
Self-Study


Exercise

Block lecture

Multimedia Coding

- Methods and Applications -

Part 3: Still Image Coding

Dr. Uwe Rauschenbach

Multimedia Coding

Part 3: Still Image Coding

Definition: Still image

- **Digital still image**
  - A still image is an ordered set of two-dimensional signals.
  - A *digital* still image is a matrix of pixels, each representing a tuple of channels.

A bit more formal:

Let be $\mathbb{P}$ the set of pixel values.

Pixel: $p_{xy} := f(x,y)$ with $x \in \mathbb{N}_0, y \in \mathbb{N}_0$ and $p_{xy} \in \mathbb{P}$

Still image: $[\mathbf{P}]^{n_x \times n_y} := \begin{pmatrix} p_{00} & p_{10} & \cdots & p_{n_x 0} \\ p_{01} & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ p_{0\,n_y} \cdots & \cdots & \cdots & p_{n_x\,n_y} \end{pmatrix}$ or short $\mathbf{P}$

Types of still images

- **The range of the pixel values describes the type of the image**

Let be $C(n_{max}) := \{c | c \in \mathbb{N}_0 \wedge c < n_{max}\}, n_{max} \in \mathbb{N}_0$

- **Image types**
  - Binary image (bi-level image) $\mathbb{P} = C(2)$
  - Grayscale image $\mathbb{P} = C(grey_{max})$
    → Pixel represents the intensity, common values for $grey_{max}$ 256 or 4096
  - Palettized / colormapped image $\mathbb{P} = C(idx_{max})$
    → Pixel represents an index into a palette (colormap), usually idx_{max} <= 256
  - True color image $\mathbb{P} = C(n^0_{max}) \times C(n^1_{max}) \times C(n^2_{max}) \times ... \times C(n^N_{max})$
    → multi-channel image with a corresponding color space (see later), usually n_{max} <= 256

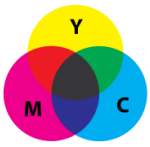
Multimedia Coding
Part 3: Still Image Coding

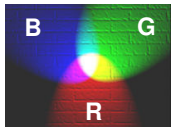
Color perception and color spaces

- **Tristimulus principle:**
  - a large fraction of the perceptible colors can be emulated by the combination of three color stimuli
    - Red / Green / Bue (RGB)
    - Cyan / Magenta / Yellow (CMY)
- **The human eye is much more sensitive regarding brightness information than regarding color information**
  - Properties of the human eye
    - Intensity perception: high photosensitivity, carrier of the image sharpness
    - Color perception: lower photosensitivity and sharpness, highest sensitivity for hues of green
  - How to exploit these properties for image compression?
    - Color components (Chrominance) can be quantized stronger and can be represented with lower resolution than intensity components (Luminance)
    - → wanted: Color space which separates Intensity and Color components, de-correlating them
    - RGB and CMY are not suited here, as each channel carries a part of the color information

Tristimulus color spaces: RGB and CMY

- **RGB and CMY can be used directly to drive graphical output devices**
  - Additive color system: RGB for self-emitting devices (screen, beamer)
  - Subtractive color system: CMY / CMYK for reflecting surfaces (color print)
- **The totality of all colors representable by a particular device is called *Gamut***
- **Common Formats**
  - 24 bit (eight bits for each of R,G,B or C,M,Y)
  - 32 bit (eight bits for each of R,G,B, plus transparency α)
  - 16 bit (5 bits for each of R and B, 6 bits for G due to the higher sensitivity of the eye for hues of green)





Conversion

$$C = I_{max} - R \qquad R = I_{max} - C$$
$$M = I_{max} - G \qquad G = I_{max} - M$$
$$Y = I_{max} - B \qquad B = I_{max} - Y$$

I_{max}: max. intensity per channel

Luminance-chrominance color spaces: YUV / YC_bC_r

- **Idea**
  - Compute the **luminance** Y from all three color channels
  - Compute two color difference signals (**chrominance**)
- **Application**
  - backwards-compatible color TV: YUV
    - Y is used by black-and-white TV sets; U and V are additional needed by color TV sets
  - Image compression: YC_bC_r
    - C_b+C_r are be represented in lower resolution and quantized stronger than Y
- **Conversion**

$$\begin{aligned} Y &= 0.29900 \cdot R + 0.58700 \cdot G + 0.11400 \cdot B \\ C_b &= -0.16874 \cdot R - 0.33126 \cdot G + 0.50000 \cdot B \\ C_r &= 0.50000 \cdot R - 0.41869 \cdot G - 0.08131 \cdot B \end{aligned}$$

$$\begin{aligned} R &= Y + 1.40200 \cdot C_r \\ G &= Y - 0.34414 \cdot C_b - 0.71414 \cdot C_r \\ B &= Y + 1.77200 \cdot C_b \end{aligned}$$

YC_bC_r: Example (with sub-sampled color components)

Further color spaces

- **„Perception-oriented“ color spaces for image editing**
  - HLS (Hue – Lightness – Saturation)
  - HSB (Hue – Saturation – Brightness)
  - HSV (Hue – Saturation – Value)
  - Components
    - Hue (Color): „which color“
    - Saturation: Degree of purity of the color (S=0: Grey value; maximum S: „pure Color“)
    - 3rd component: Brightness (grey value at S=0)
- **CIE: contains all perceivable colors; superset of all gamuts (RGB, CMY)**
- **CIE Lab: equidistant color space (i.e. neighboring colors with same distance in the model have the same perceived difference)**

Palettized images

- **Main principle:**
  - Pixel values are indices into a color map (palette)
  - The color map contains RGB triplets
  - This is a memory-efficient representation of pictures with few colors
  - Hardware support available (CLUT – Color Look Up Table in Graphics chips)
  - Transparency: one specific index value can be marked as being „transparent“ (“Magic Color”)
  - Examples: GIF, PNG

Pixel field of palettized image

0	0	1
2	3	3
2	1	0

Color map

R	G	B
255	0	0
0	255	0
0	0	255
255	255	0

Comparison: Pixel field of a true color image

255,0,0	255,0,0	0,255,0
0,0,255	255,255,0	255,255,0
0,0,255	0,255,0	255,0,0

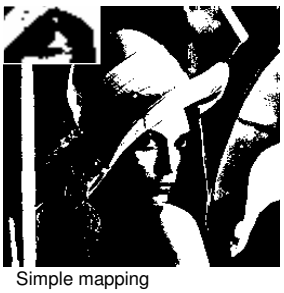
Color Quantization

- **Goal:** Convert a true color image into a palettized image with N colors

- **Basic idea of the color quantization algorithm:**
  - Recall: Vector quantization from part 1
  - Algorithm
    - Code book generation: Find the N „most representative“ colors in the image
    - Color map generation: Create the color map from the code book
    - Mapping: For each pixel: Read the RGB triplet in the input image, find the “most similar” triplet in the code book, assign the index of this tuple in the color map to the pixel in the output image
  - Different algorithms exist for finding the „most representative colors“:
    - Octree: divide the color space into sub-cubes with the same number of colors
    - Median Cut: define cutting planes that divide the RGB space into cuboids

Dithering

- **is a mapping method which reduces the effective resolution but creates the subjective impression of more color or intensity levels**
- **introduces "dithering patterns"**
- **adds noise to the image, increasing the entropy**
  - what does this mean w.r.t. compression?



Simple mapping

Dithering with error diffusion

Example: Mapping a grayscale image (8bit) to a bilevel image (1bit) without and with dithering

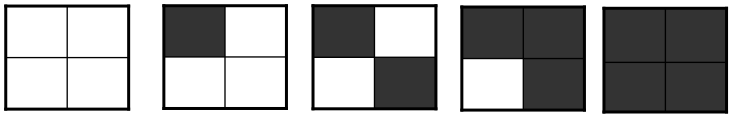
Dithering and Error Diffusion in Detail

- **Dithering is a mapping method which reduces the effective resolution but creates the subjective impression of more color or intensity levels**
  - different methods, see next slides
- **Applications**
  - halftone printing (represent grey levels with black and white (newspaper!)
  - rendering of true color images on displays with few colors (16/256)
- **Dithering and image compression**
  - Dithering adds noise to the image, increasing the entropy.
  - This means the achievable compression ratio drops.
  - Thus, dithering should only be used when the simple mapping does not give good results (e.g. Banding artefacts, flat colors)

Ordered Dithering

- **Basic idea (1 bit case):**
  - the dithering matrix contains intensity values m_i, against which the value p_i of an input pixel at the according position is compared. If $p_i > m_i$, the output pixel is set, otherwise not set.
  - the output image contains characteristic dithering patterns, which are "interpolated" by the human eye as different intensity levels.
  - depending on the application, there are many different dithering matrices (see Foley/van Dam).
  - a dithering matrix of order n can map n^2+1 different input intensity levels to a bi-level output image.
- **Example of a dithering matrix (n=2) and generated dithering patterns**

0	2
3	1

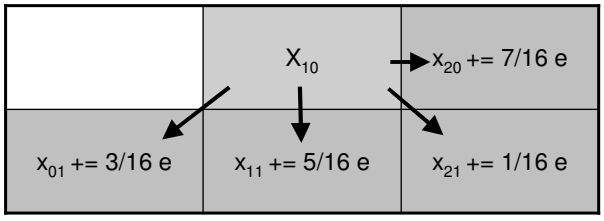


- **This can be generalized to output images with more than two levels**

Error Diffusion

- **Elegant method to spread the quantization error**
  - developed in 1975 by Floyd and Steinberg (other similar methods exist)
  - goal: minimize the average error
- **Basic idea**
  - the quantization error introduced is propagated iteratively to the neighbouring pixels, before these are quantized.
  - Error term: $e_{ij} = x_{ij}$ - *mapping*(x_{ij})

One iteration step:



Comparison of color mapping methods

Image examples, mapped from 8 bits to 1bit per pixel

Simple mapping

- maximum error
+ no patterns
→ method to use if the output image has enough colors not to introduce visible color distortions

Ordered Dithering

- smaller error
- annoying patterns
+ fast HW support
→ today, only used in hardware

Error Diffusion

+ error nearly zero
+ patterns not annoying
→ method to use if the output image has only a few colors

Multimedia Coding

Part 3: Still Image Coding

Introduction to still image coding
Classification according to loss of information

- **Image compression**

  Let be bitstream

  $$[B]^n := b_0 b_1 ... b_n \text{ mit } b_i \in \{0,1\}$$

  Then, compression and decompression are defined as follows

  $$comp(\mathbf{P}) := [B]^n \qquad decomp([B]^n) := \mathbf{P}$$

  Sequential composition leads to

  $$\hat{\mathbf{P}} := decomp(comp(\mathbf{P}))$$

- **2 classes of compression methods**
  - lossless: e.g. GIF, PNG, JPEG-LS, JBIG

    $$\hat{\mathbf{P}} = \mathbf{P}$$

  - lossy: e.g. JPEG, JPEG2000

    $$\hat{\mathbf{P}} \neq \mathbf{P}$$

NB1: lossless means “perfect reconstruction”

NB2: some of the lossless methods have a “near lossless” mode which relaxes some criterion in the algorithm leading to better compression but sacrificing perfect reconstruction. Strictly-speaking, these modes are lossy!

