National Tsing Hua University, Hsinchu, Taiwan

CS 5263: Wireless Multimedia Networking Technologies and Applications

Digital Image

Instructor: Cheng-Hsin Hsu

Some figures in these slides are taken from the online version of the Burg's book: The Science of Digital Media

Analog & Discrete Phenomena

- **Analog: continuous phenomenon, between any two points there exist infinite number of points**
  - **Most natural phenomena**
- **Discrete: points (either in time or space) are clearly separated**
- **Computers work with discrete values ➔ analog-to-digital conversion**
- **Digital media:**
  - **Better quality, less susceptible to noise**
  - **More compact to store and transmit (high compression ratios)**

Analog-to-Digital Conversion: Two Steps

- **Sampling:**
  - **choose discrete points at which we measure (sample) the continuous signal**
  - **Sampling rate is important to recreate the original signal**
- **Quantization:**
  - **Represent each sample using a fixed number of bits**
  - **Bit depth (or sample size) specifies the precision to represent a value**

Sampling

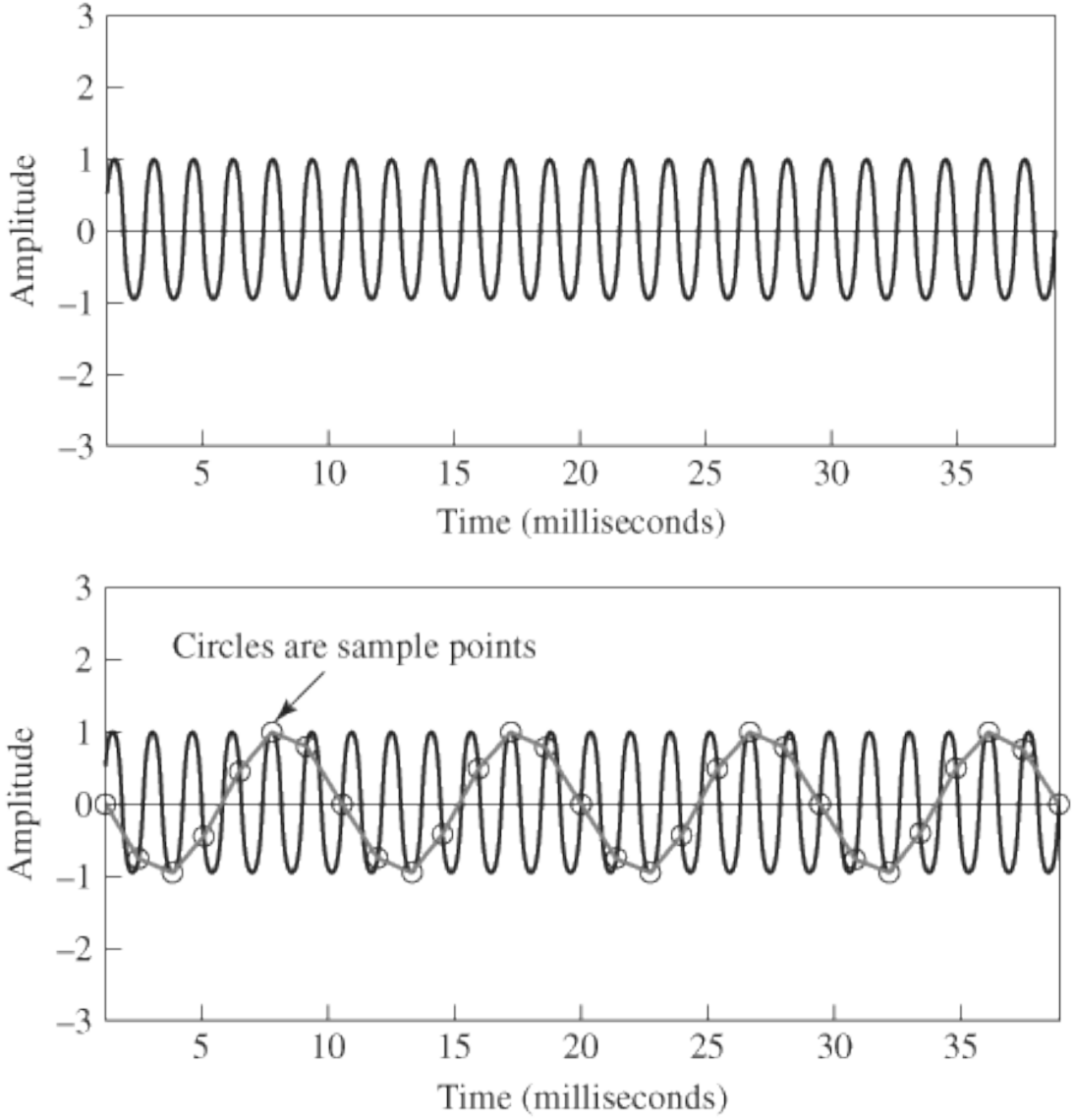
- **Nyquist frequency**
  - **The minimum sampling rate to reconstruct the original signal: $r = 2f$**
  - **f is the frequency of the signal**

