# Lecture 12:
# Lossy Image Compression and Scalar Quantization

Thinh Nguyen
Oregon State University

# Lossy Image Compression Techniques

- Scalar quantization (SQ)
- Vector quantization (VQ)
- Discrete Cosine Transform (DCT) Compression:
  - JPEG
- Wavelet Compressions:
  - SPIHT
  - EBCOT

# Lossy Image Compression Techniques

SPIHT

(Set Partition Hierarchy Tree)

Original

32:1 compression

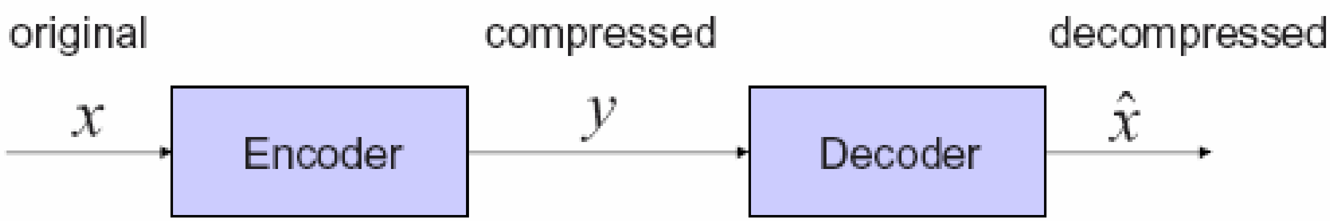
JPEG

# Images and the Eye

- Images are meant to be viewed by the human eye.

- The eye is very good at “interpolation,” that is, the eye can tolerate some distortion. So lossy compression is not necessarily bad.

# Distortion




- Lossy compression: $x \neq \hat{x}$
- Measure of distortion is commonly mean squared error (MSE). Assume $x$ has $n$ real components (pixels).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2$$

# Distortion

- Peak Signal to Noise Ratio (PSNR) is the tandard way to measure fidelity.

$$PSNR = 10\log_{10}\left(\frac{m^2}{MSE}\right)$$

where $m$ is the maximum value of a pixel possible.
For gray scale images (8 bits per pixel) $m = 255$.

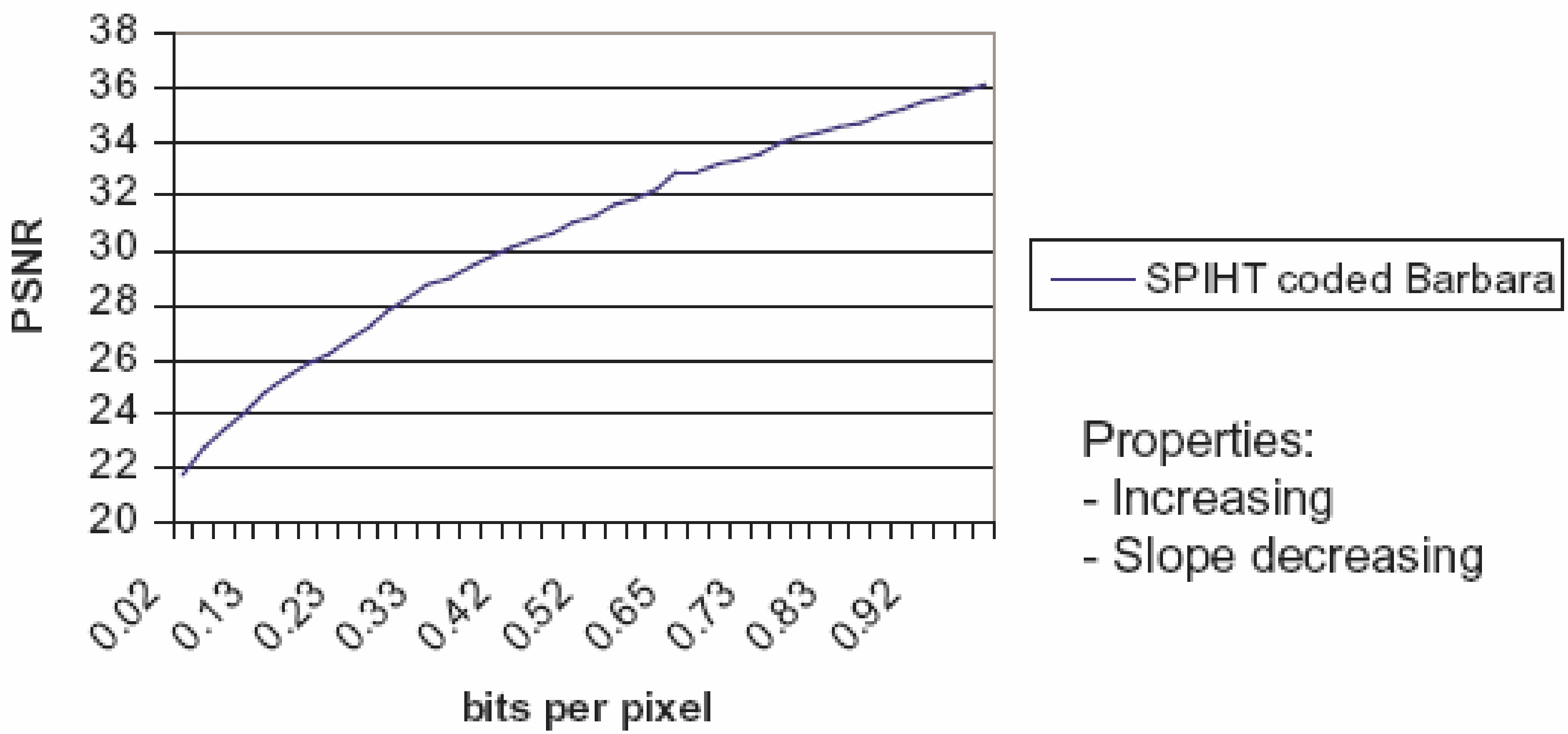
- PSNR is measured in decibels (dB):
- 0.5 to 1 dB is said to be a perceptible difference.
- Decent images *start* at about 25-30 dB.
- 35-40 dB might be indistinguishable from the original