Introduction to still image coding
Distortion measures

- **The magnitude of information loss is described by distortion measures.**
- **Subjective distortion measures**
  - good to capture subjective impression of information loss but needs laborious user testing,
  - MOS (Mean Opinion Score)
    5 grade scale: *excellent – good – fair – poor – bad*
- **Objective distortion measures**
  - can be easily computed from the signal but do not reflect subjective impression
  - MSE (Mean Square Error)

    $$MSE(\mathbf{P}, \hat{\mathbf{P}}) = \frac{1}{n_x \cdot n_y} \sum_{y=1}^{n_y} \sum_{x=1}^{n_x} (p_{xy} - \hat{p}_{xy})^2$$

  - PSNR (Peak Signal to Noise Ratio)

    $$PSNR(\mathbf{P}, \hat{\mathbf{P}}) = 10 \cdot \log_{10} \frac{(n_{max})^2}{MSE(\mathbf{P}, \hat{\mathbf{P}})} \; [dB]$$

Introduction to still image coding
Compression ratio

- **Combined with a distortion measure, the compression ratio is the major performance criterion for image compression methods**
- **Defines either a ratio or a bit rate**
  - **Ratio:**

$$CR = \frac{\text{Size of original file}}{\text{Size of compressed file}} \qquad \text{e.g., } 100$$

  - **Bit rate:** Specifies the (usually fractional) average number of bits needed to encode one pixel in the image. Unit: bpp (bits per pixel)

$$BR = \frac{\text{Size of compressed file}}{\text{Number of pixels}} \qquad \text{e.g., } 0.23 \text{ bpp}$$

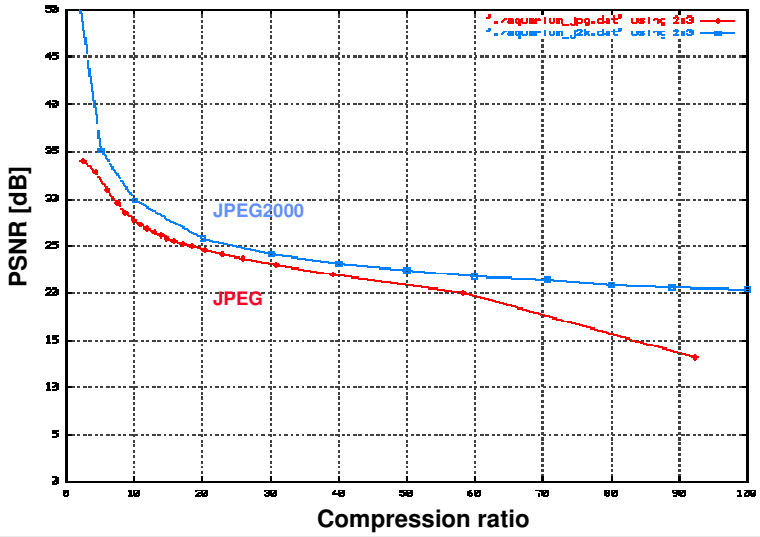
→ A compression method is the more powerful
  - the less bit rate is needed to reach the same distortion (i.e. the less bpp), or
  - the smaller the distortion is at the same bit rate (i.e. the higher the PSNR)

Introduction to still image coding
Compression artefacts

- **Lossy compression always results in compression artefacts**
  → introduced, more or less annoying structures in the picture
- **Which artefact is „less annoying“?**

JPEG: „Blocking effect“ Original JPEG2000: „Softening effect“

Introduction to still image coding
Rate-distortion curve

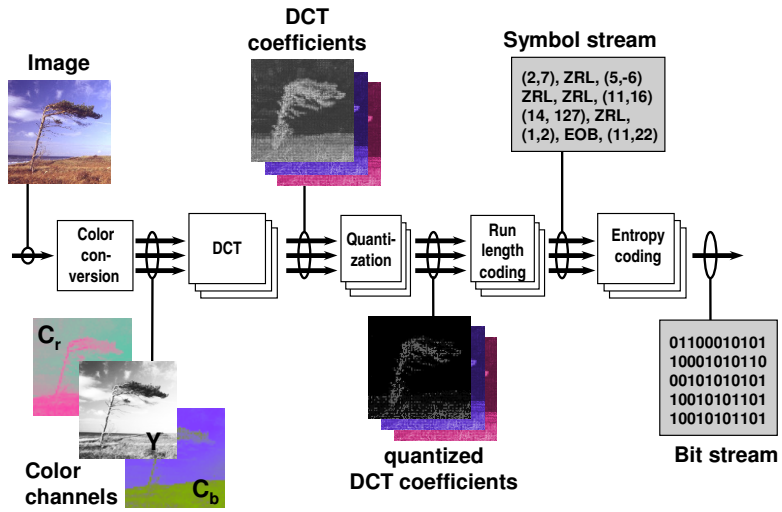


Multimedia Coding
Part 3: Still Image Coding

JPEG
Overview

- **ISO standard for compressing still images (1992)**
  - published as ISO/IEC IS 10918 and ITU-T.81
- **Developed by the „Joint Photographic Experts Group“ for photorealistic images (many colors, soft transitions)**
  - Note: There exist other compression methods (GIF, PNG, JBIG) hich are better suited to encode images with few colors → see later
- **Based on Discrete Cosine Transform (DCT); lossy**
- **Different Modes**
  - **Baseline:** Image is encoded in one pass
  - **Progressive:** Image is encoded in multiple successive quality levels, which allow to display a series of approximations of the image during download
  - **Hierarchical:** Image is encoded in multiple successive quality levels (Mode not used in practice)
  - **Lossless:** Image is encoded losslessly using context modelling (no DCT!) (Mode rarely used)
- **Reading:** Pennebaker/Mitchell // **“The” Software:** Independent JPEG Group

JPEG
Block diagram of encoding flow (revert for decoding)



JPEG
Encoding steps

1. **Color transformation RGB → YC_bC_r**
   ⇒ to separate luminance and chrominance parts
2. **Discrete Cosine Transform (DCT)**
   ⇒ to convert the pixel field into DCT coefficients
3. **Quantization of the DCT coefficients**
   ⇒ to remove „visually redundant“ information
4. **Run length coding**
   ⇒ to create long sequences of zeroes exploiting inter-coefficient relationships
5. **Entropy coding**
   ⇒ to represent the run length symbols with a minimum number of bits
6. **Data stream formatting**
   ⇒ to create a data stream with certain properties (robust, progressive)

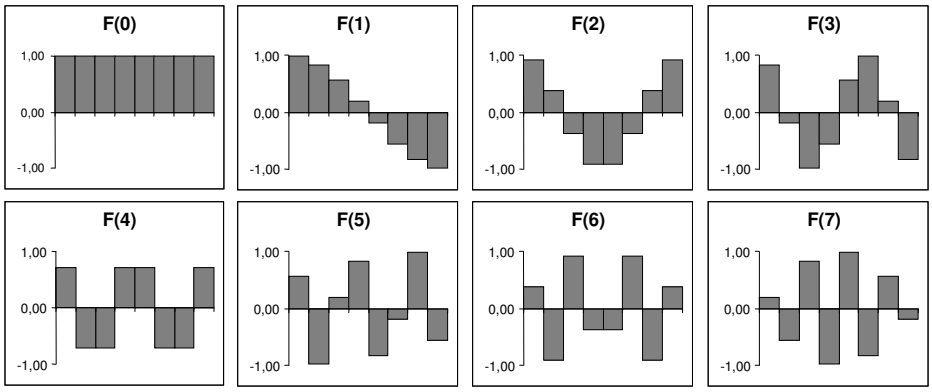
JPEG
Step 1: Color conversion

- **By definition, the JPEG standard is “color blind”: It is possible to encode multiple independent color channels**
- **I.e., this step is strictly speaking not part of the JPEG encoding**
  - But: In practice, the YC_bC_r color space is used (usually with sub-sampling of the chrominance components)
    - this is the only possible mode in JFIF (JPEG file format, see later)
    - C_b and C_r are subsampled in X and Y direction with e.g. factor 2 (more subsampling modes exist)
    - this reduces the raw data by a factor of 2 before actually starting the compression proper, without creating visible artefacts
- **The following steps are applied to each channel separately.**

JPEG
Step 2: Discrete Cosine Transformation (1)

A windowed (1D) signal can be approximated by a linear combination of the following basis functions

→ Extension to 2D signals is possible by composition of the transformation in row and column direction (separable transformation).



JPEG
Step 2: Discrete Cosine Transformation (2)

- **Subdivision of the image into blocks of 8x8 pixels**
- **Transformation of the blocks by DCT from space domain *s* into frequency domain *S***

$$
\begin{aligned}
F(x,y,u,v) &= \cos[(2x+1)u\pi/16]\cos[(2y+1)v\pi/16] \\
C(w) &= \begin{cases} 1/\sqrt{2} & \text{if} \quad w=0 \\ 1 & \text{if} \quad w>0 \end{cases} \\
S(v,u) &= \frac{C(v)}{2}\frac{C(u)}{2}\sum_{y=0}^{7}\sum_{x=0}^{7} s(y,x)F(x,y,u,v) \\
s(y,x) &= \sum_{v=0}^{7}\frac{C(v)}{2}\sum_{u=0}^{7}\frac{C(u)}{2}S(v,u)F(x,y,u,v)
\end{aligned}
$$

- **Result**
  - Field of DCT coefficients
  - Representation of the image block as a weighted sum of the basis functions *F(x,y,u,v)*

JPEG
Step 3: Quantization (1)

- **Quantization of the DCT coefficients**
  - this step causes the information loss in JPEG ("irrelevance reduction")
  - very small coefficients are quantized to zero and allow an efficient compression in the following run length encoding step
- **Quantization**
  - Quantization is realized by dividing each DCT coefficient by a quantization coefficient which is defined in a quantization table

$$q[k,l] = \left\lfloor \frac{S[k,l]}{Q_{k,l}} + \frac{sgn(S[k,l])}{2} \right\rfloor$$

  - Quantization table allows different visual thresholding of different coefficients. It must be signalled to the decoder, or a default table must be assumed.
  - Inverse quantization means the decoder multiplies the DCT coefficients with the according values

$$S'[k,l] = q[k,l] \cdot Q_{k,l}$$

JPEG
Step 3: Quantization (2)

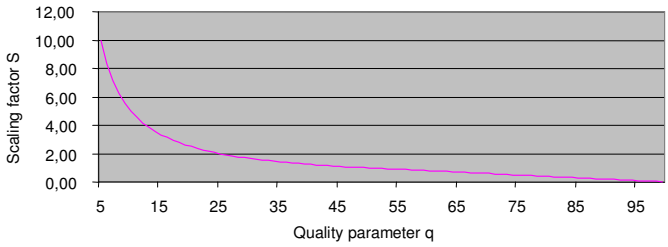
Example: Quantization table from the JPEG Standard (Y channel)

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

JPEG
Step 3: Quantization (3)

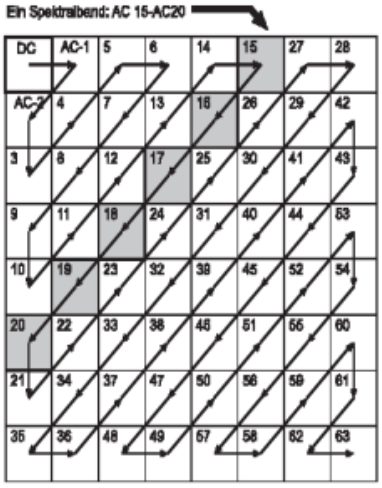
- **Controlling the information loss**
  - Many encoders allow the definition of a „quality factors“ *q* (often between 5 and 100%)
  - All coefficients in the standard quantization table are multiplied with a factor S, which is a function of *q*
  - This is not standardized!
  - Example: Computation of S in the implementation of the Independent JPEG Group (see section on Software Tools)

```
if(q<50) S:=50/q else S:=2-2*q/100
```



JPEG
Step 4: Traversing and Run Length Coding (1)

- **Result of the DCT: 1 DC coefficient (constant component) and 63 AC coefficients**
- **Coefficients are visited in a zig-zag order**
  - Assumption: The probability that a coefficient is zero is the higher, the higher the spatial frequency is which is represented by the coefficient
  - Goal: generate long sequences of zeroes
- **Encoding of coefficients**
  - magnitude-based
  - DC and AC coefficients have different characteristics and are encoded differently



JPEG
Step 4: Traversing and Run Length Coding (2)

Encoding of DC coefficients

- Since DCT coefficients of neighbouring blocks are similar, the first step is a DPCM prediction (i.e. computing the delta)
- For each delta value, a symbol m (magnitude) is generated as follows:
  - exact value for small coefficients
  - order of magnitude for large coefficients
    - → number of different symbols remains manageable
    - → in addition to m, the sign and the m-1 less significant bits are transmitted without encoding

DC coefficient DELTA	m
0	0
+/- 1	1
+/- 2 … 3	2
+/- 4 … 7	3
+/- 8 … 15	4
+/- 16 ... 31	5
+/- 32 … 63	6
+/- 64 … 127	7
+/- 128 … 255	8
+/- 256 … 511	9
+/- 512 … 1023	10
+/- 1023 … 2047	11

JPEG
Step 4: Traversing and Run Length Coding (3)

Encoding of AC coefficients

- run length encoding of a pair (n, M)
- n: number of coefficients equal to zero (0 … 15)
- M: value of the first non-zero coefficient at the end of the run, as follows
  - exact value m for small coefficients
  - order of magnitude m for large coefficients
    - → number of different symbols remains manageable
    - → in addition to m, the sign and the m-1 less significant bits are transmitted without encoding
- special symbols
  - EOB (End of Block): if all remaining coefficients in the block are equal to zero, an EOB symbol is sent after encoding the last non-zero coefficient
  - ZRL (Zero Run Length): encodes a sequence of 15 zeroes which are not followed by a non-zero coefficient

coefficient	m
+/- 1	1
+/- 2 … 3	2
+/- 4 … 7	3
+/- 8 … 15	4
+/- 16 ... 31	5
+/- 32 … 63	6
+/- 64 … 127	7
+/- 128 … 255	8
+/- 256 … 511	9
+/- 512 … 1023	10

JPEG
Step 5: Entropy coding

- **In this step, the DC and AC run length symbols are entropy-coded**
  - Huffman coding or arithmetic coding
    - in baseline mode: just Huffman coding allowed
    - arithmetic coding not used in practice due to IPR licensing issues
  - Huffman tables are transmitted to the decoder as side information
- **The encoded symbols are followed by the uncoded refinement information if needed**
  - sign
  - m-1 bits to define the exact value