- **Under sampling can produce distorted/different signals (aliasing)**

Under sampling: Example

- $f = 637$ Hz

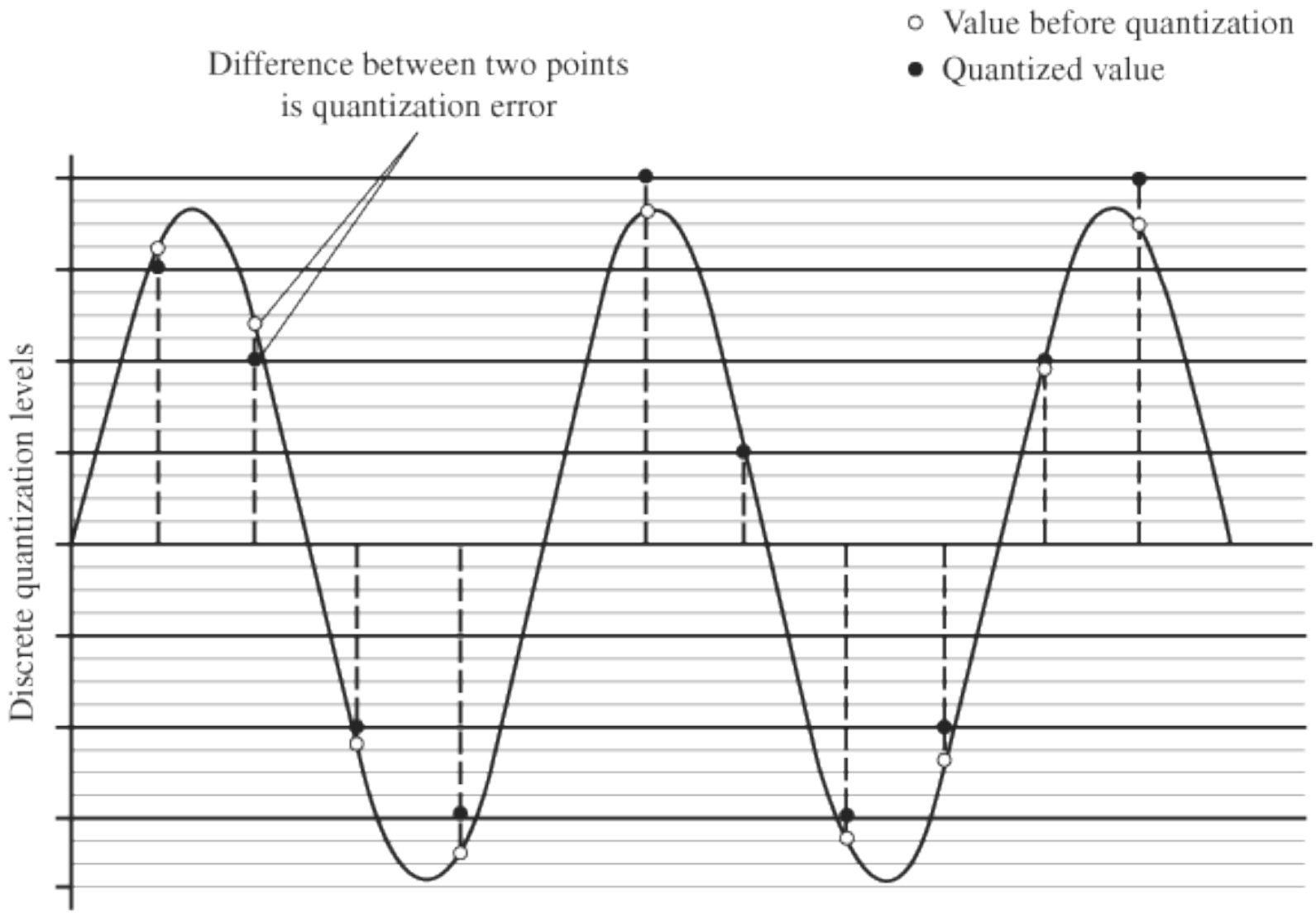
- Sampling at $770 (< 2f)$ produces a different wave



Quantization

- n bits to represent a digital sample ➔ max number of different levels is $m = 2^n$
- ➔ real, continuous, sample values are rounded (approximated) to the nearest levels
- ➔ Some information (precision) could be lost

Quantization: Example



Quantization: Error

- **Signal-to-Quantization-Noise Ratio (SQNR)**
  - **Measures amount of error introduced by quantization**
  - **~ max sample value / max quantization error**
  - **Measured in decibel (dB)**

- **Max quantization error is ½ of quantization step**
- **Quantization values range from -2^{n-1} to $2^{n-1}-1$**
- **➔ Max (absolute) quantization value is 2^{n-1} ➔**

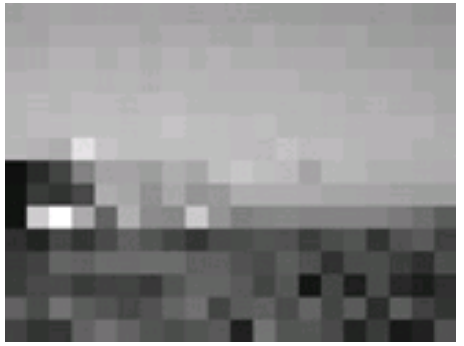
$$SQNR = 20\log_{10}\left(\frac{2^{n-1}}{1/2}\right) = 20\log_{10}\left(2^{n}\right)$$

Methods of Creating Digital Images

- **Bitmap images (our focus)**
  - Created pixel by pixel (pixel = picture element)
  - Commonly used in digital cameras, scanners, …
  - Good for photographic images, where colors change frequently and subtly
- **Vector graphics**
  - Objects are specified by math equations and colors
  - Useful for clearly delineated shapes and colors, e.g., in cartoon or poster pictures
  - Used in Adobe Illustrator, Corel Draw, Visio, Omnigraffle, and Xfig
- **Procedural modeling**
  - Computer algorithms employed to generate complex patterns, shapes, and colors

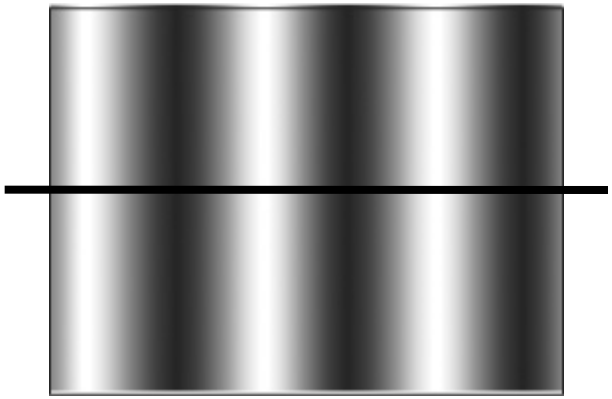
Bitmap Images: Sampling & Quantization

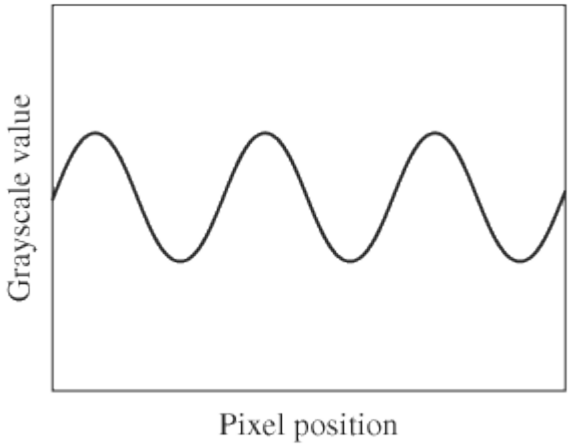
- **Consider natural scene, shot by digital camera**
- **Sampling: #pixels in *x*, *y***
  - **Camera takes *x y* samples**
  - **E.g., 1600 x 1200, limited by camera specs**
  - **Pixel: small square, its value = average color in square**
  - **Under sampling ➔ lack of details : 15 x 20**
- **Quantization:**
  - **Each pixel has 3 colors R, G, B**
  - **Bit depth: #bits for each of R, G, B**
  - **Typical: 8 bits each ➔ 2^{24} ~ 16+ m colors**
  - **low bit depth ➔ patches of colors: 12 colors**