# PSNR is not everything!

PSNR = 25.8 dB

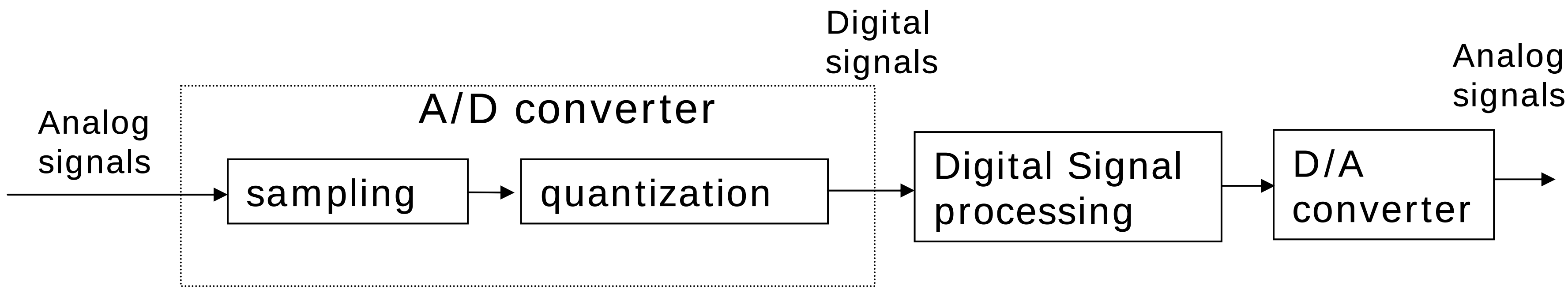
PSNR = 25.8 dB

# Distortion vs. Compression



# Quantization Problem

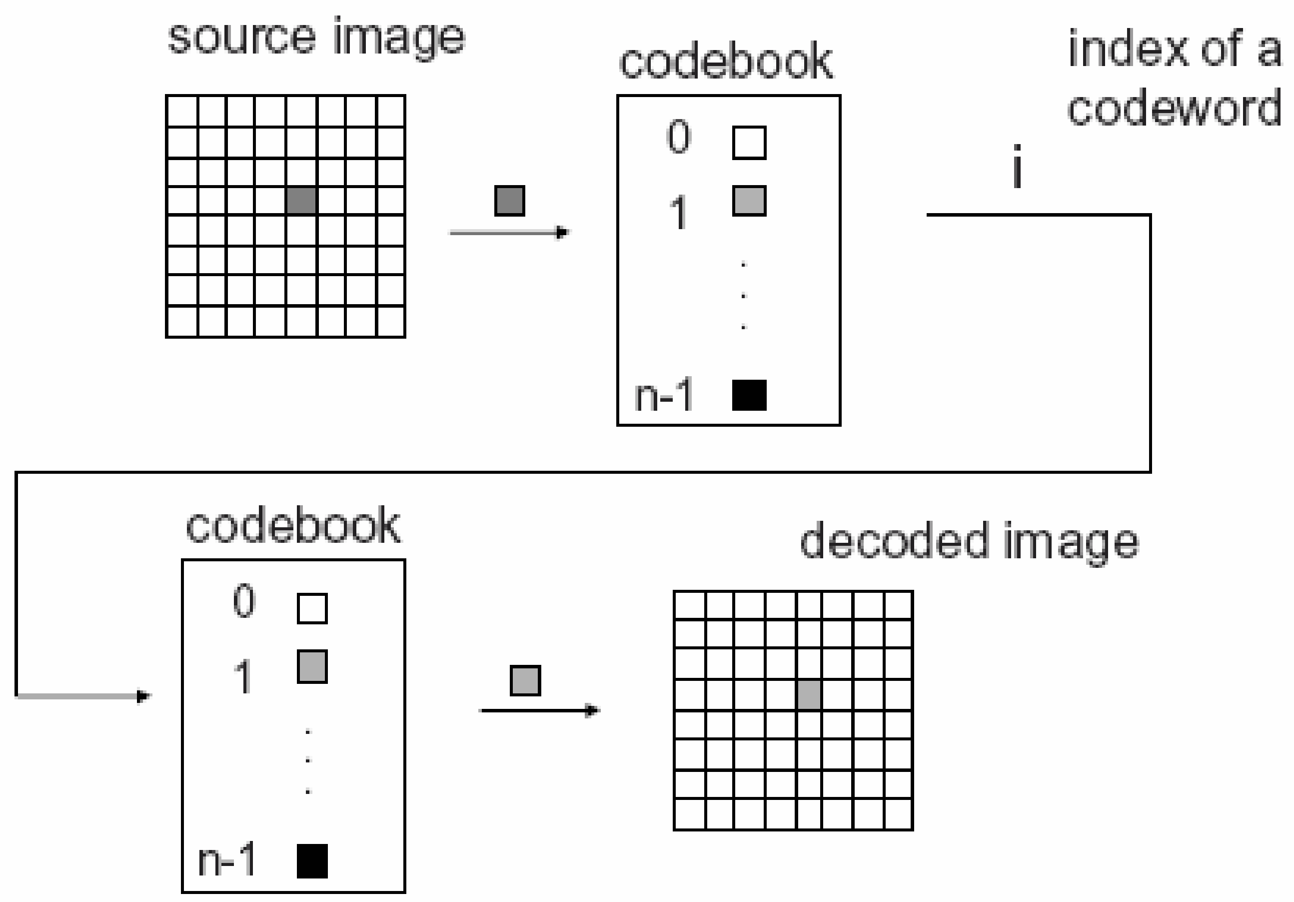
- Real-world signals are continuous!
- Signal representation in computer is discrete with finite precision!
- Higher precision requires larger storage



# Scalar Quantization Problems

- Problem 1:
  - You’re given 16-bit integers (0-65545). Unfortunately, you only have space to store 8-bit integers (0-255).
  - Come up with a representation of those 16-bit integers that uses only 8 bits!
- Problem 2:
  - You have a string of those 8-bit integers that use your representation.
  - Recreate the 16-bit integers as best you can!