JPEG
Step 6: Data Stream Formatting (1)

- **A JPEG data stream contains**
  - Marker segments: signalling of control information
  - Data segments: transmission of encoded data
- **Marker segments**
  - Structure
    - Start code 0xFF
    - Type (1 Byte)
    - Length → to support backwards compatibility (allows skipping unknown markers)
  - Stuffing bits
    - if the code 0xFF occurs in entropy-coded data, a zero byte is appended to discriminate from marker
    - this byte is removed in the parser of the decoder
  - Restart markers to Re-synchronize
    - provide re-entry points for the decoder in case of transmission errors
    - at restart markers, entropy decoder status and DC coefficient prediction are re-set

JPEG
Step 6: Data Stream Formatting (2)

Example of a JPEG data stream

```
SOI – start of image
  COM – comment
  DQT – define quantization tables
        SOF – start of frame
              DHT – define Huffman tables
              SOS – start of scan
                    encoded data
                    RST – restart marker
                    encoded data
                    …
              DHT – define Huffman tables
              SOS
              …
EOI – end of image
```

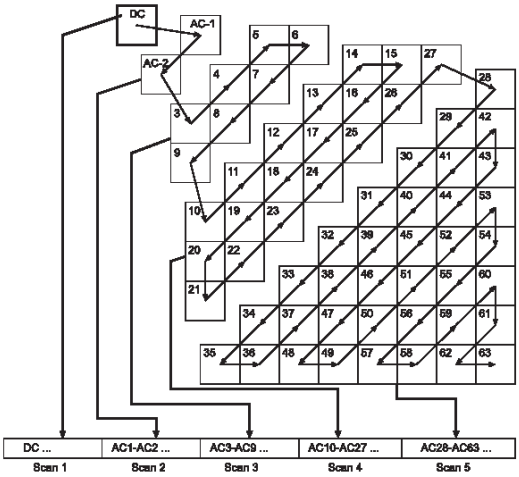

JPEG
Progressive Mode

- **Basic idea**
  - Allow the presentation of approximations of the image in reduced quality at the receiver side during reception of the image (i.e. before all image data have been received)
  - To support that, the image is encoded in multiple passes
  - Each pass is named *Scan*
- **Two dimensions**
  - Spectral selection (selection of spectral bands of DCT coefficients)
  - Successive approximation (selection of bitplanes of DCT coefficients)
  - Both modes can be combined for better quality of intermediate approximations
- **Applications**
  - Use of online services over slow connections
  - Dynamic memory management in digital cameras
    - if the memory is full, cut away the last scan of images to gain space for a new picture

JPEG
Progressive Mode: Spectral Selection

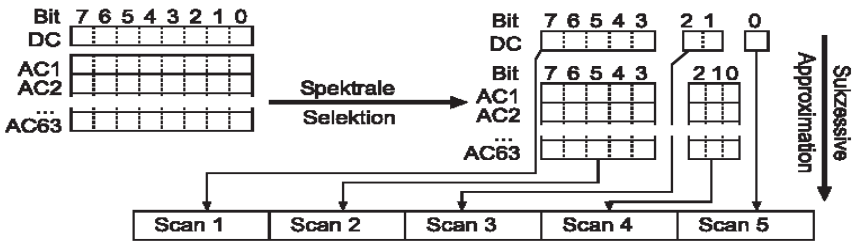
Basic principle

- one scan corresponds to one or more spectral bands
- the encoding is done as described already, with the only difference that it is separated into scans
- note: the DC coefficient is always encoded in a separate scan



JPEG
Progressive Mode: Successive Approximation

- **Basic Idea**
  - One scan contains one ore more bitplanes of all DCT coefficients (DC and AC separated)
  - Within each bitplane, significance and refinement information are treated differently
    - Significance information: A coefficient changes its status from zero to nonzero in the current pass (encoded by the run length coding method described earlier)
    - Refinement information: Transmission of further bits to refine a coefficient which is already known to the decoder as significant (these bits are sent uncoded)
- **Example: Successive approximation and spectral selection combined**



Multimedia Coding
Part 3: Still Image Coding

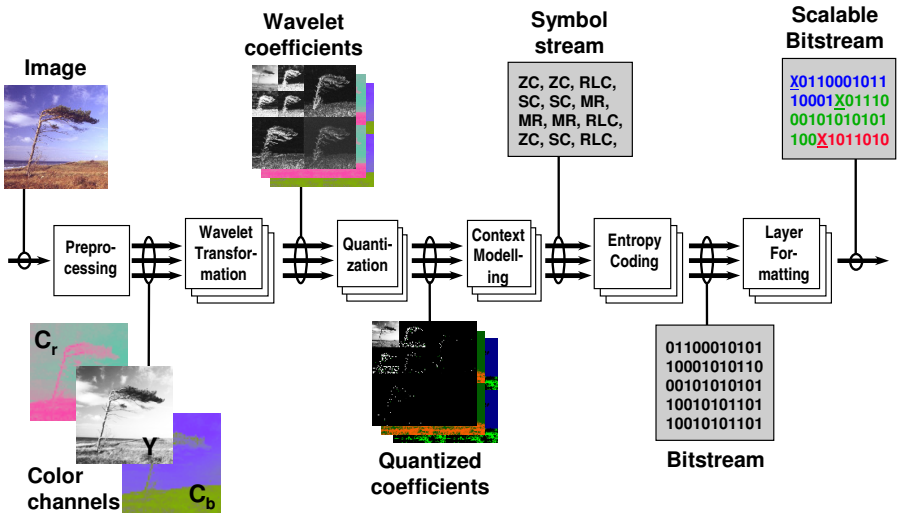
JPEG2000
Overview

- **ISO Standard for encoding still images**
- **Better compression than JPEG**
- **Wavelet based**
- **Scalable data stream supporting fast transcoding („one for all“)**
  - data stream is divided into „Layers“ which are described by packet headers
  - each layer leads to an improvement of the image
  - only those packets must be decoded which are needed for the targeted degree of detail of the image

JPEG2000
The standard in brief

- **Published as ISO/IEC IS 15444**
- **Structure** (parts in **bold** are covered in this lecture)
  - ***Part 1 (Core)*** – royalty-free (not: patent-free!), base encoding system, base file format *JP2*
  - Part 2 (Extensions) – further functionalities
  - Part 3 (Motion JPEG 2000) – Encoding of image sequences, without exploiting inter-frame coherency (i.e. Intra-only) → application in Digital Cinema
  - Part 4 (Conformance)
  - Part 5 (Reference software) – Java and C Implementations
  - ***Part 6 (Compound image file format)*** – supports different compression methods within the same document, for document processing and archiving
  - Part 7 (discontinued)
  - Part 8 (JPSEC) – Secure JPEG2000 – encryption, integrity protection, proof of source
  - ***Part 9 (JPIP)*** – interactive protocol for the browsing of big images, including API
  - Part 10 (JP3D) – for 3D images (e.g. tomography)
  - Part 11 (JPWL, wireless) – additional error protection for wireless transmission
  - Part 12 (ISO Base Media File Format) – file format harmonized with MPEG-4

JPEG2000
Block diagram of encoding flow (revert for decoding)



JPEG2000
Encoding steps

1. **Preprocessing and color transformation RGB → YC_bC_r**
   ⇒ to create a flexible co-ordinate grid and to separate luminance and chrominance information
2. **Discrete (dyadic) wavelet transformation (DWT)**
   ⇒ convert the pixel field into wavelet coefficients
   ⇒ irreversible or reversible (integer) transformation
3. **Quantization of the wavelet coefficients (optional)**
   ⇒ in JPEG2000 this step is optional, as the visually irrelevant information is removed in the Data stream formatting step.
4. **Context modelling**
   ⇒ create a low-entropy symbol stream by exploiting of statistical relationships in the neighborhood
5. **Entropy coding**
   ⇒ to represent the symbols with a minimum number of bits
6. **Data stream formatting**
   ⇒ create a scalable data stream

JPEG2000
Step 1: Preprocessing

- **Divide the image into one or more *tiles*; define a reference coordinate system**
- **Symmetric bit shifting of all pixel values** ($0 .. 2^n \rightarrow -2^{n-1}-1 .. 2^{n-1}$)
- **Color transformation**
  - the data stream syntax of JPEG2000 supports the definition of the used color transformation
  - 2 standard modes
    - irreversible color transformation (YC_bC_r, like JPEG)
    - reversible color transformation (YUV)
  - usually, the chrominance components are sub-sampled in each direction by a factor of 2 to reduce the data volume
    → exploit the reduced color sensitivity of the human visual system
    → reduce data volume by factor 2 without perceivable compression artifacts

JPEG2000
Color transformation

- **Reversible color transformation**

$$Y = \left\lfloor \frac{R + 2G + B}{4} \right\rfloor \qquad G = Y - \left\lfloor \frac{U + V}{4} \right\rfloor$$

$$U = R - G \qquad R = U + G$$

$$V = B - G \qquad B = V + G$$

- **Irreversible color transformation**
  - although there exists a reverse transformation, rounding errors lead to small differences between original and reconstructed signal

$$\begin{aligned} Y &= 0.29900 \cdot R + 0.58700 \cdot G + 0.11400 \cdot B \\ C_b &= -0.16874 \cdot R - 0.33126 \cdot G + 0.50000 \cdot B \\ C_r &= 0.50000 \cdot R - 0.41869 \cdot G - 0.08131 \cdot B \end{aligned} \qquad \begin{aligned} R &= Y + 1.40200 \cdot C_r \\ G &= Y - 0.34414 \cdot C_b - 0.71414 \cdot C_r \\ B &= Y + 1.77200 \cdot C_b \end{aligned}$$

JPEG2000
Example of YC_bC_r (with sub-sampled chrominance components)

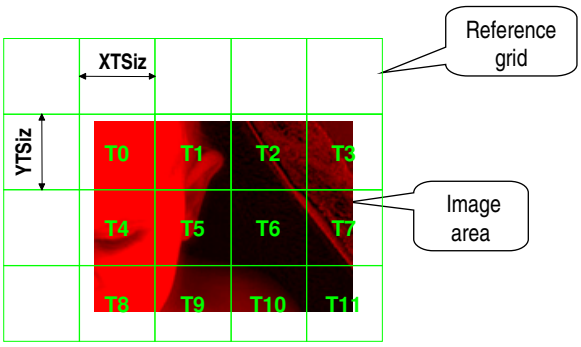
3 x 512 x 512 = 768 KB

512 x 512 + 2 x 256 x 256 = 384 KB

JPEG2000
Tiling of large images

JPEG2000 uses tiles to partition large images

- each tile is treated as an independent image during encoding, to allow random access to image parts
- a reference grid is used to define tiling
- the points of origin of reference grid, image and first tile do not necessarily have to be the same point

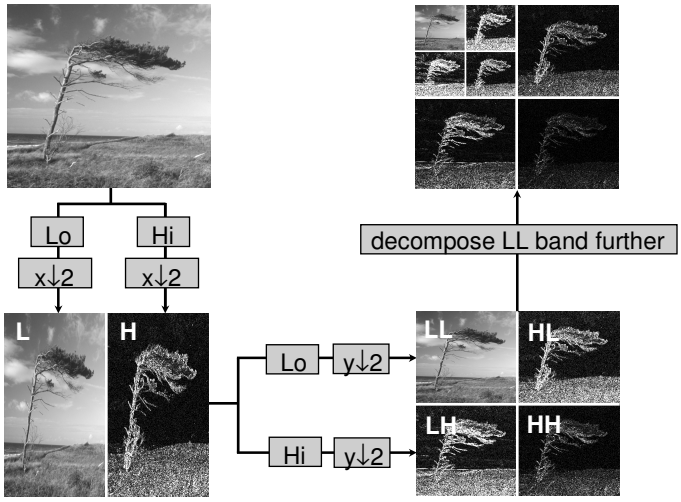


reference grid decreases quality loss during image editing tasks (cropping, mirroring, rotating in 90° steps), since changes can be performed locally at the image boundary