Representing Images as Waveforms

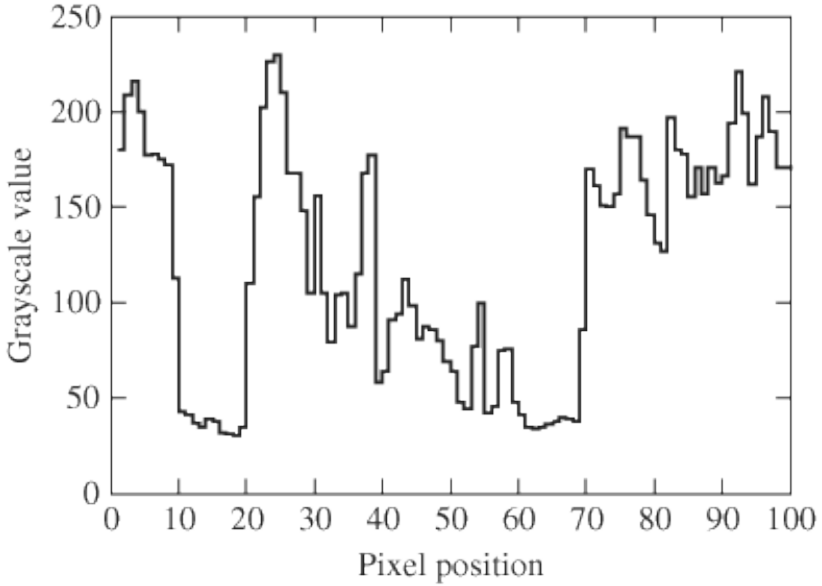
- Consider one row in a grayscale image
  - Pixel values vary
- Plot pixel value as function of position:
  - $z = f(x,y)$
- Grayscale
  - 0: black, 255: white
- Same waveform analysis applies to R, G, B color components





Representing Images as Waveforms

- Complex images ➔ complex waveforms
- Sudden changes in pixel values (color)➔ high frequency components in the waveform (signal)
- How do we analyze such complex signals?
- Using Fourier Theory



Fourier Analysis

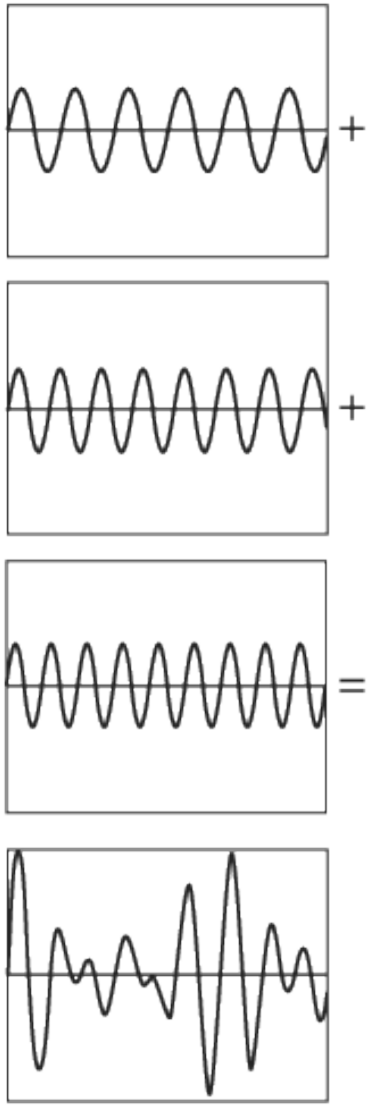
- **Fourier showed that any periodic signal can be decomposed into an infinite sum of sinusoidal waveforms**

$$f(x) = \sum_{n=0}^{\infty} a_n \cos(n w x)$$

- **Sinusoidal waves = frequency components**
- **Coefficients = weights of different components**
- **This analysis allows us to**
  - **determine the signal's frequency components**
  - **store complex signals in digital form**
  - **filter out unwanted/unimportant components**
    - **i.e., components that cannot be seen or heard by humans**

Fourier Analysis: Sound Wave

- **Signal that has only three components**
  - **All other coefficients are 0**



Frequency Components of Images

- **Images have discrete points ➔ Discrete Cosine Transform (DCT)**
- **Consider one row of *M* pixels in an image**
  - **Can be represented as sum of *M* weighted cosine functions (called basis functions):**

$$f(r) = \sum_{u=0}^{M-1} \frac{\sqrt{2}\,C(u)}{\sqrt{M}}\, F(u)\, \cos\left(\frac{(2r+1)u\pi}{2M}\right) \quad \text{for} \quad 0 \le r < M$$

$$\text{where } C(u) = \frac{\sqrt{2}}{2} \text{ if } u = 0 \text{ otherwise } C(u) = 1$$

Simple Example

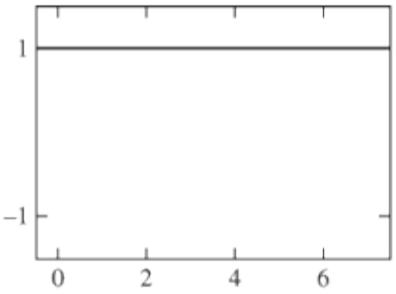
- 8 pixels with values:
  [0, 0, 0, 153, 255, 255, 220, 220]

- We compute weight for each of the eight basis functions

Example (cont'd)

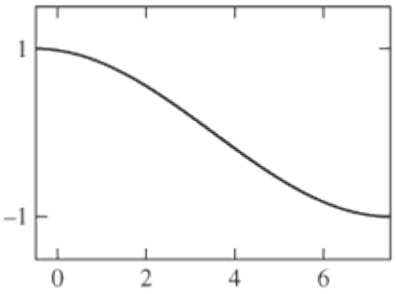
- **Basis function 0**
  Direct Current (DC) component
  no change in color

$$\cos\left(\frac{(2r+1)0\pi}{16}\right) = \cos(0)$$



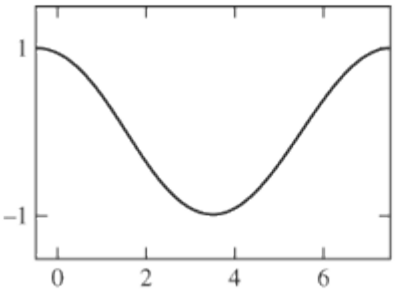
- **Basis function 1**
  low frequency
  slow change in color

$$\cos\left(\frac{(2r+1)\pi}{16}\right)$$



- **Basis function 2**
  bit faster change in color

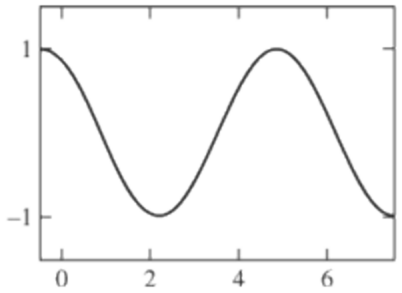
$$\cos\left(\frac{(2r+1)2\pi}{16}\right)$$



Example (cont'd)

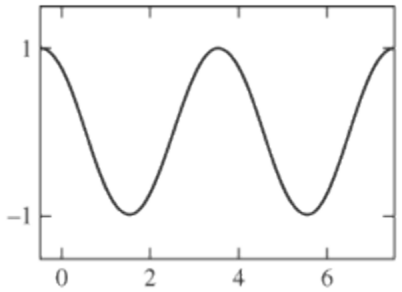
- **Basis function 3**

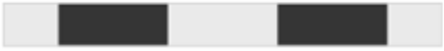
$$\cos\left(\frac{(2r+1)3\pi}{16}\right)$$



