1 \\ 2 & 0 & 2 & 0 & -1 & 1 & 1 & -1 \\ -1 & 0 & 0 & -1 & 0 & 2 & 1 & -1 \\ -3 & 2 & -4 & -2 & 2 & 1 & -1 & 0 \end{bmatrix}$$

- Typical JPEG normalization array:

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

75

## JPEG Quantization Example

- The resulting quantized DCT array:

$$\tilde{\tilde{I}}_k^Q = \begin{bmatrix} 79 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

- Notice all the **zeros**. The **DC value**  $\tilde{\tilde{I}}_k^Q(0, 0)$  is in the upper left corner.
- This works because the DCT has **good energy compaction**, leading to a sparse image representation.

76

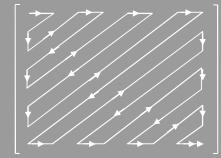
## Data Re-Arrangement

- Rearrange quantized AC values

$$\tilde{\tilde{I}}_k^Q(u, v); (u, v) \neq (0, 0)$$

- This array contains **mostly zeros**, especially at high frequencies. So, rearrange the array into a 1-D vector using **zig-zag ordering**:

$$\begin{bmatrix} 0 & 1 & 5 & 6 & 14 & 15 & 27 & 28 \\ 2 & 4 & 7 & 13 & 16 & 26 & 29 & 42 \\ 3 & 8 & 12 & 17 & 25 & 39 & 41 & 43 \\ 9 & 11 & 18 & 24 & 31 & 40 & 44 & 53 \\ 10 & 19 & 23 & 32 & 39 & 45 & 52 & 54 \\ 20 & 22 & 33 & 38 & 46 & 51 & 55 & 60 \\ 21 & 34 & 37 & 47 & 50 & 56 & 59 & 61 \\ 35 & 36 & 48 & 49 & 57 & 58 & 62 & 63 \end{bmatrix}$$



77

## Data Re-Arrangement Example

- Reordered quantized block from the previous example:

$$[ 79 \quad 0 \quad -2 \quad -1 \quad -1 \quad -1 \quad 0 \quad 0 \quad -1 \quad (55 \text{ 0's}) ]$$

- **Many zeros.**

78

## Handling of DC Coefficients

- Apply simple DPCM to the DC values  $\tilde{I}_k^Q(0, 0)$  between adjacent blocks to reduce the DC entropy.
- The difference between the current-block DC value and the left-adjacent-block DC value is found:
 
$$e(k) = \tilde{I}_k^Q(0, 0) - \tilde{I}_{k-1}^Q(0, 0)$$
- The differences  $e(k)$  are losslessly coded by a **lossless JPEG Huffman coder** (with agreed-upon table).
- The first column of DC values must be retained to allow reconstruction.

79

## Huffman Coding of AC Values

SSSS	AC coefficients
0	0
1	+1, 1
2	-3, -2, 2, 3
3	-7, ..., -4, 4, ..., 7
4	-15, ..., -8, 8, ..., 15
5	-31, ..., -16, 16, ..., 31
6	-63, ..., -32, 32, ..., 63
7	-127, ..., -64, 64, ..., 127
8	-255, ..., -128, 128, ..., 255
9	-511, ..., -256, 256, ..., 511
10	-1023, ..., -512, 512, ..., 1023
11	-2047, ..., -1024, 1024, ..., 2047
12	-4095, ..., -2048, 2048, ..., 4095
13	-8191, ..., -4096, 4096, ..., 8191
14	-16383, ..., -8192, 8192, ..., 16383
15	-32767, ..., -16384, 16384, ..., 32767

- The AC pairs (Skip, Value) are coded using another JPEG Huffman coder.
- A Huffman code represents which category (SSSS) the AC pair falls in.
- RRRR bits are used to represent the runlength of zeros, SSSS additional bits are used to represent the AC magnitude and sign.
- See Pennebaker and Mitchell for details!

82

## Huffman Coding of DC Differences

SSSS	DPCM difference
0	0
1	+1, 1
2	-3, -2, 2, 3
3	-7, ..., -4, 4, ..., 7
4	-15, ..., -8, 8, ..., 15
5	-31, ..., -16, 16, ..., 31
6	-63, ..., -32, 32, ..., 63
7	-127, ..., -64, 64, ..., 127
8	-255, ..., -128, 128, ..., 255
9	-511, ..., -256, 256, ..., 511
10	-1023, ..., -512, 512, ..., 1023
11	-2047, ..., -1024, 1024, ..., 2047
12	-4095, ..., -2048, 2048, ..., 4095
13	-8191, ..., -4096, 4096, ..., 8191
14	-16383, ..., -8192, 8192, ..., 16383
15	-32767, ..., -16384, 16384, ..., 32767
16	32768

- A Huffman Code is generated and stored/sent indicating which **category** (SSSS) the DC value falls in (See Pennebaker and Mitchell for the Huffman codes).
- SSSS is also the # bits allocated to code the DPCM differences and sign

80

## Run-Length Coding of AC Coefficients

- The AC vector contains **many** zeros.
- By using RLC considerable compression is attained.
- The AC vector is converted into 2-tuples (Skip, Value), where
  - Skip = number of zeros preceding a non-zero value
  - Value = the following non-zero value.
- When final non-zero value encountered, send (0,0) as **end-of-block**.