JPEG2000
Step 2: Wavelet transformation

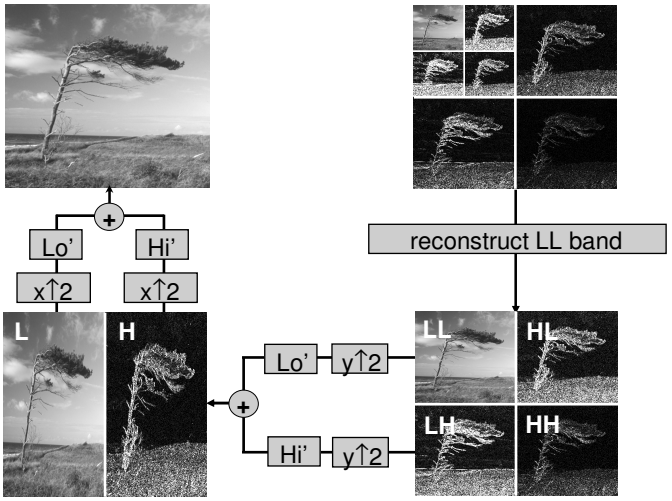
- A **filter bank** decomposes a signal into two “partial” signals: an **approximation signal** (low pass band) and a **detail signal** (high pass band)
- **Cascading:** low pass band is recursively decomposed further → octave bands
- **Dyadic decomposition scheme:** the frequency in neighboring octave bands differs by a factor of 2
- **Properties**
  - energy compaction (most energy is concentrated in the low pass band) → irrelevance reduction in high pass bands
  - perfect reconstruction of the signal by Inverse Wavelet Transformation → lossless compression possible
  - de-correlation of image data → high compression ratio possible
  - multi-resolution representation → refinement of resolution possible
  - separable transformation: 2D transformation is realized as a sequence of two 1D transformations

JPEG2000
2D Wavelet transformation illustrated (1)



two-level dyadic wavelet transformation

JPEG2000
2D Wavelet transformation illustrated (2)

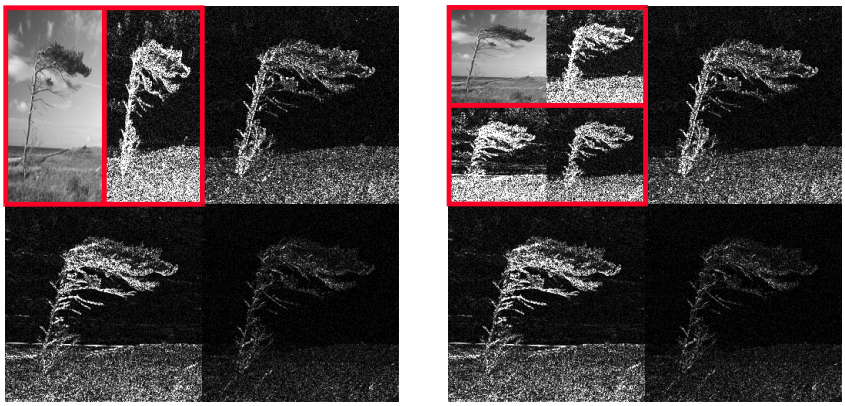


inverse two-level dyadic wavelet transformation

JPEG2000
2D Wavelet transformation: example (1)

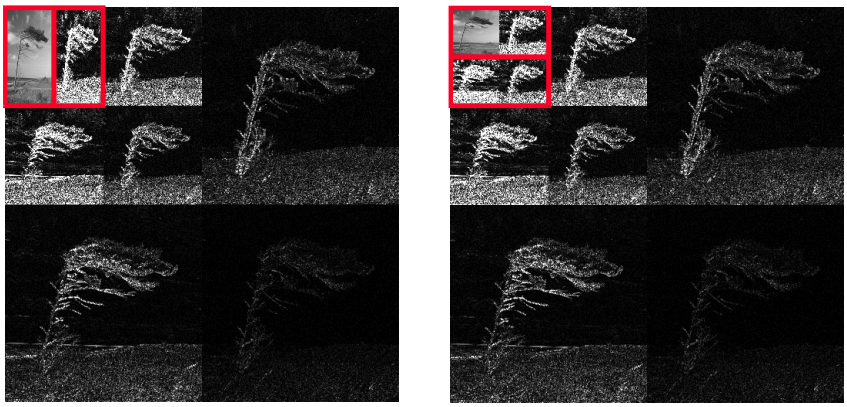
Dyadic wavelet transformation: first decomposition level

JPEG2000
2D Wavelet transformation: example (2)



Dyadic wavelet transformation: second decomposition level

JPEG2000
2D Wavelet transformation: example (3)



Dyadic wavelet transformation: third decomposition level

JPEG2000
Naming of the subbands

LL	LH_1	LH_2	LH_3
HL_1	HH_1		
HL_2		HH_2	
HL_3			HH_3

JPEG2000
Computing the wavelet transformation by convolution (1)

- **The convolution operation is defined as follows:**
  - Be x[n] a time-discrete input signal and h[n] the impulse response of a filter. Then, y[n] is defined to be the convolution sum of x and h. It is computed as follows:

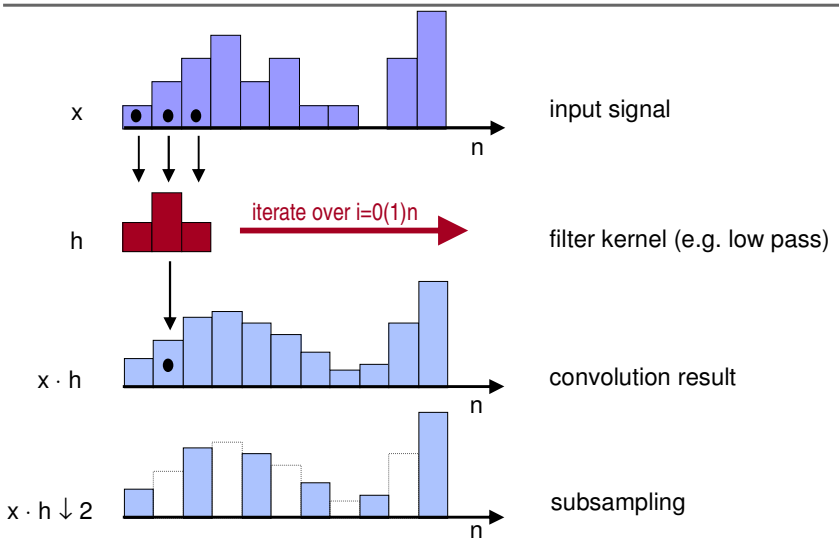
$$y[n] = \sum_{k=-\infty}^{\infty} h[k] \cdot x[n-k]$$

  - In practice, filters with finite impulse response (FIR) are used. The array of filter coefficients is also known as "filter kernel".

$$y[n] = \sum_{k=-N}^{N} h[k] \cdot x[n-k]$$

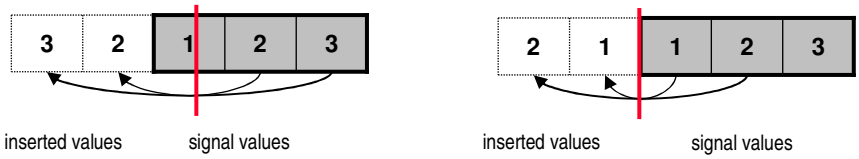
NB: the German word for "convolution" is "Faltung".

JPEG2000
Computing the wavelet transformation by convolution (2)



JPEG2000
Computing the wavelet transformation by convolution (3)

- **Subsampling and convolution combined in a single step**
  - Each second sample in the convolution result is removed in the subsequent subsampling step and is set to zero during inverse DWT. That means, operations involving such samples can be "optimized away"
- **Boundary extension**
  - Problem: Filter kernel may "dangle" about the boundary of the pixel field during convolution
  - Solution: suitable mirroring of the filter coefficients and signal values
  - Depending on the structure of the filter, two different types of boundary extension are used:
    - *sample-symmetric* (left) → e.g. biorthogonal filters, in JPEG2000
    - *boundary-symmetric* (right) → e.g. orthogonal filters, not used in JPEG 2000

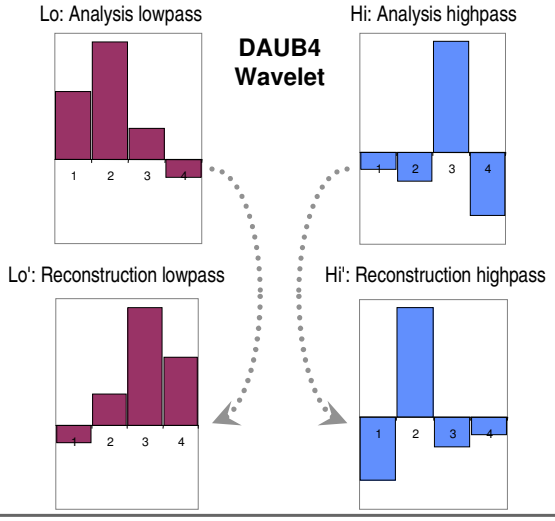


JPEG2000
Excursus: The Haar transformation

$$
\begin{aligned}
i &= 0(1)n/2-1 \\
l_i &= (s_{2i}+s_{2i+1})/\sqrt{2} \\
h_i &= (s_{2i}-s_{2i+1})/\sqrt{2} \\
s'_{2i} &= (l_i+h_i)/\sqrt{2} \\
s'_{2i+1} &= (l_i-h_i)/\sqrt{2}
\end{aligned}
$$

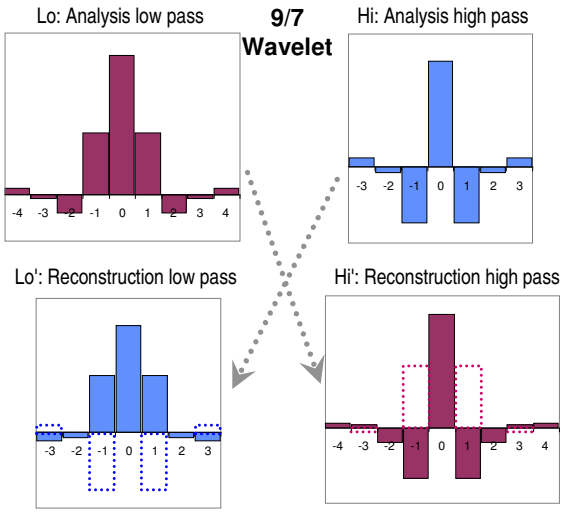
- function similar to a wavelet
- first description by Haar in 1910
- low complexity
- bad rate-distorion properies
- not used for image compression anymore
- however, there are proposals to use the Haar wavelet for temporal decorellation in video compression

JPEG2000
Excursus: Orthogonal wavelet filters



- Daubechies' wavelet family
- unsymmetrical filter kernels
- reconstruction filters are derived by „mirroring" from analysis filters („Quadrature Mirror Filters")
- not used for image compression anymore today, as biorthogonal wavelets offer better rate-distortion properties

JPEG2000
Biorthogonal wavelet filters



The filters in JPEG2000

- symmetric kernels
- reconstruction filter is derived from analysis filter by changing a few signs
- 9/7 wavelet: very good rate-distortion ratio, approximately reversible because of real-valued coefficients
- There exists a shorter filter: 5/3 wavelet – faster to compute, acceptable rate-distortion ratio, reversible because integer-valued coefficients

JPEG2000
Filter coefficients of the 9/7 and 5/3 wavelets

JPEG200 offers a reversible integer mode (5/3 wavelet) and a lossy mode (9/7 wavelet)

Filter	n	0	±1	±2	±3	±4
5/3	Lo	3/4	1/4	-1/8		
	Hi	1	-1/2			
	Lo'	1	1/2			
	Hi'	3/4	-1/4	-1/8		
9/7	Lo	0.60294902	0.26686412	-0.07822327	-0.01686412	0.02674876
	Hi	1.11508705	-0.59127176	-0.05754352	0.09127176	
	Lo'	1.11508705	0.59127176	-0.05754352	-0.09127176	
	Hi'	0.60294902	-0.26686412	-0.07822327	0.01686412	0.02674876

Exercise: Computing the Wavelet Transform

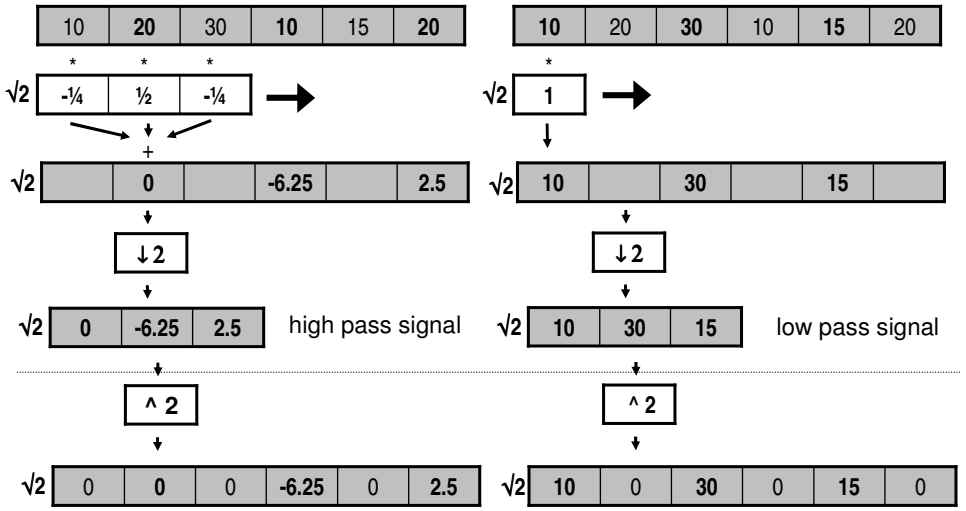
- **by convolution**
- **by Lifting scheme**