- **Basis function 4**

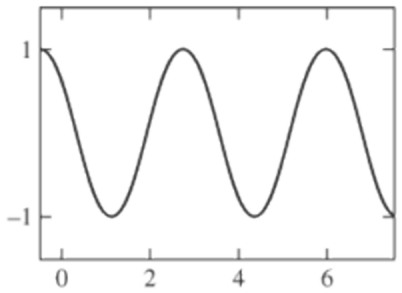
$$\cos\left(\frac{(2r+1)4\pi}{16}\right)$$





- **Basis function 5**

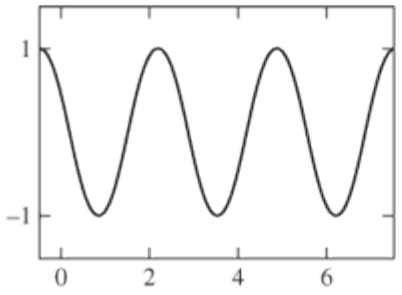
$$\cos\left(\frac{(2r+1)5\pi}{16}\right)$$



Example (cont'd)

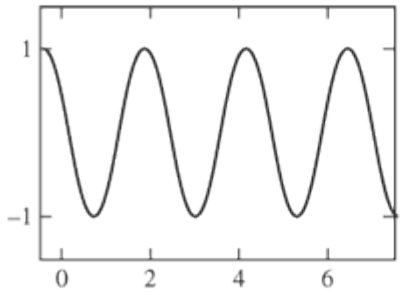
- **Basis function 6**

$$\cos\left(\frac{(2r+1)6\pi}{16}\right)$$



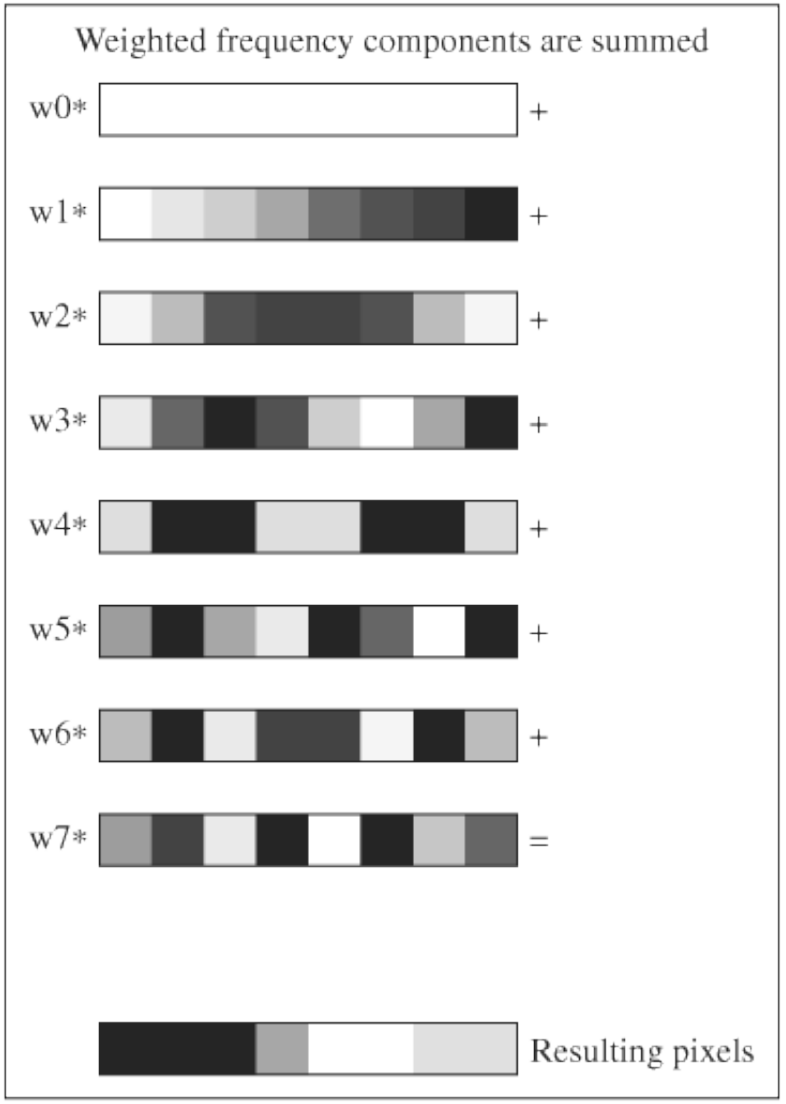
- **Basis function 7**
  Highest frequency component
  Notice the rapid change in pixel value (color)

$$\cos\left(\frac{(2r+1)7\pi}{16}\right)$$



Example (cont'd)

- **When all components are added up with their weights, we get the original row of pixels**



Two Dimensional DCT

- **We can compute the coefficients (weights) for an *M* x *N* image from:**

$$F(r,s) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} \frac{2C(u)C(v)}{\sqrt{MN}} f(u,v) \cos\left(\frac{(2r+1)u\pi}{2M}\right)\cos\left(\frac{(2s+1)v\pi}{2N}\right)$$

for $0 \le r < M, \quad 0 \le s < N$

where $C(\delta) = \frac{\sqrt{2}}{2}$ if $\delta = 0$ otherwise $C(\delta) = 1$

- **This is called 2D discrete cosine transform**
- **The inverse also exists: swap *f(.,.)* with *F(.,.)***

DCT: Notes

- DCT takes bitmap image (matrix of colors) and returns frequency components (matrix of DCT coefficients)
- In other words, DCT transforms signal from spatial domain to frequency domain
- DCT usually applied on blocks of 8 x 8 pixels
  - For efficient computation of coefficients