81

## JPEG Decoding

- **Decoding** is accomplished by reversing the Huffman coding, RLC and DPCM coding to recreate  $\tilde{I}_k^Q$
- Then **multiply by the normalization array** to create the lossy DCT

$$\tilde{I}_k^{\text{lossy}} = \mathbf{Q} \otimes \tilde{I}_k^Q$$

- The decoded image block is the IDCT of the result
 
$$I_k^{\text{lossy}} = \text{IDCT} \left[ \tilde{I}_k^{\text{lossy}} \right]$$

83

## JPEG Decoding

- The **overall compressed image** is recreated by putting together the compressed 8 x 8 pieces:

$$I^{\text{lossy}} = \left[ I_k^{\text{lossy}} \right]$$

- The compressions that can be attained range over:
  - 8 : 1 (very high quality)
  - 16 : 1 (good quality)
  - 32 : 1 (poor quality for most applications)

84

## JPEG Examples

Original  
512 x 512  
*Barbara*



85



64 : 1



16 : 1

Original  
512 x 512  
*Boats*



89



32 : 1



16 : 1

87

90



32 : 1

91



64 : 1

92

[DEMO](#)

## JPEG Color Quantization Table

$$\mathbf{Q} = \begin{bmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{bmatrix}$$

- Sparser sampling and more severe quantization of the chrominance values is acceptable.
- Color is primarily a **regional attribute** and contains less detail information.

94

## Wavelet Image Compression

### COLOR JPEG

- Basically, RGB image is converted to YUV or YCrCb image (Y = intensity, UV or CrCb = chrominances – see Module 10)
- Each channel Y, U, V is **separately** JPEG-coded (no cross-channel coding)
- Chrominance images may be **subsampled** first
- Different quantization / normalization table may be used

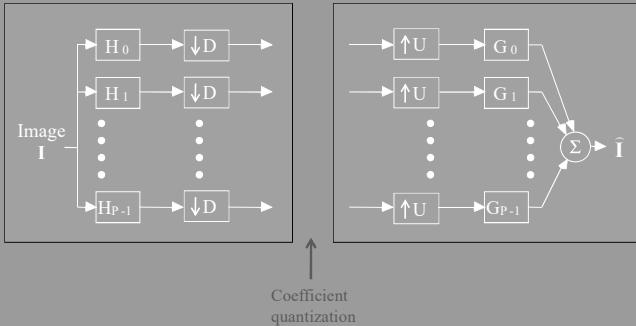
93

### Wavelet Image Coding

- **Idea:** Compress images in the wavelet domain.
- Image blocks are not used. Hence: **no blocking artifacts!**
- Forms the basis of the **JPEG2000 standard**.
- Also requires **perfect reconstruction filter banks**.

96

## Compression Via DWT

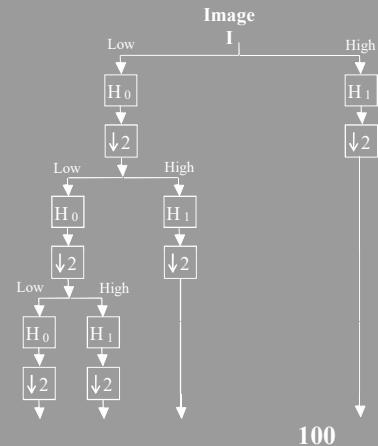


Filters  $H_p$  form a **orthogonal or bi-orthogonal basis** with reconstruction filters  $G_p$ .

97

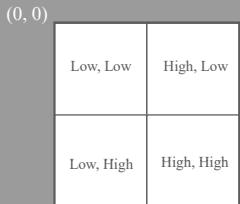
## Wavelet Filter Hierarchy

- Filter outputs are wavelet coefficients.
- Subsampling yields NM non-redundant coefficients. The image can be exactly reconstructed from them.
- Idea: Wavelet coefficients quantized (like JPEG): higher frequency coefficients quantized more severely.
- The **JPEG2000 Standard** is wavelet based.