JPEG2000
Example: Wavelet transformation by convolution (1)

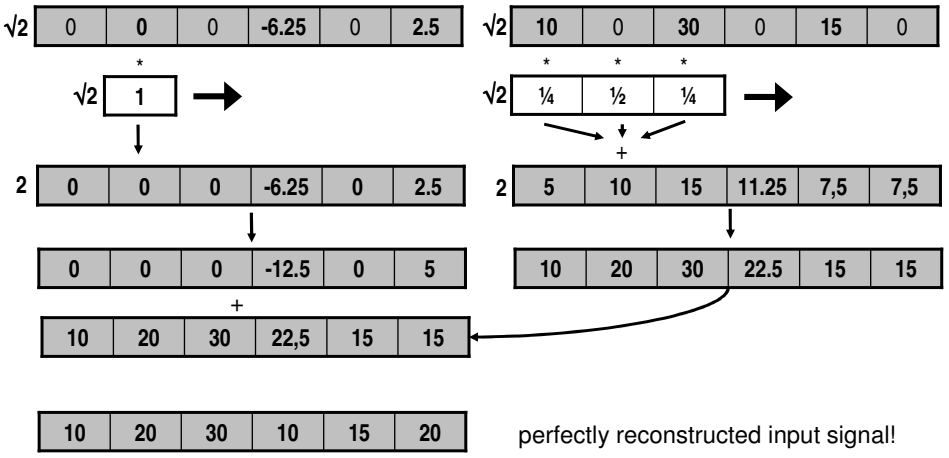
Exercise

- Given: the most simple biorthogonal wavelet (1/3) consisting of the analysis high pass $\sqrt{2}\cdot(-¼, ½, -¼)$ and the analysis-"low pass" $\sqrt{2}\cdot(1)$.
  - just suited as simple example to run the calculations by hand. Its decorrelation properties are just too bad – don't try to use this at home for image compression…
- from this, we can compute the synthesis filters $\sqrt{2}\cdot(1)$ and $\sqrt{2}\cdot(¼, ½, ¼)$

JPEG2000
Example: Wavelet transformation by convolution (2)



JPEG2000
Example: Wavelet transformation by convolution (3)

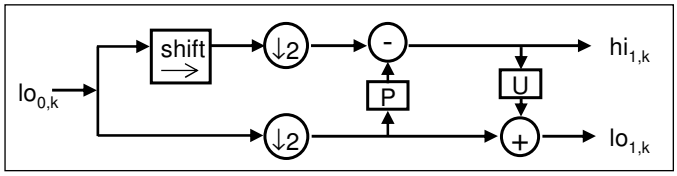


JPEG2000
Lifting – a method for fast DWT implementation

- **Classical implementation: convolution**
  - Disadvantage: each wavelet coefficient is multiplied multiple times with the same filter coefficient
- **Speedup: De-compose the convolution into a series of so-called *lifting operations* [Sweldens]**
  - Advantage: Re-use of intermediate results
- **Approach**
  - Splitting of the signal into 2 channels: even and odd samples
  - Transformation of the channels in a series of steps, using data of the respective other channel, until the signals in the channels meet certain conditions (i.e. represent low pass and high pass signals)
- **Source**

  I. Daubechies and W. Sweldens. Factoring Wavelet Transforms into Lifting Steps. *J. Fourier Anal. Appl.*, 4(3), pp.245-267, 1998. http://cm.bell-labs.com/who/wim/papers/papers.html

JPEG2000
Lifting: Integer wavelet transformation (5/3)



Subsampling („lazy wavelet")

$$lo_{1,k} := lo_{0,2k} \qquad hi_{1,k} := lo_{0,2k+1}$$

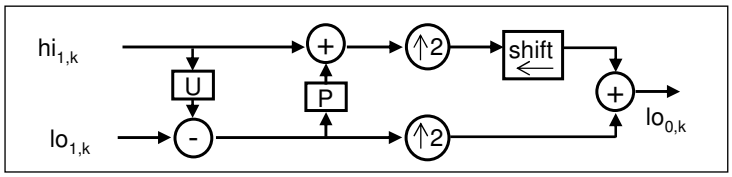
P: Prediction of the „odd" samples by neighboring „even" samples; prediction error becomes the new high pass signal

$$hi_{1,k} -= \left\lfloor \frac{lo_{1,k} + lo_{1,k+1}}{2} \right\rfloor$$

U: Update: Smoothing of the „even" samples to become the low pass signal

$$lo_{1,k} += \left\lfloor \frac{hi_{1,k-1} + hi_{1,k} + 2}{4} \right\rfloor$$

JPEG2000
Lifting: Inverse integer wavelet transformation (5/3)



U: Update: Subtract high pass signal from low pass signal to reconstruct „even" samples

$$lo_{1,k} -= \left\lfloor \frac{hi_{1,k-1} + hi_{1,k} + 2}{4} \right\rfloor$$

P: revert the **Prediction** of the „odd" samples by neighboring „even" samples -- add the „prediction error" (high pass signal)

$$hi_{1,k} += \left\lfloor \frac{lo_{1,k} + lo_{1,k+1}}{2} \right\rfloor$$

Upsample and combine

$$lo_{0,2k} := lo_{1,k} \qquad lo_{0,2k+1} := hi_{1,k}$$

JPEG2000
Excursus: Embedded data streams

Def (Shapiro 1993): An *embedded data stream* is a data stream which contains all encodings of the same image at lower bit rates embedded into the prefix of the data stream for a target bit rate.

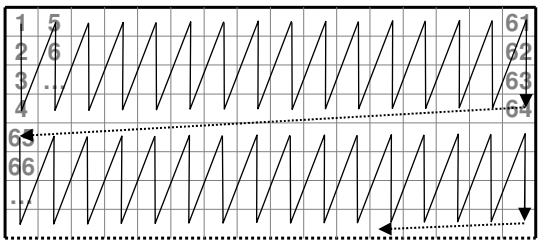
- **What does this mean?**
  - each prefix data stream can be decoded
  - decoding of a prefix data stream reconstructs the image at a lower degree of detail than decoding of the full data stream
  - „degree of detail" can have multiple dimensions (e.g. resolution, „quality") → details later
  - opportunity to differentially refine the image during the transmission process
    1. start
    2. transmit prefix data stream, buffer it, decode it, render image
    3. if target bit rate not reached, goto 1
  - discretization: *Layers* → details later

JPEG2000
Step 4: Context modeling and block coding

- **For JPEG2000, the EBCOT method (Embedded Block Coding with Optimized Truncation) has been developed**
- **Basic idea:**
  - Division of the subbands of the coefficient field into *code blocks* which are encoded independent from each other (*intra-band encoding*)
  - Sub-bitplane wise encoding in 3 passes: Significance, Magnitude refinement, Cleanup
    - this allows *embedded encoding*: Data which yield a large improvement of picture quality are encoded before data which yield a smaller quality improvement
    - the data stream is cut off when the desired data rate or quality (e.g. PSNR) has been reached
  - Context modeling is used to exploit neighborhood relationships (textures)
  - Adaptive arithmetic encoding (MQ Coder)
    - adapt the encoding to the signal statistics
    - limit the output 16bit words to the range 0x0000 .. 0xFF8F (0xFF90 .. 0xFFFF are used for markers to format the data stream)

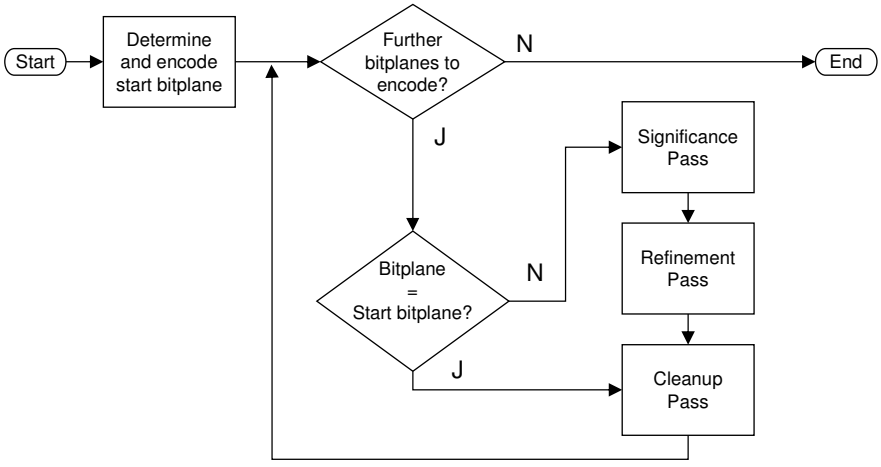
JPEG2000
Block coding: Traversing of the coefficient field

- **Wavelet coefficients of one subband are divided into** *code blocks*
  - The number of most significant bitplanes which do not contain a 1-bit is signaled to the decoder as side information for each code block. Starting from the following bit plane, the current bit of each coefficient is encoded in exactly one of three passes (encoding of "partial bit planes")
    - Significance: encode coefficients which have newly become significant in this bitplane
    - Magnitude refinement: refine coefficients which have already become significant in higher bitplanes
    - Cleanup: encode coefficients which have not been encoded in the first two phases (this is the only pass for the most significant bit plane)
- **Order in which the coefficients of one code block are visited**



one code block:
width: *16*
height: *n (a power of two)*

JPEG2000
Block coding: Flow diagram



Note: When we talk about "encoding" here and in the next slides, this also applies analogously to "decoding".

JPEG2000
Context modeling (1)

- **Contexts characterize such neighbors of a coefficient which may be already known at the decoder side**
  - goal: to exploit neighborhood relationships for creating better symbol statistics
  - for the different passes, different contexts are used
  - 256 contexts are possible. → combine to 17 contexts for faster implementation
- **JPEG2000 discriminates between horizontal, vertical and diagonal neighbors of a coefficient X as follows:**
  - (boundary handling: ignore coefficients outside the current code block)

D_0	V_0	D_1
H_0	X	H_1
D_2	V_1	D_3

Basic algorithm for bitplane Encoding/Decoding

```
context := AnalyzeContext(pass, bit,
                D0, D1, D2, D3, H0, H1, V0, V1)
if(encoding)
     MQCodec.Encode(J2kEncode(pass,bit,X), context)
else
     X[bit] := J2kDecode(pass, bit,
                     MQCodec.Decode(context))
```


JPEG2000
Context modeling (2)

- **Semantics of `AnalyzeContext(pass,bit,D0,D1,D2,D3,H0,H1,V0,V1)`**
  - Analyze the current bitplane of the neighboring coefficients Dx, Hx, Vx
  - Output a context number according to the outcome of this analysis and the current pass
    - → the context numbers are not defined in the standard – each implementation can assign them at will.
    - → we have chosen numbers for use in this lecture which are given in the context tables and referenced in the flow diagrams
- **Semantics of `J2kEncode(pass,bit,X)`**
  - According to the rules of the current pass, the current bit of the current coefficient X is encoded (i.e. it is output as a symbol or a sequence of symbols)
- **Semantics of `MQCodec.Encode(sym,context)`**
  - The symbol (or the sequence of Symbols) is encoded by arithmetic coding. The symbol statistics table to be used for that is determined by the context number.
  - Implementation: for each context number, the encoder maintains a statistics table, containing the occurrence probability of the symbols. After encoding a symbol, the statistics are updated. The context number `context` is the index to identify the table to be currently used.
- **Decoding: analogously**

JPEG2000
Block coding: Significance pass (1)

This pass encodes such coefficients which are becoming significant (i.e. ≠ 0) in the current bitplane

GIVEN THAT

these coefficients have a nonzero probability to be significant, based on the values of their neighbors.

- This means, at least one significant neighbor must be present (context ≠ 0).
- The remaining newly-significant coefficients (context = 0) are kept for the Cleanup Pass.
- For each significant coefficient (with context ≠ 0), a 1-bit followed by the sign is encoded.
- For insignificant coefficients (with context ≠ 0), a 0-bit is encoded.

JPEG2000
Contexts for significance information

LL, LH			HL			HH		Context number
∑H	∑V	∑D	∑H	∑V	∑D	∑H+∑V	∑D	
2	-	-	-	2	-	-	≥3	8
1	≥1	-	≥1	1	-	≥1	2	7
1	0	≥1	0	1	≥1	0	2	6
1	0	0	0	1	0	≥2	1	5
0	2	-	2	0	-	1	1	4
0	1	-	1	0	-	0	1	3
0	0	≥2	0	0	≥2	≥2	0	2
0	0	1	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0

SIGNIFICANCE contexts: The sum of the number of significant coefficients in the neighborhood is used as a model for the significance of the coefficient to be currently coded.