DCT: Notes

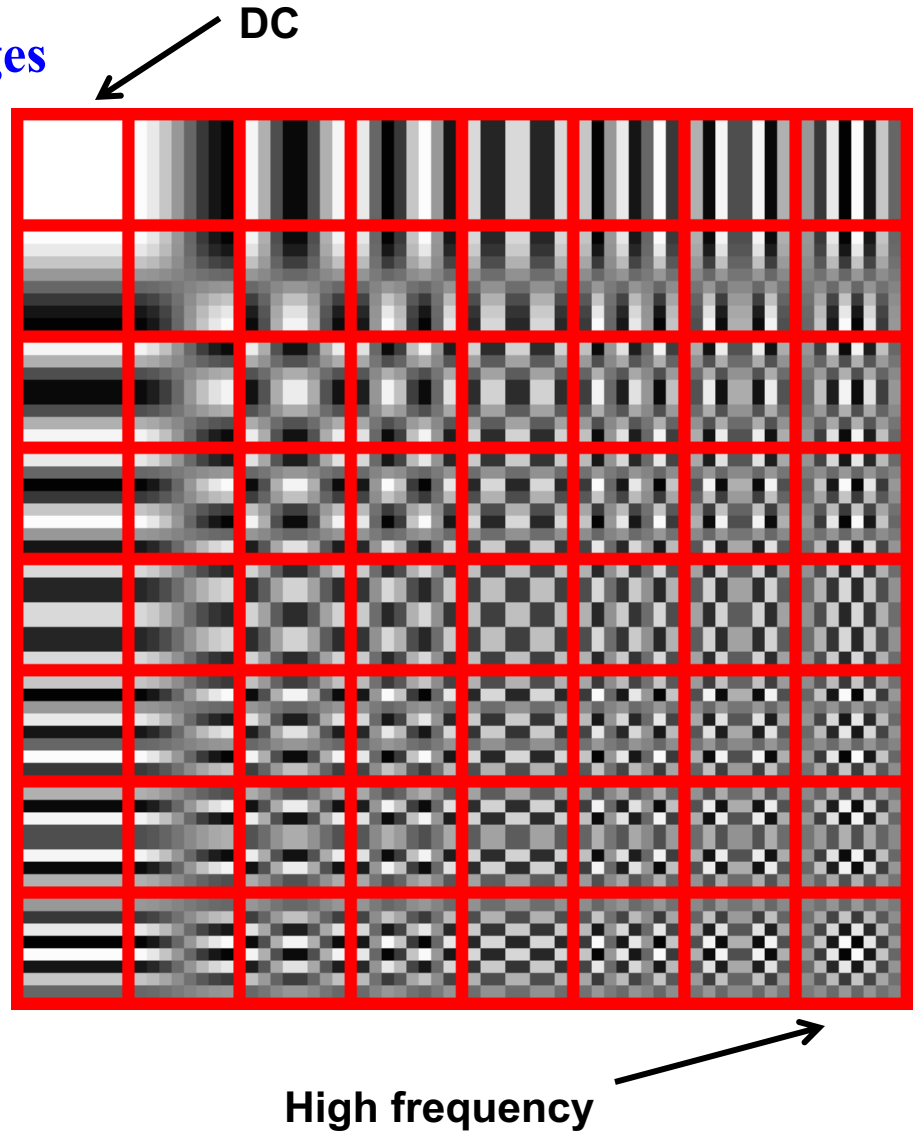
- DCT basis functions for 8 x 8 images
- Squares represent

$$\cos\left(\frac{(2r+1)u\pi}{2M}\right)\cos\left(\frac{(2s+1)v\pi}{2N}\right)$$

as *r*, *s* vary from 0 to 7

- Frequency increases as we move right & down
  - ½ cycle for each square

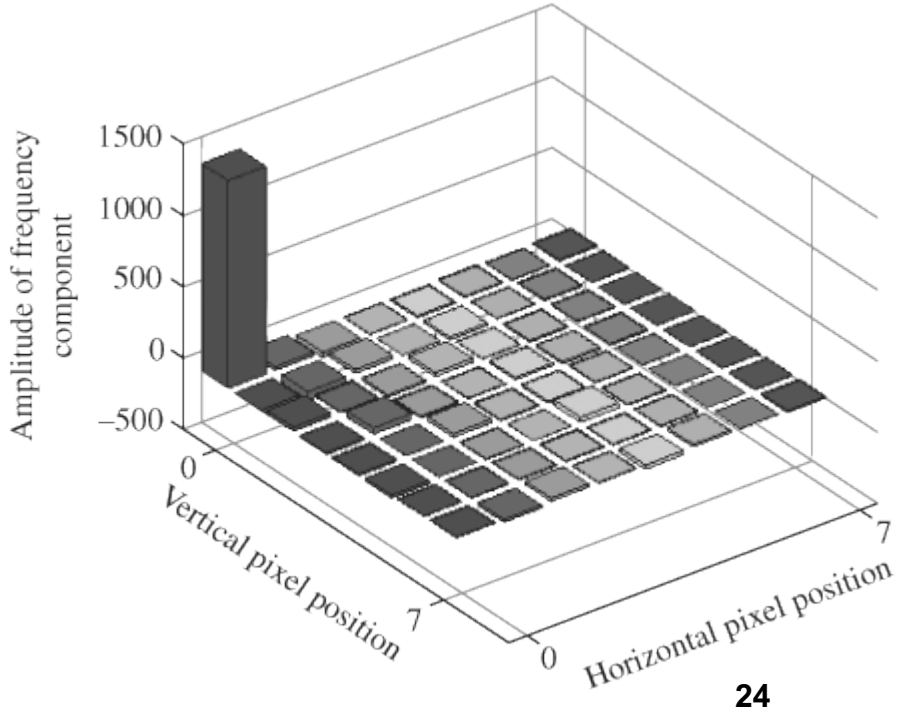
- Any 8 x 8 image can be represented by choosing appropriate weights for these basis functions



DCT & Compression

- Compute DCT for the 8 x 8 image:
- What do you expect?
- Not too much color changes ➔ DC & few low frequency components

- Does DCT help in image compression? How?
- Many coefficients are 0 ➔ easy to represent them in compact form



Summary of Image Representation

- **Analog (natural) signals are converted to digital by**
  - **Sampling (by at least double the signal frequency– Nyquist)**
  - **Quantization (some loss of info, depending on the quantization step (number of bits used) → SQNR**
- **Images can be represented as waveforms (value vs space)**
- **Waveforms can be decomposed into essential frequency components using Fourier analysis**
- **2-D DCT is used to analyze images (discrete signals)**
  - **Allows us to understand frequency components of an image**
  - **Helps in compression**

Compression Methods

- **Audio, image, and video require huge storage and network bandwidth if not compressed**
- **Example:**
  - **10-min video clip (with no audio)**
  - **30 frames per sec**
  - **Frame resolution = 720 pixels x 480 pixels**
  - **Bits per pixel = 8 x 3 = 24**
  - **Video file size ~= 17.4 GB**
  - **Bandwidth required for streaming > 240 Mb/s**

Types of Compression

- **Lossless Compression: no information is lost**
  - **Run-Length Encoding (RLE)**
  - **Entropy Encoding**
    - **Shannon-Fano, Huffman**
  - **Arithmetic Encoding**
- **Lossy Compression: some (unimportant) information is lost**
  - **E.g., frequencies not heard in audio, subtle details not noticed in image ➔ high compression ratios**

Run-Length Encoding

- **Idea: consider the following pixel values**
  - 255, 255, 255, 255, 240, 240, 240, 150, 150, 150
- **RLE (value, repetition): (255,4), (240, 3), (150, 3)**
- **Size of compressed string?**
  - Value: needs 8 bits (ranges from 0 to 255)
  - Repetition: depends on the longest run in the image
  - Assume repetition takes 8 bits for the above example ➔
  - Compression ratio = (10 x 1 byte) / (3 x 2 bytes) = 1.67
- **RLE used in image/video compression**
  - Usually there are rows of pixels with same color
  - Or rows of zero AC coefficients
- **RLE may increase size in some situations!**
  - 255, 255, 240, 210➔ (255, 2), (240,1), (210,1) ➔ ratio = 4/6=0.67