# Scalar Quantization



# Scalar Quantization Strategies

- Build a codebook with a training set, then always encode and decode with that fixed codebook.
  - Most common use of scalar quantization.
- Build a codebook for each image and transmit the codebook with the image.
- Training can be slow.

# Distortion from Scalar Quantization

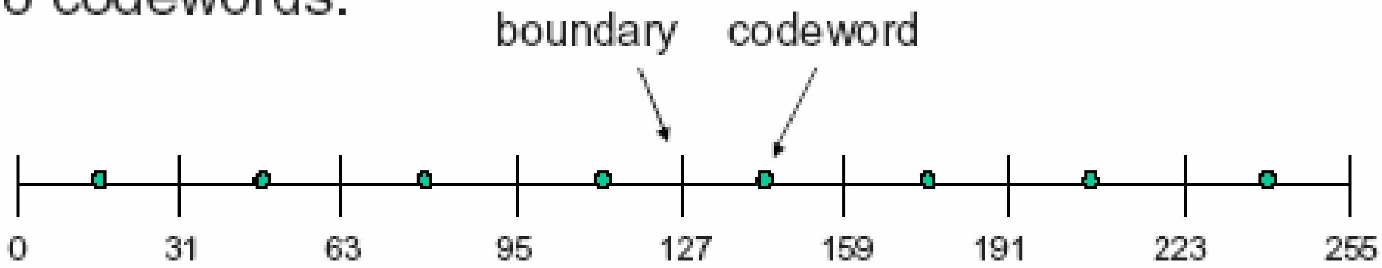
- Let the image be pixels $x_1, x_2, \ldots x_T$.
- Define index($x$) to be the index transmitted on input $x$.
- Define c($j$) to be the codeword indexed by $j$.

$$D = \sum_{i=1}^{T} (x_i - c(\text{index}(x_i))^2 \qquad \text{(Distortion)}$$

$$MSE = \frac{D}{T}$$

# Uniform Quantization Example

- 512 x 512 image with 8 bits per pixel.
- 8 codewords.