JPEG2000
Contexts for sign coding

sgn[sgn(H_0) + sgn(H_1)]	sgn[sgn(V_0) + sgn(V_1)]	XOR bit	Context number
1	1	0	13
1	0	0	12
1	-1	0	11
0	1	0	10
0	0	0	9
0	-1	1	10
-1	1	1	11
-1	0	1	12
-1	-1	1	13

Encoding

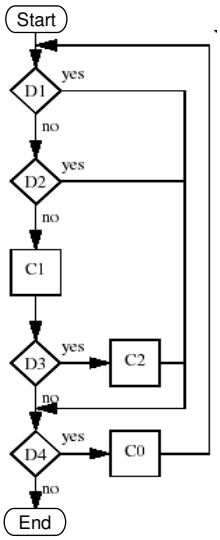
```
if(X<0) signbit := 0
else signbit := 1
encode := signbit XOR XORBit
MQCodec.Encode(signbit, context)
```

Decoding

```
decoded := MQCodec.Decode(context)
signbit := decoded XOR XORBit
```

SIGN contexts: Horizontal resp. vertical sign changes in the neighborhood are used as a model for the sign of the current coefficient. XORBit is used to discriminate different directions of sign changes within the same context.

JPEG2000
Block coding: Significance pass (2)



	Contexts	Description
D1	-	Is the current coefficient significant?
D2	-	Is the context = 0?
C1	1 ... 8 (SIGNIFICANCE)	Encode significance bit: 1 for significant coefficients, 0 otherwise
D3	-	Has the current coefficient just become significant (i.e. has a 1bit been encoded in C1)?
D4	-	Are there further significant coefficients in this pass?
C2	9 ... 13 (SIGN)	Encode the sign of the current coefficient (1 neg, 0 pos)
C0	-	Go to next coefficient / to next column

JPEG2000
Block coding: Magnitude refinement pass (1)

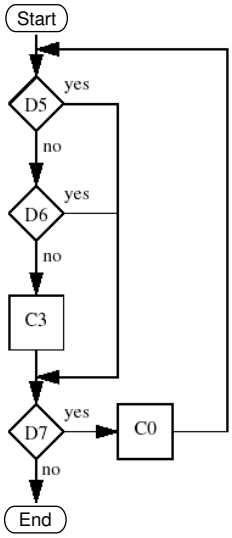
- **This pass refines the precision of coefficients already known as significant by 1 bit.**
- **It exploits correlations between the significance of neighboring coefficients and the value at the second-highest bit plane of the current coefficient**

JPEG2000
Contexts for magnitude refinement

$\sum H+\sum V+\sum D$	First refinement?	Context no.
irrelevant	no	16
≥ 1	yes	15
0	yes	14

REFINE contexts: Statistical relationships between the significance of neighboring coefficients and the second-highest bitplane of the current coefficient are exploited.

JPEG2000
Block coding: Magnitude refinement pass (2)



	Contexts	Description
D5	-	Is the coefficient insignificant?
D6	-	Has the coefficient been encoded/decoded in the immediately preceding Significance pass?
C3	14 … 16 (REFINE)	Encode the bit of the current coefficient in the current bitplane
D7	-	Are there further coefficients in this refinement pass?
C0	-	Go to next coefficient / next column

JPEG2000
Cleanup pass (1)

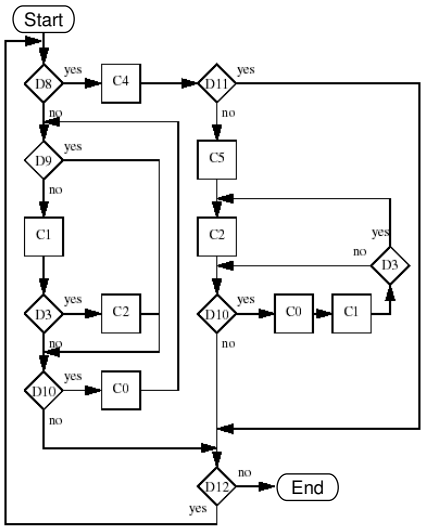
- **The cleanup pass encodes the significance information of all coefficients in a bit plane which have not been encoded in the two earlier passes**
  - → These are those coefficients which have become newly significant but have no significant neighbors (i.e. context = 0)
- **Specialty: for the highest bitplane with nonzero bits, the cleanup pass is the only pass.**
- **Run length encoding**
  - Newly-significant coefficients without significant neighbors are sparsely distributed → ideally-suited for run length coding
  - A 2 bit field encodes for each column of 4 coefficients which is the first significant coefficient.
  - This allows to efficiently encode long runs of insignificant coefficients.
  - Due to the run length encoding, the cleanup pass is the most complex pass.

JPEG2000
Additional cleanup contexts

Two further contexts are used in the cleanup pass

Context no.	Meaning	Description
17	RUN LENGTH	Encodes an all-zero column with 4 coefficients
18	UNIFORM	Encodes data with assumed uniform distribution

JPEG2000
Cleanup pass (2)



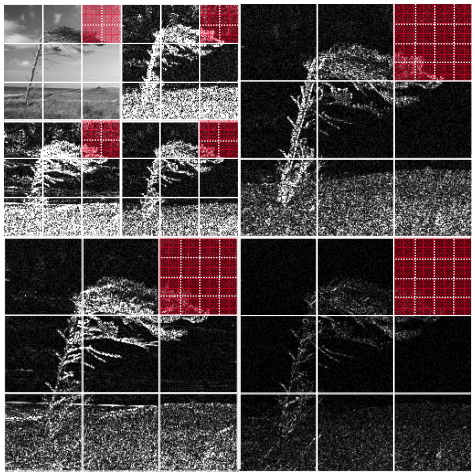
	Context	Description
D8	-	Are there 4 uncoded coefficients in the column, each with a context of 0?
D9	-	Is the current coefficient significant?
C1	0 ... 8 (SIG-NIFICANCE)	Encode significance bit: 1 for significant coefficients, else 0
D3	-	Has the current coefficient just become significant?
D10	-	Are there further coefficients in the column?
C4	17 (RUN LENGTH)	Encode run length of 4 zeroes (0) or no run length of 4 zeroes (1)
C2	9 ... 13 (SIGN)	Encode the sign bit of the current coefficient (1 neg, 0 pos)
C0	-	Go to next coefficient / to next column
D11	-	Is the current bit =0 for all the 4 coefficients in the column?
C5	18 (UNIFORM)	Encode the position of the first nonzero coefficient in the column as 2bit binary number
D10	-	Are there further coefficients in the column?
D12	-	Are there further coefficients in the cleanup pass?

JPEG2000
Step 5: Layer generation

- **The EBCOT encoding has generated an embedded data stream which is formatted in a next step into so-called *layers*.**
  - A layer is a unit in the data stream which leads to a defined enhancement of the decoded image.
  - Layer boundaries can only be at defined positions in the data stream.
  - The packetization of the data stream allows flexible access to the layers. If needed, packets can be re-sorted in order to create a different sequence of layers (see “progression orders” later).
- **Tier1 and tier2 encoding**
  - Tier 1: EBCOT (resulted in an embedded data stream, decorated with block and pass numbers)
  - Tier 2: formatting of the encoded data into layers
    - Flexible packetization allows different progression orders for different applications (e.g. Zoom: refine resolution; Browse: refine SNR)
    - A rate-distortion optimization can be performed by inserting the encoded data blocks in such an order into the resulting data stream, that blocks which contribute a larger distortion reduction (at same data rate) are inserted first.

JPEG2000
Layer generation: Precincts

- **Idea**
  - For random access to image parts, the image can either be divided into *tiles* (already introduced) or *precincts*.
- **Combining code blocks to precincts**
  - A precinct consists of one or more code blocks which contribute to the same image area.
  - A precinct is bounded to one sub band.
- **Precinct size**
  - varies from sub band to sub band
  - dimensions are powers of two

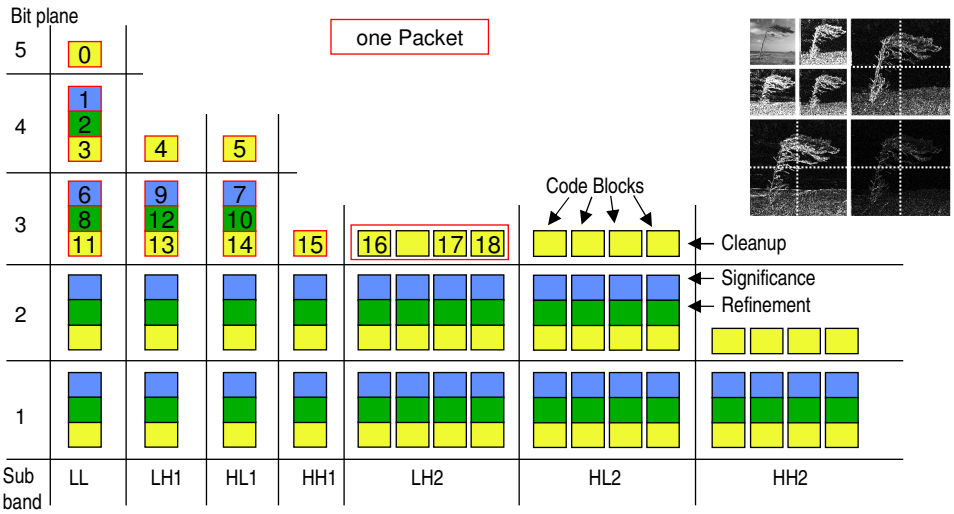


Example configuration.
Dotted lines delimit blocks. Each red area = one precinct.

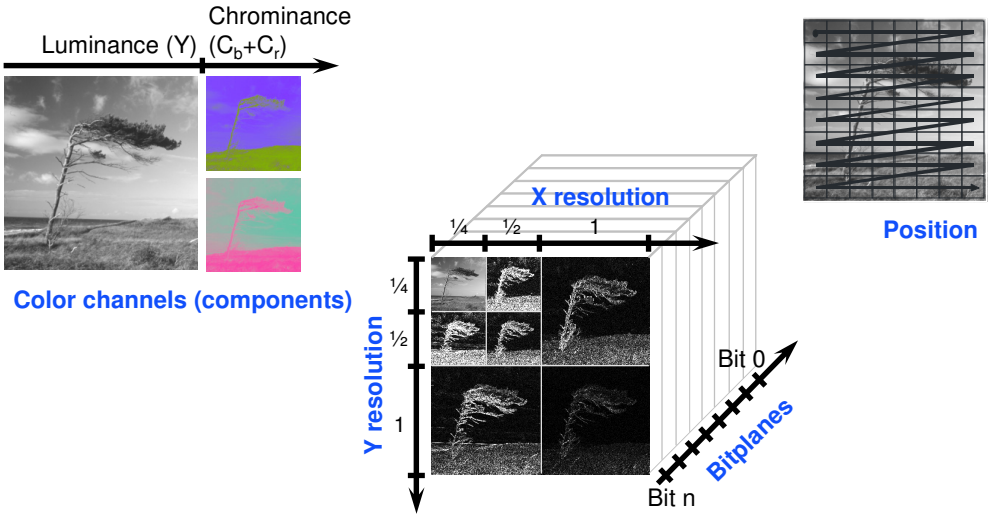
JPEG2000
Layer generation: Packets

- **One layer consists of multiple packets.**
  - a packet contains the codestream portion contributing to one color component, partial bitplane, sub band and precinct
  - a packet may be empty.
- **Packet header**
  - the packet header signals all information needed to know the contents of the packet
    - codeblocks included
    - empty packet yes/no
    - number of “all zero” most significant bit planes
    - number of coding passes per code block
    - length of coded data per code block
- **Per region, zero or more partial bit planes are put into one layer**
  - region=code block: use codeblock inclusion info in packet header
  - region=precinct: empty packets can be used to skip precincts.