Entropy Encoding

- **Entropy of information source *S* generating symbols is:**

$$H(S) = \sum_i p_i \log_2 \left(\frac{1}{p_i} \right)$$

  - p_i **: probability of symbol *i* appearing**

- **Entropy measures the degree of randomness (uncertainty) of symbols generated by the source**
  - ***S* always generates same specific symbol ➔ *H(S) = 0***
  - ***H(S)* increases as uncertainty increases, max when all symbols are equally likely to appear**

- **Shannon showed that:**
  - **The minimum average number of bits needed to represent a string of symbols equals to its entropy**

Entropy Encoding: Examples

- **Ex1: image with 256 pixels, each with different color ➔**
  - $p_i = 1/256$ for $i = 0,1,2, \ldots, 255$ ➔ Entropy = 8
  - That is, average #bits to encode each color is 8 (cannot do better)
- **Ex2: image with 256 pixels, with the following**
  - Entropy = 2.006 ➔ min avg #bits to represent a color
  - We can achieve this by assigning different #bits (codes) to different colors (variable length encoding)
- **Huffman and Shannon-Fano algorithms approximate this**

Color	Freq	p_i	Min #bits
black	100	0.391	1.356
white	100	0.391	1.356
yellow	20	0.078	3.678
orange	5	0.020	5.678
red	5	0.020	5.678
purple	3	0.012	6.415
blue	20	0.078	3.678

Huffman Algorithm

- **Variable Length Encoding:**
  - **Fewer number of bits for more frequent colors**
- **Prefix-free code:**
  - **No ambiguity during decoding → space saving (otherwise, we need either length field or terminating sequence)**
- **Two passes:**

I. Determine the codes for the colors

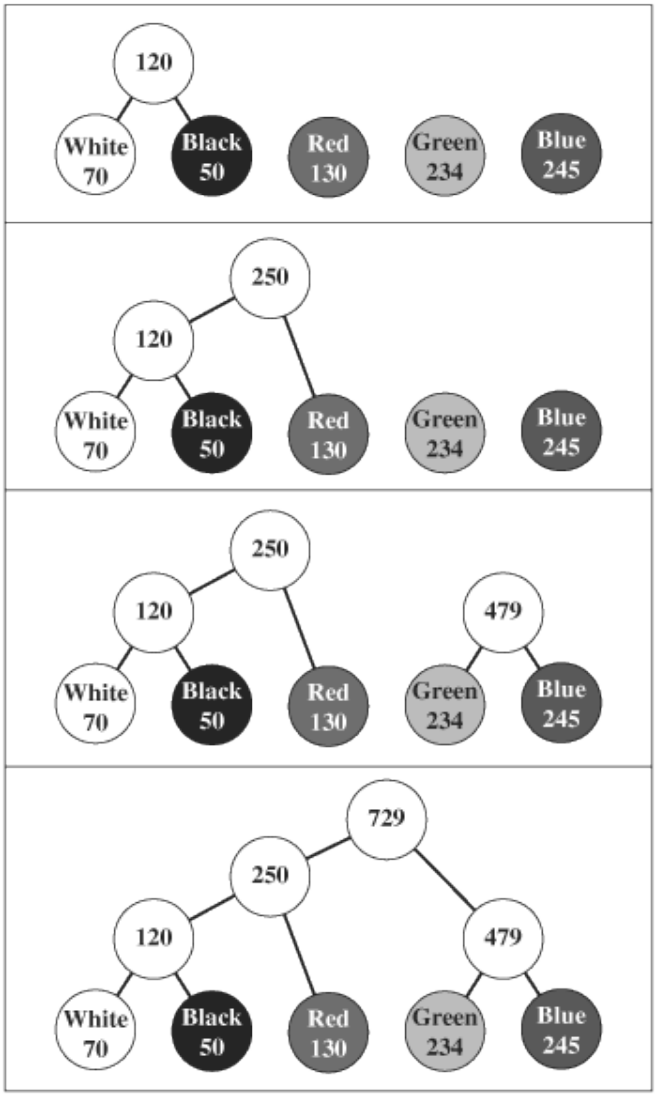
1. Compute frequencies
2. Build Huffman tree (bottom up)
3. Assign codes (top down)

II. Replace each color by its code

Huffman Algorithm: Example

- Image with five colors

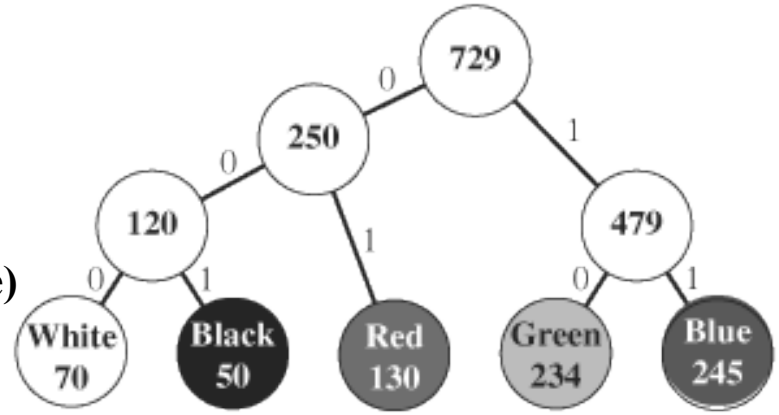
- Huffman tree



Huffman Algorithm: Example (cont'd)

- **Assigning codes top down**
- **Decoding**
  - **Need frequencies (or the tree)**
  - **Bits from the compressed file are matched (left=0 & right=1) from root down**
- **Note: each color needs integer number of bits, although the optimal (according to Shannon) may be fractional ➔**
- **Arithmetic Encoding**

Label branches with 0s on the left and 1s on the right.



For each leaf node, traverse tree from root to leaf node and gather code for the color associated with the leaf node.

White	000
Black	001
Red	01
Green	10
Blue	11

Note that not all codes are the same number of bits, and no code is a prefix of any other code.