Codebook

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Codeword | 16 | 47 | 79 | 111 | 143 | 175 | 207 | 239 |

# Uni. Quant. Encoder and Decoder

Encoder

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| input | 0-31 | 32-63 | 64-95 | 96-127 | 128-159 | 160-191 | 192-223 | 224-255 |
| code | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |

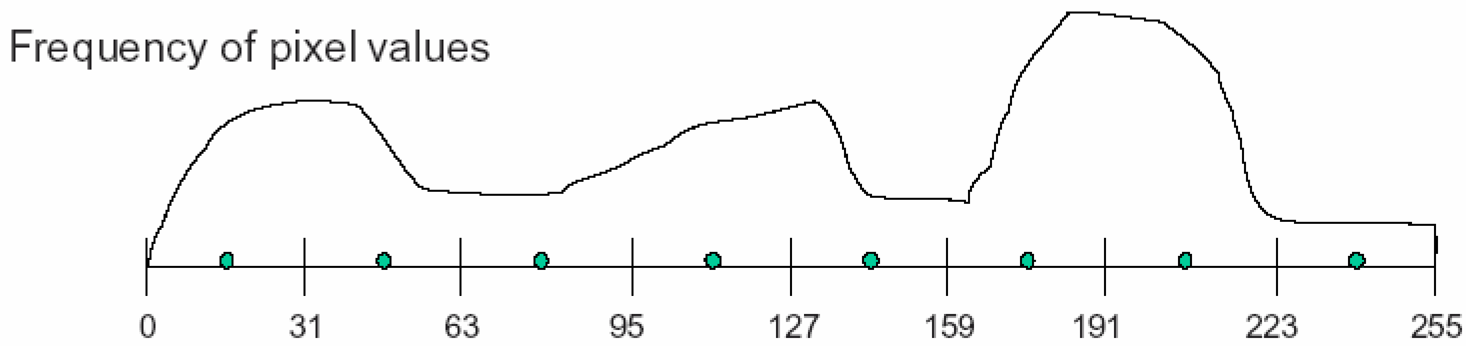
Decoder

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| code | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| output | 16 | 47 | 79 | 111 | 143 | 175 | 207 | 239 |

Bit rate = 3 bits per pixel

Compression ratio = 8/3 = 2.67

# Improve Bit Rate




$p_j$ = the probability that a pixel is coded to index $j$.
Potential average bit rate is entropy.

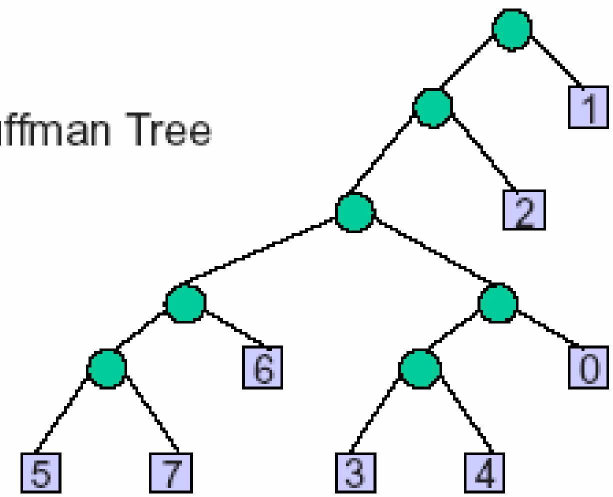
$$H = \sum_{j=0}^{7} p_j \log_2\left(\frac{1}{p_j}\right)$$

# Example

- 512 x 512 image = 262,144 pixels

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| input | 0-31 | 32-63 | 64-95 | 96-127 | 128-159 | 160-191 | 192-223 | 224-255 |
| frequency | 25,000 | 95,000 | 85,000 | 10,000 | 10,000 | 10,000 | 18,000 | 9,144 |