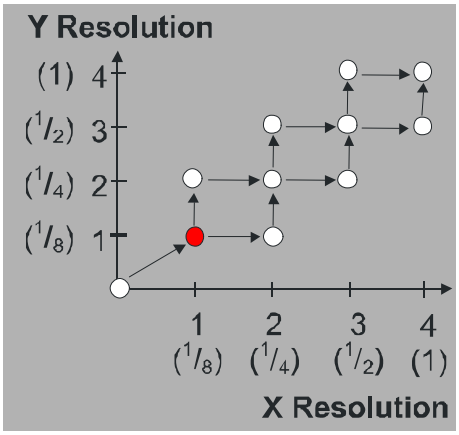
JPEG2000
Layer generation: Illustration



JPEG2000
Possibilities of building layers



JPEG2000
Layers are a partially ordered set

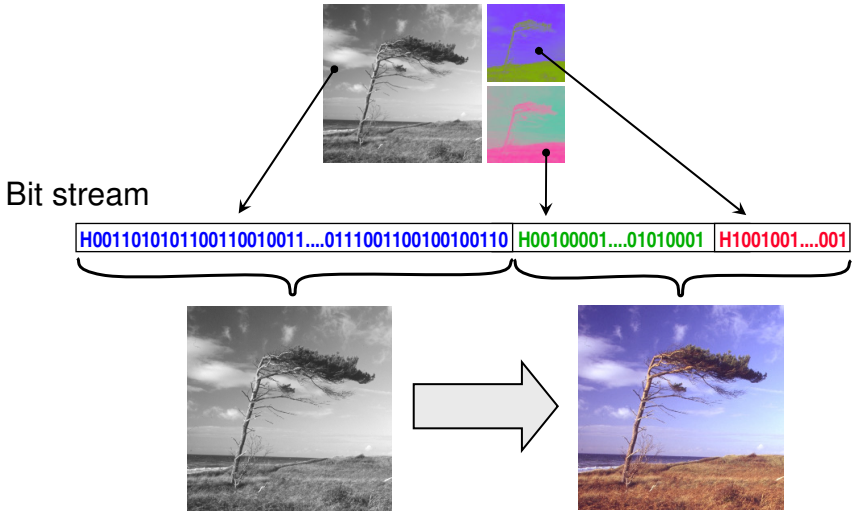


- **Layers are building on top of each other**
  - to decode a layer, other layers are needed
- **This defines a partial order**
- **Example: X and Y resolution in a wavelet representation**
  - First layer (without predecessor) is called "base layer" (marked red)

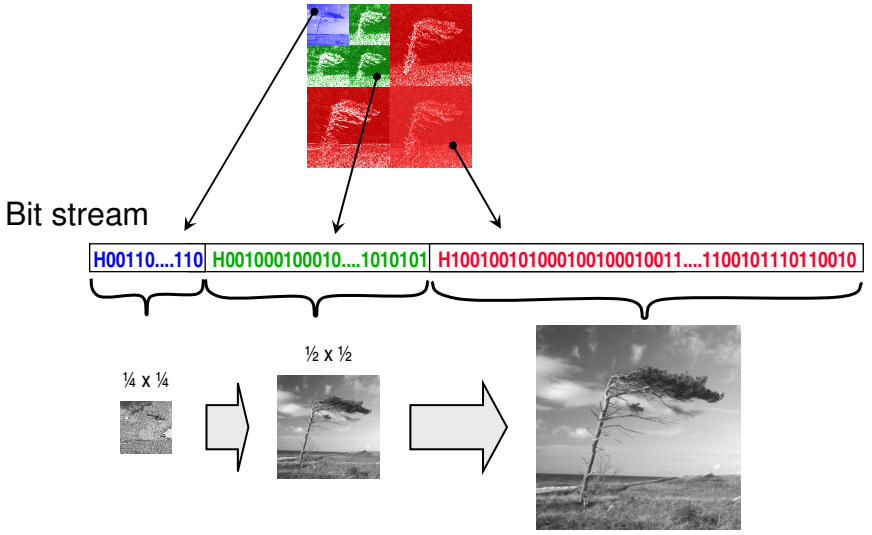
JPEG2000
Progression orders according to the JPEG2000 standard

- **The coefficient field is traversed in nested for loops.**
  - nesting is defined by the chosen progression order.
  - nesting can be signaled per tile.
- **Allowed progression orders**
  - Bitplane-resolution-component-position: for(bitplane) for(subband) for(component) for(pos)
    → progressive quality enhancement
  - Resolution-bitplane-component-position
    → progressive resolution enhancement with step-wise quality enhancement
  - Resolution-position-component-bitplane
    → progressive resolution enhancement
  - Position-component-resolution-bitplane
    → image builds up form top left to bottom right
  - Component-position-resolution-bitplane
    → first grayscale image, then color image

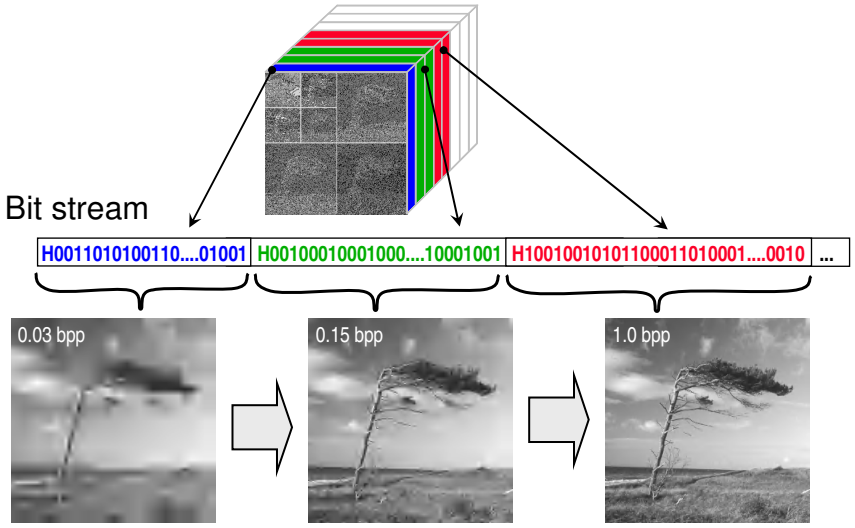
JPEG2000
Example: Component-progressive bit stream



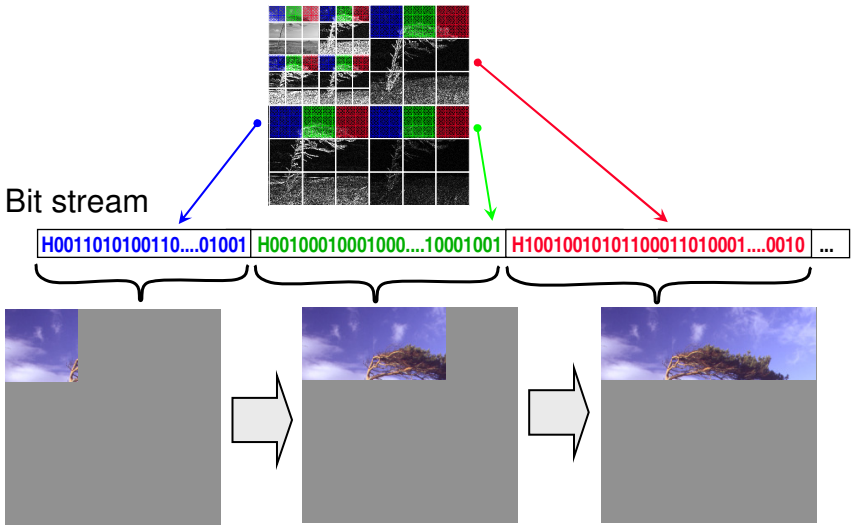
JPEG2000
Example: Resolution-progressive bit stream



JPEG2000
Example: Bitplane-progressive bit stream



JPEG2000
Example: Position-progressive bit stream

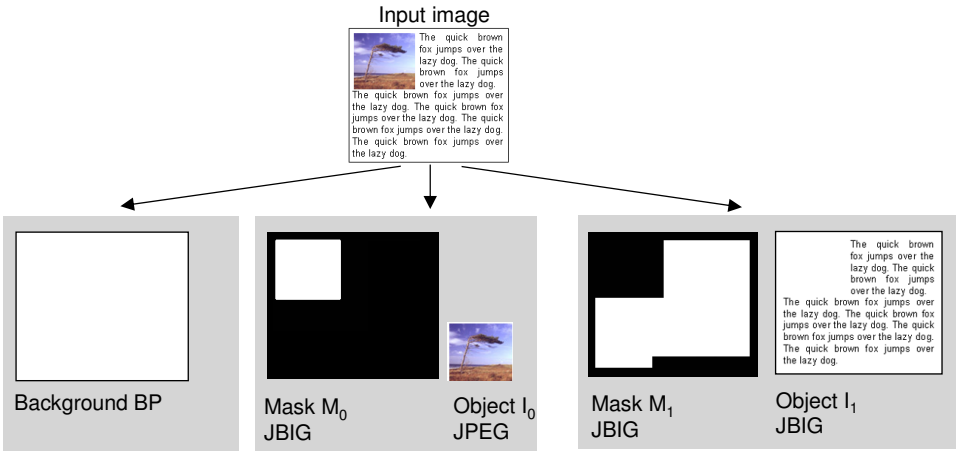


JPEG2000
JPEG2000 Part 6 – Compound Images

- **Situation**
  - different image compression methods are differently well-suited for different image classes (e.g. JPEG/JPEG2000 for photographic images, PNG for graphics, JBIG2 for bi-level images)
  - all these classes may be combined in a single image, e.g. in color scans
  - such images are called MRC (Mixed Raster Content, ISO/IEC 16485)
- **Basic idea**
  - use the best-suited compression method for each image region
  - store additional mask information to combine the parts (binary or alpha masks)
- **JPEG2000 part 6 defines a file format for such content (JPM)**
  - basis: JP2 and JPX file formats
  - support for multi-page documents
  - supported compression methods for image objects: JPEG, JBIG2, JPEG2000, JPEG-LS
  - supported compression methods for mask objects: Fax G.3/G.4, JBIG, JBIG2, JPEG2000

JPEG2000
Mixed Raster Content

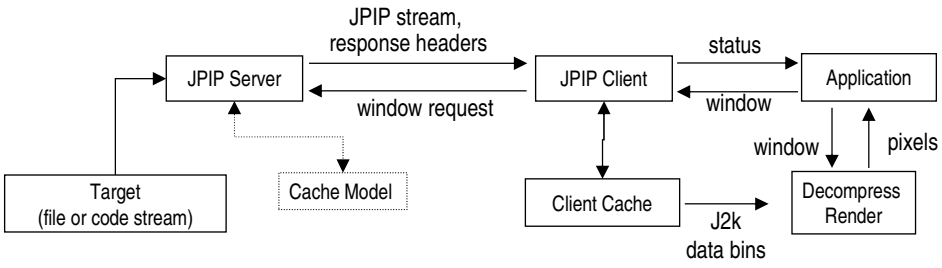
$$PageImage = BP + M_0 \times I_0 + M_1 \times I_1 + M_2 \times I_2 + \ldots$$



JPEG2000
JPEG2000 Part 9 – Interactive Protocol (1)

JPEG2000 Part 9 defines a protocol (JPIP) for random access to coded JPEG2000 data streams

- main application: image data bases
- flexible access: one JPEG2000 file on the server serves image regions at different resolution and/or quality levels from a single encoded data stream over the network, without re-encoding the image
- all requests relate to a part of the image – the *„Focus Window“*
- server decides the optimum sequence of the data to be transmitted



JPEG2000
JPEG2000 Part 9 – Interactive Protocol (2)

Example session with JPIP

1 whole image is transmitted over a slow connection
2a user is especially interested in the name of the boat and defines an according region
2b data for this region are transmitted with precedence, leading to fast refinement
3 after the region of interest is fully available, data transmission for the remaining parts continues

JPEG2000
JPEG2000 Part 9 – Interactive Protocol (3)

Parts of a client request

- mandatory
  - **fsiz:** size of the requested image
    → server derives the needed scaling factor from fsiz and from the image size in its database
  - **rsiz:** size of the *Focus Window*, in relation to fsiz
  - **roff:** offset of the *Focus Window*, in relation to fsiz
- optionally in addition
  - color components
  - number of quality layers to transmit
  - explicit request of parts of the data stream

JPEG2000
JPEG2000 Part 9 – Interactive Protocol (4)

Basis protocols: HTTP, TCP

- **Standard methods:** HTTP GET or POST Requests
  - Request e.g.

    ```
    http://host.jpeg.org/images/kids.jp2?rsiz=640,480&roff=320,240&fsiz=1280,1024
    ```

  - Response of the server: Status in the header, Data in the body of the HTTP reply, e.g.