Arithmetic Encoding

- **Avoids the disadvantage of Huffman encoding**
  - Comes closer to the optimal
  - Still uses statistical analysis (entropy coding)
- **It encodes a whole string of symbols in one floating point number**
  - Each symbol is assigned a *probability interval* with size proportional to its frequency of occurrence
- **The code (floating point number) of a sequence of symbols is created by successively narrowing the range between 0 and 1 for each symbol**

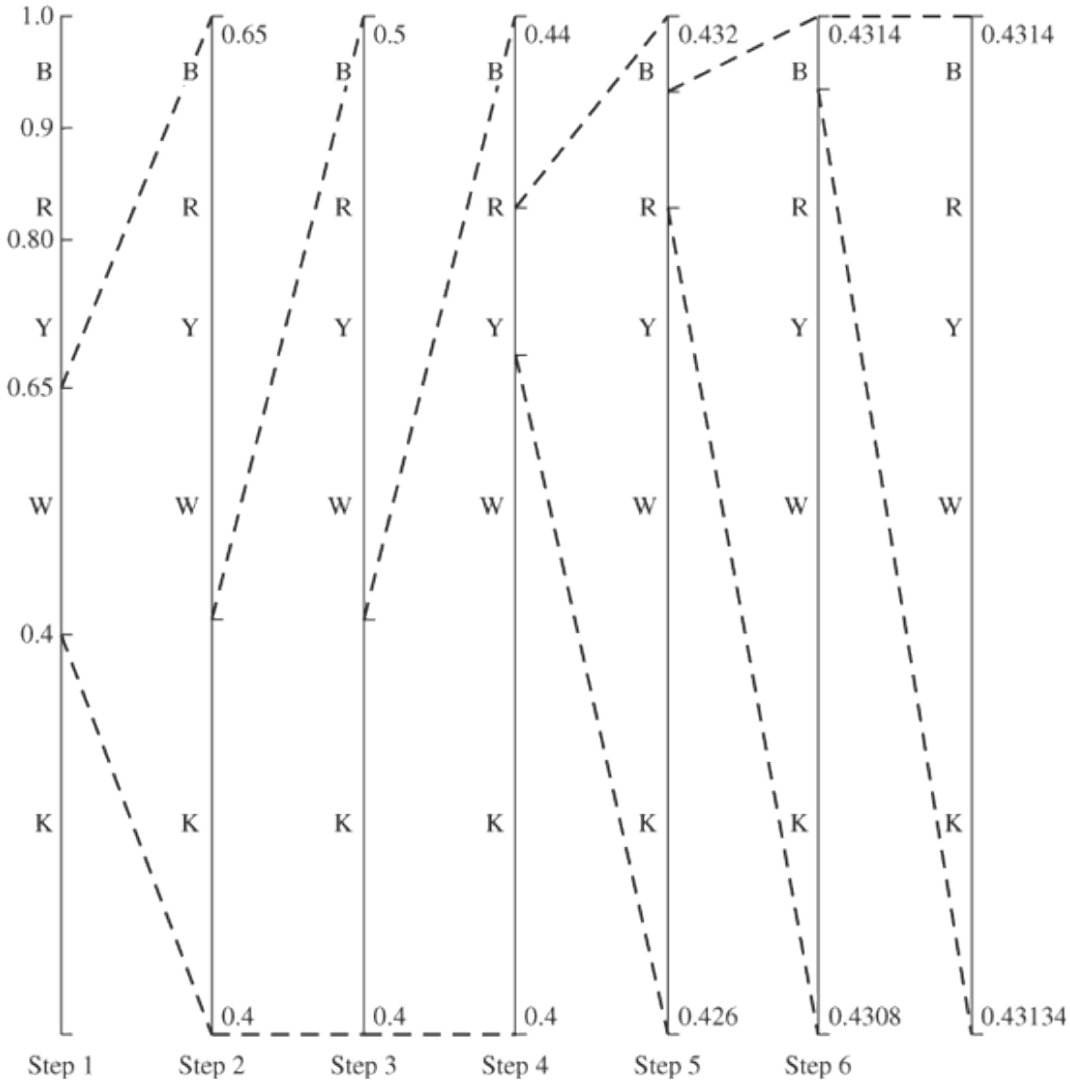
Arithmetic Encoding: Example

- 100 pixels with frequencies ➔
- Consider encoding 6 pixels: W K K Y R B
  - W: interval 0.4 – 0.65
  - K: interval 0 – 0.4 of the W's interval
  - And so on …

Color	Frequency	Probability Interval
black (K)	40/100 = 0.4	0–0.4
white (W)	25/100 = 0.25	0.4–0.65
yellow (Y)	15/100 = 0.15	0.65–0.8
red (R)	10/100 = 0.1	0.8–0.9
blue (B)	10/100 = 0.1	0.9–1.0

Arithmetic Encoding: Example

- **Encoding of W K K Y R B**



Arithmetic Encoding: Example

- **Decoding:**
  - **Assume final number (code) is 0.43137**
  - **Falls in W's interval ➔ first symbol is W**
  - **Subtract low value of W's interval and scale by its width ➔ (0.43137 – 0.4)/0.25 = 0.12548**
  - **which is in K's interval ➔ second symbol is K**
  - **… and so on**

Arithmetic Encoding: Notes

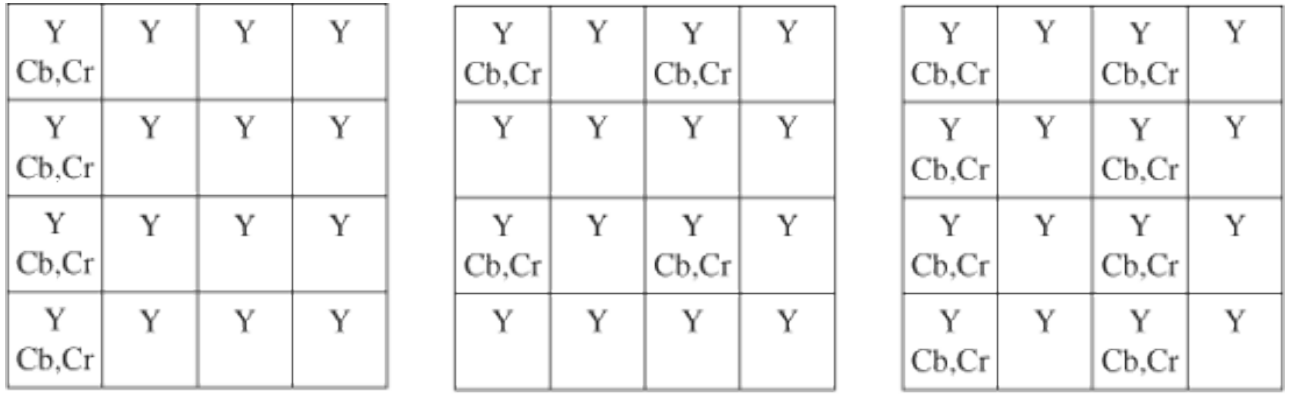
- Form of entropy encoding
- But gives closer to optimal results (more compression) than Huffman encoding
- Can be done using only integer operations
- IBM and other companies hold patents on algorithms for arithmetic encoding
- Used in recent video coding standards (H.264/AVC)

JPEG Compression

- Divide image into 8 × 8 pixel blocks
- Convert image to luminance/chrominance model, e.g., YCbCr
  - Optional; could apply same algorithm on each of the R, G, B components
- Apply 2d DCT
  - Shift pixel values by -128 (makes image more centered around 0)
- Quantize DCT coefficients
- Store DC value (upper left corner) as the difference between current DC value and DC from previous block
- Do run-length encoding
  - in zigzag order
- Do entropy encoding, *e.g.*, Huffman
- Store file in standard format (header contains info for decoder, e.g., quantization tables, Huffman codes, …)