Huffman Tree

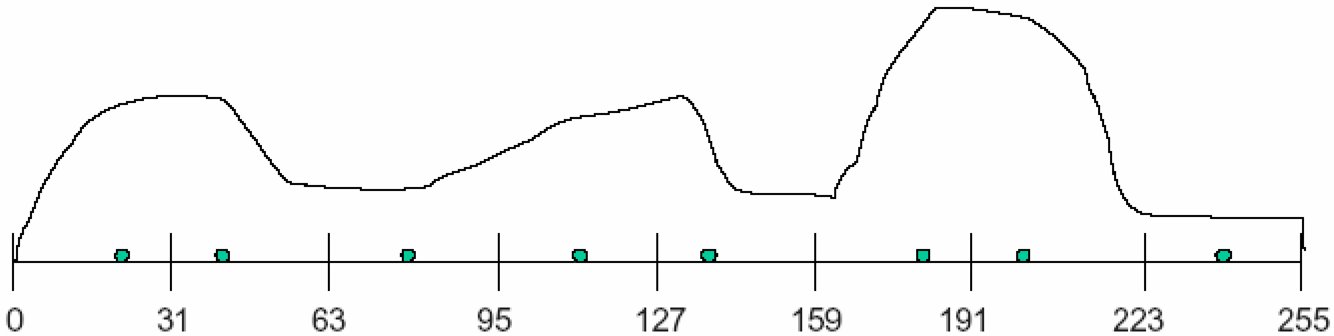



ABR= ( 95000 x 1+
85000 x 2 +
43000 x 4 +
39144 x 5)/262144
=2.41 bpp

Arithmetic coding should work better.

# Improve Distortion

- Choose the codeword as a weighted average (the *centroid*).




Let $p_x$ be the probability that a pixel has value $x$.
Let $[L_j,R_j)$ be the input interval for index $j$.
$c(j)$ is the codeword indexed by $j$.

$$c(j) = \text{round}\left( \sum_{L_j \le x < R_j} x \cdot p_x \right)$$

# Example

All pixels have the same index.

| pixel value | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|
| frequency | 100 | 100 | 100 | 40 | 30 | 20 | 10 | 0 |

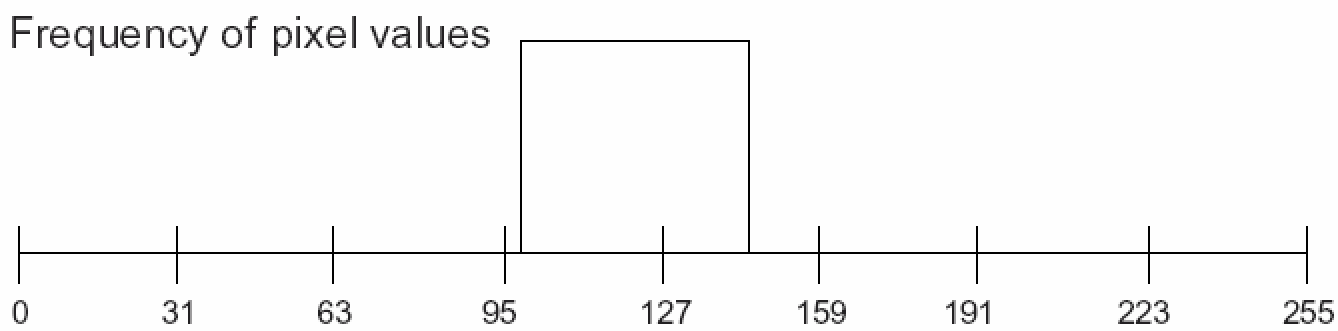
$$\text{New Codeword} = \text{round}\left(\frac{8\cdot 100 + 9\cdot 100 + 10\cdot 100 + 11\cdot 40 + 12\cdot 30 + 13\cdot 20 + 14\cdot 10 + 15\cdot 0}{400}\right) = 10$$

$$\text{Old Codeword} = 11$$

$$\text{New Distortion} = 140\cdot 1^2 + 130\cdot 2^2 + 20\cdot 3^2 + 10\cdot 4^2 = 10000$$