## Wavelet Decompositions

- Recall that wavelet filters **separate or decompose** the image into high- and low-frequency bands, called **subbands**. Each frequency band can be coded separately.



- These are called **quadrature filters**.

98

## Bi-Orthogonal Wavelets

- Recall the **perfect reconstruction condition**:  
 $\tilde{H}_0(\omega)\tilde{G}_0(\omega) + \tilde{H}_1(\omega)\tilde{G}_1(\omega) = 2$  for all  $0 \leq \omega \leq \pi$  (DSFT)  
hence also  $\tilde{H}_0(u)\tilde{G}_0(u) + \tilde{H}_1(u)\tilde{G}_1(u) = 2$  for all  $0 \leq u \leq N-1$  (DFT)  
and  
 $\tilde{H}_0(\omega+\pi)\tilde{G}_0(\omega) + \tilde{H}_1(\omega+\pi)\tilde{G}_1(\omega) = 0$  for all  $0 \leq \omega \leq \pi$  (DSFT)  
hence also  $\tilde{H}_0(u+N/2)\tilde{G}_0(u) + \tilde{H}_1(u+N/2)\tilde{G}_1(u) = 0$  for all  $0 \leq u \leq N-1$  (DFT)
- The filters  $H_0, H_1, G_0, G_1$  can be either **orthogonal quadrature mirror filters**, or they can be bi-orthogonal

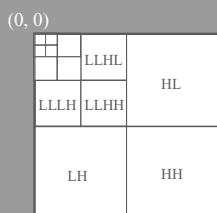
$$\sum_{i=1}^{N-1} H_n(i) G_m(i) = \begin{cases} 1; & m = n \\ 0; & m \neq n \end{cases}$$

for all  $m, n = 0, 1$ .

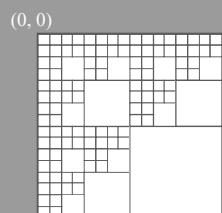
101

## Wavelet Decompositions

- Typically frequency division is done iteratively on the **lowest-frequency bands**:



"Pyramid" Wavelet Transform



"Tree-Structured" Wavelet Transform

99

## Bi-Orthogonal Wavelets

- Bi-orthogonal wavelets are useful since they can be made even-symmetric.
- This is important because much better results can be obtained in **image compression**.
- Even though they do not have as **good energy compaction** as orthogonal wavelets.

102

## Bi-Orthogonal Wavelets

- Wavelet based image compression usually operates on large image blocks (e.g., 64x64 in JPEG2000)
- It is much more efficient to realize block DWTs using cyclic convolutions (shorter).
- As with the DCT, **symmetric extended signals** (without discontinuities) give better results than **periodic extended**.
- Symmetric wavelets give better approximations to symmetric extended signals.**
- There are standard bi-orthogonal wavelets.

103

## Daubechies 9/7 Bi-Orthogonal Wavelets

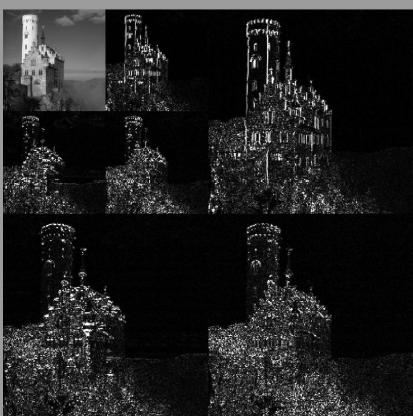
Analysis LP	Analysis HP	Synthesis LP	Synthesis HP
$H_0(0) = 0.026749$	$H_0(0) = 0$	$G_0(0) = 0.026749$	$G_0(0) = 0.026749$
$H_0(1) = -0.016864$	$H_1(1) = 0.091272$	$G_0(1) = -0.091272$	$G_1(1) = 0.016864$
$H_0(2) = -0.078223$	$H_1(2) = -0.05754$	$G_0(2) = -0.05754$	$G_1(2) = -0.078223$
$H_0(3) = 0.266864$	$H_1(3) = 0.59127$	$G_0(3) = 0.59127$	$G_1(3) = -0.266864$
$H_0(4) = 0.602949$	$H_1(4) = 1.115087$	$G_0(4) = 1.115087$	$G_1(4) = 0.602949$
$H_0(5) = 0.266864$	$H_1(5) = -0.59127$	$G_0(5) = 0.59127$	$G_1(5) = -0.266864$
$H_0(6) = -0.078223$	$H_1(6) = -0.05754$	$G_0(6) = -0.05754$	$G_1(6) = -0.078223$
$H_0(7) = -0.016864$	$H_1(7) = 0.091272$	$G_0(7) = -0.091272$	$G_1(7) = 0.016864$
$H_0(8) = 0.026749$	$H_1(8) = 0$	$G_0(8) = 0$	$G_1(8) = 0.026749$



These are used in the JPEG2000 standard.