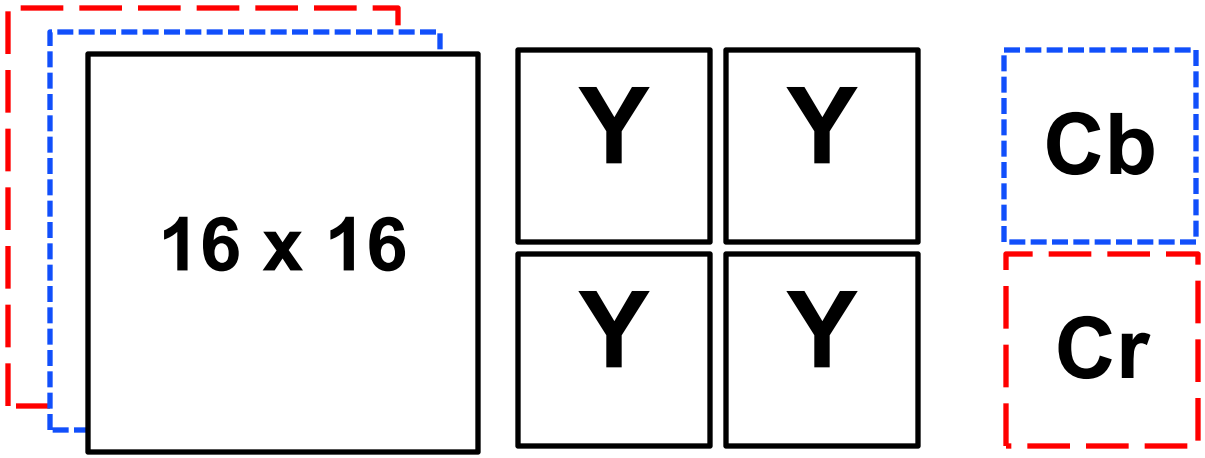
Chroma Subsampling

- **Eye is more sensitive to changes in light (luminance) than in color (chrominance) ➔ subsample CbCr**
- **Subsampling notation: *a:b:c***
  - **From 4 x 4 block: take *a* samples from Y; *b* samples from each of Cb & Cr from top row; and *c* samples from each of Cb & Cr from bottom row**
  - **Common examples: 4:1:1; 4:2:0; 4:2:2**
  - **Ex: 4:2:0 yields a saving of (16 * 3) / (16 + 4 * 2) = 2**



Subsampling and Macroblocks

- **With subsampling (i.e., if YCbCr is used), we create 8 x 8 blocks as follows:**
  - **Divide the image into 16 x 16 macroblocks**
  - **➔ four 8 x 8 blocks for Y (no subsampling for Y)**
  - **#of blocks for CbCr depends on the subsampling**
  - **E.g., 4:2:0 & 4:1:1 ➔ one block for Cb & one for Cr**



Quantize DCT Coefficients

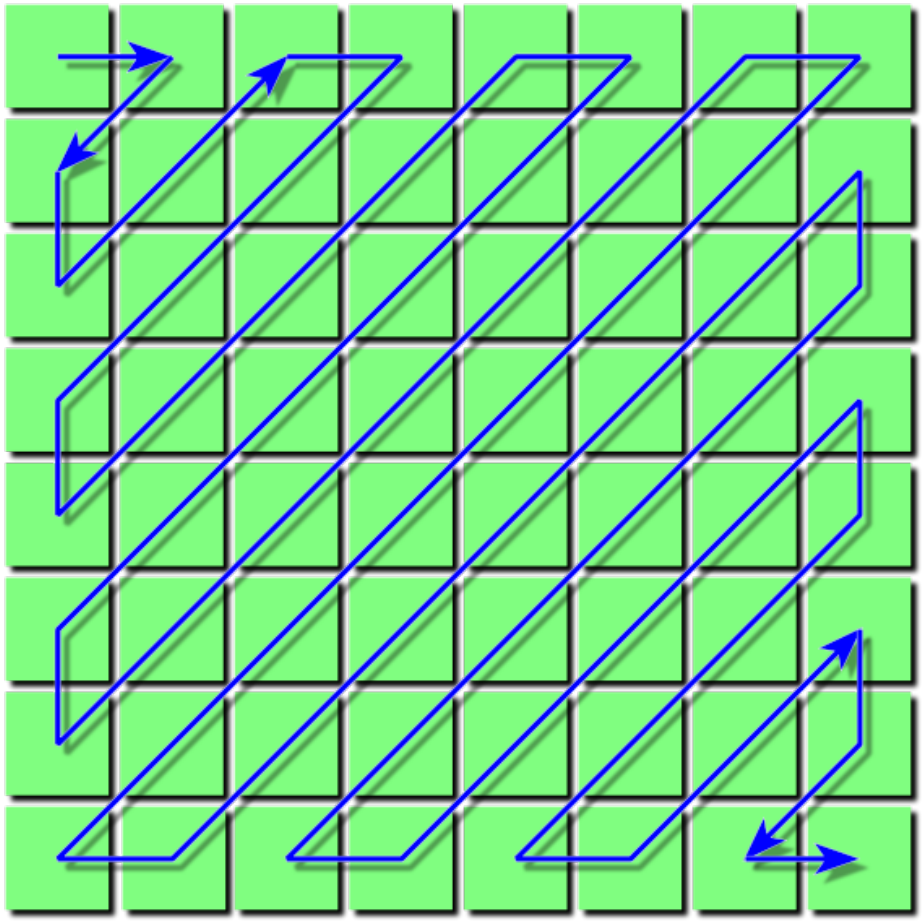
- **Uniform Quantizer: Divide each coefficient by integer and round**
  - The only lossy operation in the whole compression algorithm
    - Larger integers ➔ Larger compression AND larger distortion/error
  - High frequency coefficients are usually small ➔ become zeros ➔ more compression
  - Quantization Table
    - Each coefficient could have a different quantizer
    - Larger quantizers for high frequency coefficients

```
uint_8 luminance_quant_tbl[DCTSIZE2]={
  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
};
```

```
u_int8 chrominance_quant_tbl[DCTSIZE2]={
  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
};
```

Run-length Encoding

- **Done in zigzag order**
  - sorts values from low-frequency to high frequency components ➔
  - longer strings of 0's (because high frequency components are usually 0)



Summary on Image Compression

- **Lossless compression: RLE, Entropy, Arithmetic**
- **Lossy compression: ignores less important info to achieve higher compression ratios**
- **Chroma subsampling: take fewer samplers from colors**
  - **de-emphasize color components because eyes are more sensitive to luminance**
  - **E.g., 4:2:2 → 4 samples Y, 2 samples CbCr each from even row, 2 samples CbCr from odd rows**
- **JPEG compress**
  - **Blocks → convert to YCbCr → DCT → Quantize → zigzag RLE → Entropy coding**