$$\text{Old Distortion} = 130\cdot 1^2 + 120\cdot 2^2 + 110\cdot 3^2 = 16000$$

# Extreme Case

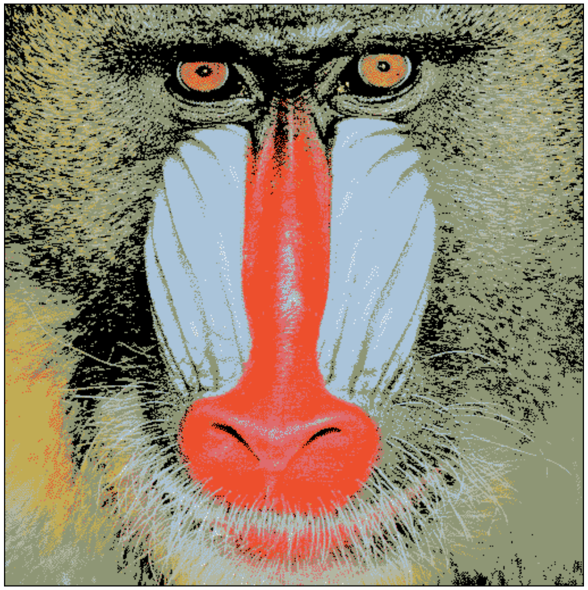



Only two codewords are ever used!!