104

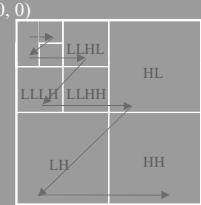
## D9/7 DWT of an Image



105

## Zero-Tree Encoding Concept

- There are many proposed wavelet-based image compression techniques. Many are complex.
- A simple concept is **zero-tree encoding**: Similar to JPEG, scan from lower to higher frequencies and **run-length code** the zero coefficients.



Zig-zag scanning to find zeroed coefficients

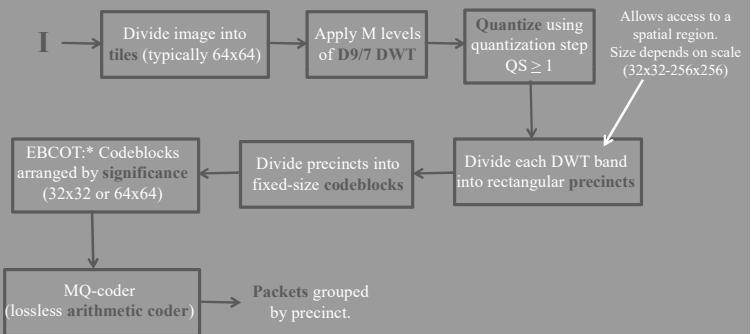
106

## JPEG2000

- Too complex to go into depth, and not widely-accepted enough to spend much time on.
- Advantages:**
  - Better compression at low bitrates
  - Scalable (can truncate bitstream at any point)
  - No Blocking artifacts
- Disadvantages:**
  - Complex encoders/decoders
  - Only slightly better compression at high bitrates
  - Ringing artifacts

107

## JPEG2000

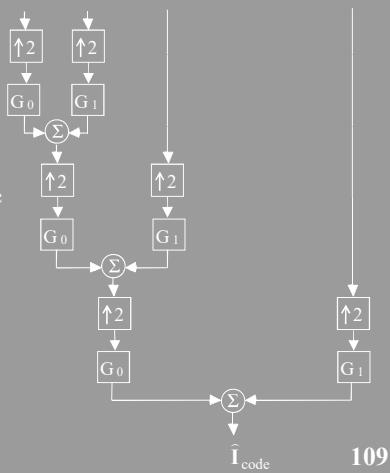


\*Embedded Block Coding with Optimal Truncation

108

## Wavelet Decoding

- Decoding requires
  - Decompression of the wavelet coefficients
  - Reconstructing the image from the decoded coefficients:



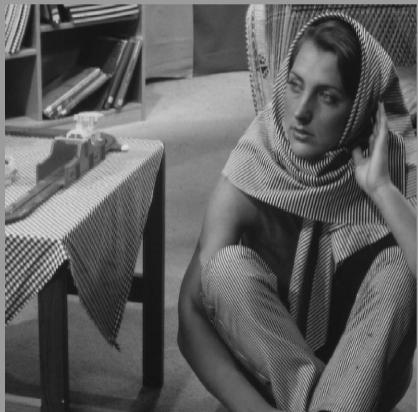
109



32 : 1

## JPEG2000 Wavelet Compression Example

Original  
512 x 512  
*Barbara*



110



64 : 1



16 : 1

111



128 : 1

114

Original  
512 x 512  
Boats



115

64 : 1



118



16 : 1

WAV 16:1

116

128 : 1



119



32 : 1

WAV 32:1

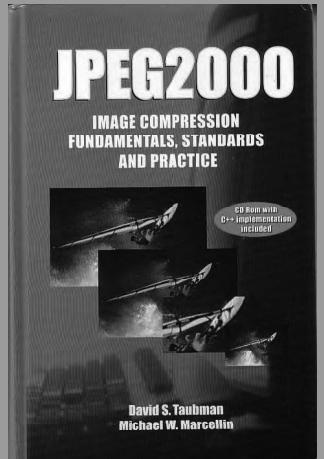
117

120

## Comments on Wavelet Compression

- This has been a **watered-down** exposition of wavelet coding!
- **Reason:** there has not yet been much adoption of JPEG2000.
- Why? It is **vastly** more complex than “old” JPEG. A consideration in an era of **cheap memory**.
- There are questions of **propriety**.
- Its performance is primarily better in the **high compression regime**.

- If interested....



121

## Comments

- So far we have been **processing images** – now we will try to **understand them** – **image analysis** ... onward to **Module 7**.

122