# Quantization Examples: Mandrill


8 bit quantization


4 bit quantization


3 bit quantization

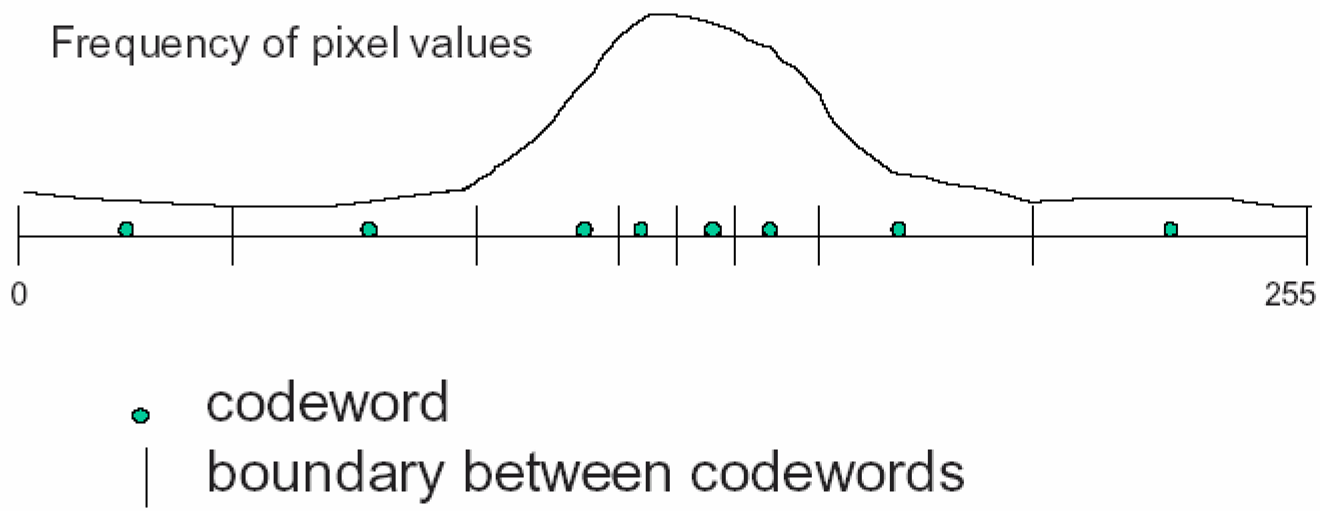
# Quantization Examples: Pepper

8 bit quantization

4 bit quantization

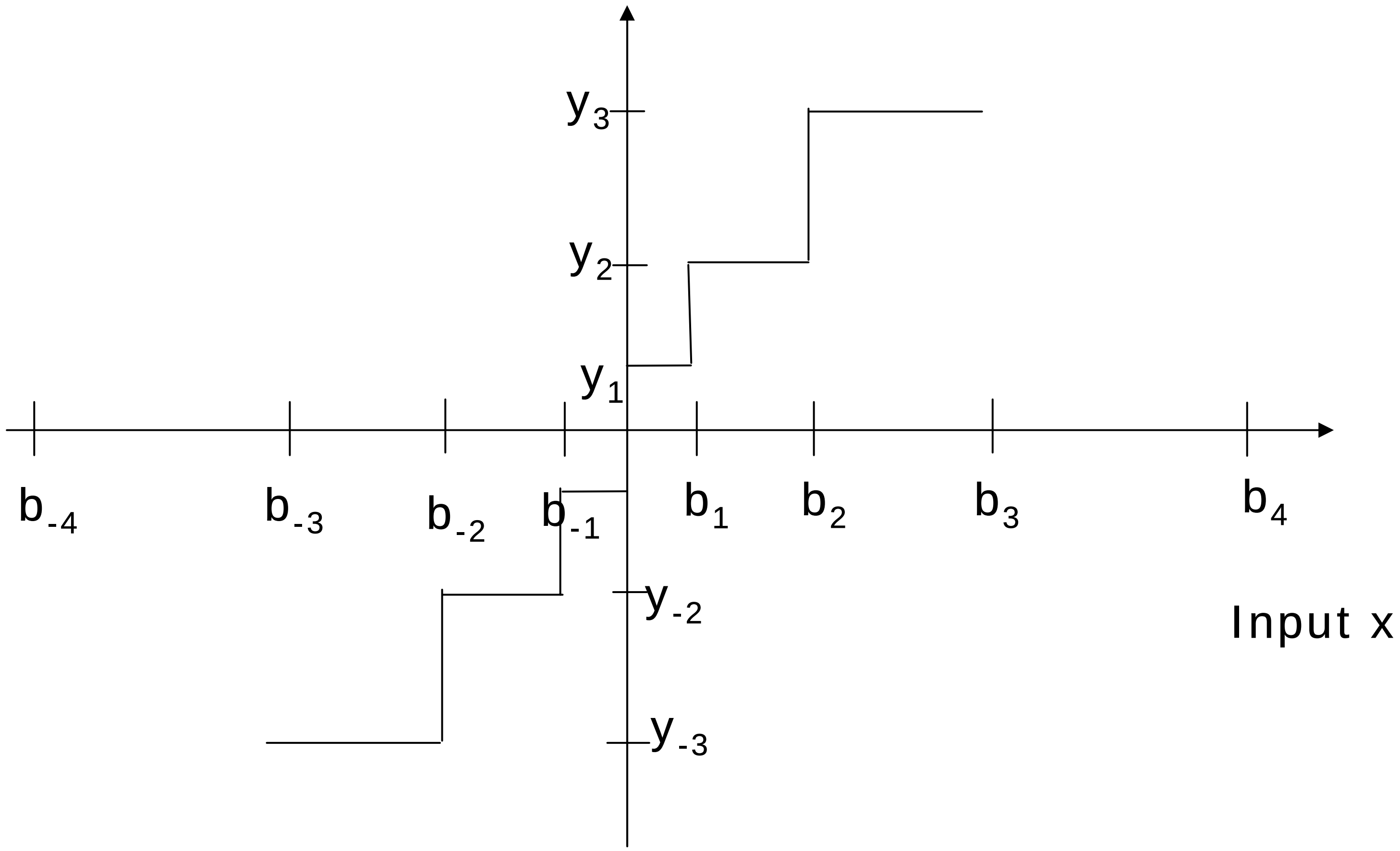
3 bit quantization

# Non-uniform scalar quantization



# Non-uniform scalar quantization

- Problem: Given M reconstruction levels, find the boundaries of these construction levels ($b_1, b_2, \ldots b_M$) and reconstruction levels ($y_1, y_2, \ldots y_M$) to minimize the distortion.



# Non-uniform scalar quantization

- LLoyd (1957) shows that the solutions $y_i$ and $b_i$ must satisfy the following 2 conditions:

$$y_j = \frac{\int_{b_{j-1}}^{b_j} x f(x)\,dx}{\int_{b_{j-1}}^{b_j} f(x)\,dx} \qquad b_j = \frac{y_{j+1} + y_j}{2}$$

$f(x)$ : is the probability density function of input x

# Non-uniform scalar quantization

- Proof: Take derivatives of with respect to $y_i$ and $b_i$ of MSE, setting the result to zero, and solve for $y_i$ and $b_i$

$$MSE = \sum_{i=1}^{M} \int_{b_{j-1}}^{b_j} (x - y_i)^2 f(x) dx$$

# Lloyd's Algorithm

- Lloyd (1957)
- Creates an optimized (but probably not optimal) codebook of size n.
- Let $p_x$ be the probability of pixel value x.
  - Probabilities is either known or might come from a training set.
- Given codewords c(0),c(1),...,c(n-1) and pixel x. Let index(x) be the index of the closest code word to x.
- Expected distortion is

$$D = \sum_x p_x (x - c(\mathrm{index}(x)))^2$$

- Goal of the Lloyd algorithm is to find the codewords that minimize distortion.
  - Lloyd finds a local minimum by an iteration process.

# Lloyd's Algorithm

Choose a small error tolerance $\varepsilon > 0$.
Choose start codewords c(0),c(1),...,c(n-1).
Compute X(j) := {x : x is a pixel value closest to c(j)}.
Compute distortion D for c(0),c(1),...,c(n-1).
Repeat:
  Compute new codewords:

$$c'(j) := \text{round}\Big(\sum_{x \in X(j)} x \cdot p_x\Big)$$

  Compute X'(j) = {x : x is a pixel value closest to c'(j)}.
  Compute distortion D' for c'(0),c'(1),...,c'(n-1).
  if $|(D - D')/D| < \varepsilon$ then quit,
  else c := c'; X := X', D := D'.
End{repeat}

# Example

Initially c(0) = 2 and c(1) = 5

| pixel value | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| frequency | 100 | 100 | 100 | 40 | 30 | 20 | 10 | 0 |

$$X(0) = [0,3],\ X(1) = [4,7]$$

$$D(0) = 140 \cdot 1^2 + 100 \cdot 2^2 = 540;\ D(1) = 40 \cdot 1^2 = 40$$

$$D = D(0) + D(1) = 580$$

$$c'(0) = \text{round}((100 \cdot 0 + 100 \cdot 1 + 100 \cdot 2 + 40 \cdot 3)/340) = 1$$

$$c'(1) = \text{round}((30 \cdot 4 + 20 \cdot 5 + 10 \cdot 6 + 0 \cdot 7)/60) = 5$$

# Example

| pixel value | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| frequency | 100 | 100 | 100 | 40 | 30 | 20 | 10 | 0 |

$$c'(0) = 1;\ c'(1) = 5$$

$$X'(0) = [0,2];\ X'(1) = [3,7]$$

$$D'(0) = 200 \cdot 1^2 = 200$$

$$D'(1) = 40 \cdot 1^2 + 40 \cdot 2^2 = 200$$

$$D' = D'(0) + D'(1) = 400$$

$$\left|(D - D')/D\right| = (580 - 400)/580 = .31$$

$$c := c';\ X := X';\ D := D'$$

# Example

| pixel value | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| frequency | 100 | 100 | 100 | 40 | 30 | 20 | 10 | 0 |

$$c(0) = 1; c(1) = 5$$

$$X(0) = [0,2]; X(1) = [3,7]$$

$$D = 400$$

$$c'(0) = \text{round}((100 \cdot 0 + 100 \cdot 1 + 100 \cdot 2)/300) = 1$$

$$c'(1) = \text{round}((40 \cdot 3 + 30 \cdot 4 + 20 \cdot 5 + 10 \cdot 6 + 0 \cdot 7)/100) = 4$$

# Example

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| pixel value | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| frequency | 100 | 100 | 100 | 40 | 30 | 20 | 10 | 0 |

$$c'(0) = 1;\ c'(1) = 4$$

$$X'(0) = [0,2];\ X'(1) = [3,7]$$

$$D'(0) = 200 \cdot 1^2 = 200$$

$$D'(1) = 60 \cdot 1^2 + 10 \cdot 2^2 = 100$$

$$D' = D'(0) + D'(1) = 300$$

$$|(D - D')/D| = (400 - 300)/400 = .17$$

$$c := c';\ X := X';\ D := D'$$

# Example

| pixel value | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| frequency | 100 | 100 | 100 | 40 | 30 | 20 | 10 | 0 |

$$c(0) = 1; c(1) = 4$$

$$X(0) = [0,2]; X(1) = [3,7]$$

$$D = 300$$

$$c'(0) = \text{round}((100 \cdot 0 + 100 \cdot 1 + 100 \cdot 2)/300) = 1$$

$$c'(1) = \text{round}((40 \cdot 3 + 30 \cdot 4 + 20 \cdot 5 + 10 \cdot 6 + 0 \cdot 7)/100) = 4$$

# Example

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| pixel value | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| frequency | 100 | 100 | 100 | 40 | 30 | 20 | 10 | 0 |

$c'(0) = 1; c'(1) = 4$

$X'(0) = [0,2]; X'(1) = [3,7]$

$D'(0) = 200 \cdot 1^2 = 200$

$D'(1) = 60 \cdot 1^2 + 10 \cdot 2^2 = 100$

$D' = D'(0) + D'(1) = 300$

$|(D - D')/D| = (300 - 300)/300 = 0$

Exit with codeword $c(0) = 1$ and $c(1) = 4$.

# Scalar Quantization Notes

- Useful for analog to digital conversion.
- With entropy coding, it yields good lossy compression.
- Lloyd algorithm works very well in practice, but can take many iterations.
  - For *n* codewords should use about 20*n* size
  - representative training set.
  - Imagine 1024 codewords.