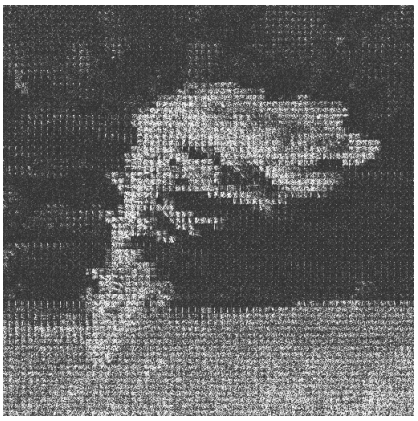
    ```
    HTTP/1.1 200 OK
    JPIP-fsiz: 6000,8000
    JPIP-rsiz: 300,200
    JPIP-roff: 2500,3000
    Content-type: image/jpp-stream
    (JPIP Response)
    ```

- **Alternative:** HTTP GET to control; transmission of the requested data over an additional TCP connection
  - Response of the Server: Status in the header of the HTTP reply, Data over TCP

JPEG2000
JPEG2000 Part 9 – Interactive Protocol (5)

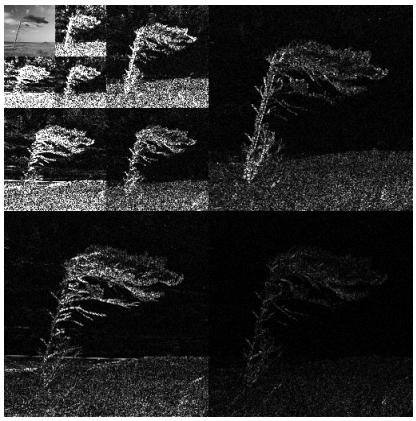
- **Stateless vs. stateful operating mode**
  - JPIP allows the refinement of image areas without re-transmitting data already sent
  - **stateless mode:** Client maintains its state and sends it to the server along with each request
  - **stateful mode:** Server can store the state for a session. In this case, a *cache model* of the already transmitted data is stored
- **JPIP defines commands to**
  - access image data for a region
  - access metadata
  - control the server's cache model (stateful sessions only)
  - transmit client capabilities
  - upload images
- **Interactive image browsing without JPIP**
  - JPEG2000 part 9 also defines an index structure as part of a JPEG2000 file, which can be used to browse this image without a JPIP server component, using plain HTTP 1.1 range requests

JPEG2000 vs. JPEG
Coefficient fields of DCT and DWT



DCT

8x8 blocks of DCT coefficients



DWT

Hierarchical subbands of wavelet coefficients

JPEG2000 vs. JPEG
Comparison based on some criteria

	JPEG	**JPEG2000**
Quality: low bit rates	bad	acceptable
Quality: medium bit rates	good	good
Region of interest	-	yes
Scalability: resolution	-	yes
Scalability: SNR resp. quality	possible	yes
Scalability: color components	yes	yes
Artefacts	„Blocks“	„Smoothing“

Multimedia Coding
Part 3: Still Image Coding

GIF
Overview

- **Graphics Interchange Format, Compuserve, 1987/1989**
  - today, GIF87 is not used anymore but GIF89 has spread throughout the Web
- **Image format for colormapped images (max. 256 colors)**
  - (using local colormaps, also true color GIFs are possible. These are not memory-efficient, though.)
- **Features**
  - global and local colormaps
  - interlacing to speed up image display when accessing online services over slow links
  - animation support (full frames as well as image blocks)
  - one color can be marked as transparent (“Magic Color”)
  - data compression using LZW

GIF
LZW compression (1)

- **The LZW compression algorithm is the foundation and the curse of GIF.**
- **Basic idea**
  - use a dictionary to store strings which occur in the input stream for later referencing
  - if a string repeats (i.e. it is in the dictionary), a reference (index) is output instead of the string
  - the dictionary is initialized to contain all possible strings of length 1
  - then, on processing, each string which is not yet in the dictionary is appended, starting with strings of length 2 and continuing to greater lengths
- **Patent situation**
  - the LZW algorithm has been patented by UNISYS.
  - the patent has expired in 2003 in U.S. and in 2004 in Europe, Japan and Canada.
  - it shows the danger software patents impose on IT infrastructures: The patent has been announced very late, when the format was already widespread. Depending on how critical the role of the patented technology is, license costs can endanger the whole infrastructure.
  - the patent situation was a major reason for the development of PNG.

GIF
LZW compression (2)

Compression algorithm

```
[1] Initialize the dictionary with all possible strings of length 1
      („root codes“)
[2] Initialize current prefix prefix with „“
[3] Read next character K from input stream
      Exception handling: if EOF, output code for prefix and exit.
[4] Is prefix+K in the dictionary?
      [4a] yes
            prefix:=prefix+K
            go to [3]
      [4b] no
            append prefix+K to dictionary
            output code for prefix
            prefix:=K
            go to [3]
```

GIF
LZW compression (3)

Decompression algorithm

```
[1] Initialize the dictionary
[2] Read first code from input stream into variable C
[3] Read string for C from dictionary and output it
[4] old := C
[5] Read next code from input stream into variable C
      Exception handling: If EOF, exit.
[6] Does an entry for C exist in the dictionary?
      [6a] yes
            Read string for C from dictionary and output it
            prefix := dictionary entry for old
            K := first character of the dictionary entry for C
            Append prefix+K to the dictionary
      [6b] no
            prefix := dictionary entry for old
            K := first character from prefix
            Output prefix+K and append it to the dictionary
[7] go to [4]
```


GIF
LZW compression (4)

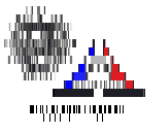
GIF extensions to LZW

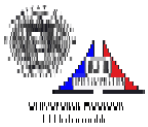
- **Dictionary initialization:** the dictionary is initialized with 2^N+2 root codes, where 2^N denotes the size of the colormap (number of colors in the image). Each root code between 0 and 2^N-1 corresponds to an index into the colormap. Additionally, two control characters CC and EOI with the codes 2^N and 2^N+1 are used (see below)
- **Variable code length:** the length $N+1$ of the codes depends on the actual number of dictionary entries. If this number reaches the value 2^N-1 upon appending a new entry, N is increased by 1. The initial code length is the color depth N + 1 bit , the maximum length 12 bits
- **Handling of dictionary overflows:** If the dictionary contains more than 4095 entries it is newly initialized. To signal this, a special control character „CC“ (clear code) is sent. For consistency, this is also the start character in the stream.
- **End of image:** is signaled by a second character (EOI).

GIF
Interlacing scheme

- **Purpose of interlacing**
  - display of intermediate steps during progressive transmission
- **Idea: 4 step transmission**
  - transmit every 8th row first
  - then every missing 4th row
  - then every missing 2nd row
  - then every missing row

1	1	1	1	1	1	1	1
4	4	4	4	4	4	4	4
3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4
2	2	2	2	2	2	2	2
4	4	4	4	4	4	4	4
3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4



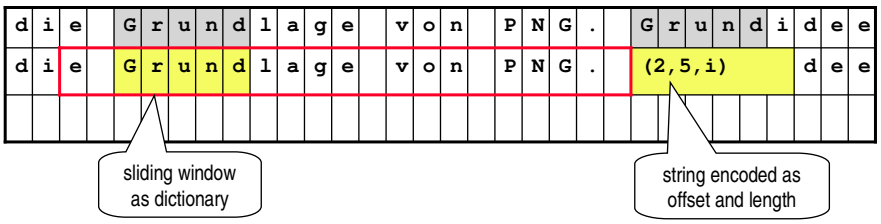


PNG
Overview

- **Portable Network Graphics, 1987/1989**
  - developed by the Internet community as an open source and patent-free alternative for GIF
  - pronounced “ping”
- **Design goals**
  - avoid the LZW patent
  - support colormapped images and true color images; plus additional transparency channel (alpha)
- **Features**
  - encoding of colormapped images, transparency by Magic Color → replacement for GIF
  - encoding of true color images with different color depth, transparency by alpha channel
  - interlacing for progressive refinement in online services
  - animation support in the sister format MNG [pronounced “ming”]
  - data compression uses the patent-free deflate method (LZ77 plus Huffman-Kodierung, like in gzip)

PNG
LZ77 compression (1)

- **The LZ77 compression method is the foundation of PNG.**
- **Basic idea**
  - The method is based on a dictionary which stores strings for referencing them in case they re-occur.
  - In contrast to LZW, the already read/decoded data stream is used as the dictionary; references consist of an offset into this dictionary relatively to the current position in the data stream, plus a length
  - To be able to use fixed-length references, a sliding window of fixed size is used as the dictionary.



PNG
LZ77 compression (2)

Compression algorithm

```
[1] Set codingPosition := 0; windowPointer := 0; length := 0;
[2] Find the longest string in window which matches the head of the
    input stream
       [2a] if such a string exists, output pair (index, length);
       [2b] otherwise output nothing and set length := 0;
[3] output first non-matching character of input stream
[4] codingPosition += length + 1; windowPointer += length + 1;
[5] if characters are remaining in the input stream go to [2]
```

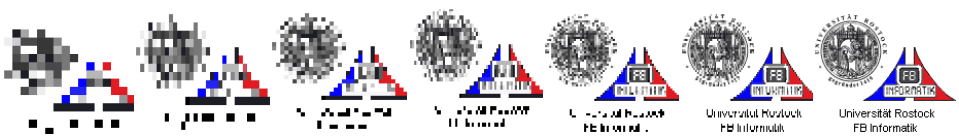
PNG
Deflate compression

- **Deflate encodes the output of LZ77 using Huffman coding. The input data are processed in blocks.**
- **3 modes (applied per block)**
  - uncompressed (to avoid expansion, e.g. for data already compressed)
  - compressed with standard Huffman table
  - compressed with own Huffman table which is transmitted as side information
- **Alphabet of Huffman coding**
  - 256 characters („literals“)
  - all possible lengths
    - the offset is not part of the alphabet, but is written uncoded after the length into the stream (as offsets are assumed to be uniformly distributed)
  - end-of-block symbol

PNG
Interlacing scheme

- **Purpose of interlacing**
  - display of intermediate steps during progressive transmission
- **Idea: 7 step transmission, applied for each 8x8 block**
  - transmit 1st pixel of every 8th row
  - then 5th pixel of every 8th row
  - then 1st and 5th pixel of every missing 4th row
  - then 3rd and 7th pixel of every 4th row
  - then every 2nd pixel of every missing 2nd row
  - then every pixel of every 2nd row
  - then the missing rows

1	6	4	6	2	6	4	6
7	7	7	7	7	7	7	7
5	6	5	6	5	6	5	6
7	7	7	7	7	7	7	7
3	6	4	6	3	6	4	6
7	7	7	7	7	7	7	7
5	6	5	6	5	6	5	6
7	7	7	7	7	7	7	7



Further methods for bi-level images

- **Fax G4**
  - developed for the lossless compression of fax images (bi-level)
- **JBIG (Joint Bi-level Image Group)**
  - ITU/ISO standard (ITU-T.82; ISO 11544) for the compression of bi-level Images (black&white)
  - lossless, resolution-progressives
  - Ingredients: prediction, context modeling, arithmetic coding
- **JBIG2**
  - ITU/ISO standard (ITU T.88; ISO/IEC 14492) for the compression of bi-level Images (black&white), compresses 2-4 times better than JBIG
  - lossless or near-lossless
  - Basic idea:
    - detect repeating patterns of pixels (“symbols”, e.g. letters) and store them in a symbol library
    - represent the picture as references to these patterns
    - relaxed similarity criteria in matching the patterns lead to higher, lossy compression (vector quantization!)
  - used in PDF V 1.4 and above

Further methods for true-color images (1)

- **Lossless JPEG**
  - the JPEG standard also defines a lossless mode which is **not** based on the DCT
  - Ingredients: Prediction, arithmetic coding
  - rarely used
- **JPEG-LS**
  - ISO-Standard: ISO IS 14495 for the lossless or near-lossless compression of true-color images
  - Basic Idea:
    - Context modeling and prediction
    - Gradient detector to select predictor and context
    - Entropy coder: Rice-coding
  - Comparison with JPEG2000
    - if lossless compression is needed, JPEG-LS is significantly faster than JPEG2000, at similar compression ratio
    - JPEG-LS is a plain compression method, missing the rich feature set of JPEG2000 to support mobile and interactive applications

Further methods for true-color images (2)

- **JPEG-XR**
  - currently (2007/2008) under specification by JPEG
  - based on Microsoft's HD Photo (Windows Media Photo) technology, specifically developed to support digital imaging
  - Features
    - colour depth up to 32 bits per channel
    - "Photo Core Transform": block-based, lossless, fast, block size 4x4, 2x2
    - adaptive traversal, flexible DC prediction, adaptive Huffman coding
  - Claim: Image quality as of JPEG2000 at computational complexity of JPEG

Image file formats

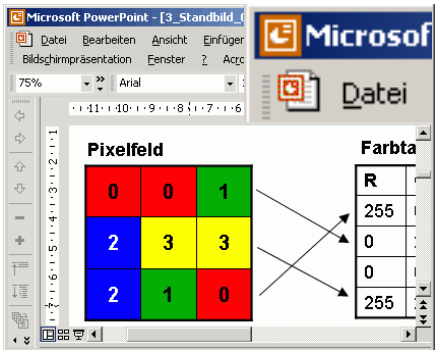
- **File formats describe containers for image data and are not compression methods!**
- **JFIF – JPEG File Image Format**
  - Image file format for JPEG images
  - Restricts JPEG to a subset used by most applications (e.g. only YC_bC_r color space)
  - Adds data (resolution, thumbnail, EXIF metadata)
- **TIFF – Tagged Image File Format**
  - may contain uncompressed or compressed image data (LZW, Fax, JPEG) in different color depths and with different color formats
  - one picture or multiple pictures per file
  - TAGS describe the embedded picture data
  - extensible by private and new tags *(„thousands of incompatible file formats")*

Multimedia Coding
Part 3: Still Image Coding

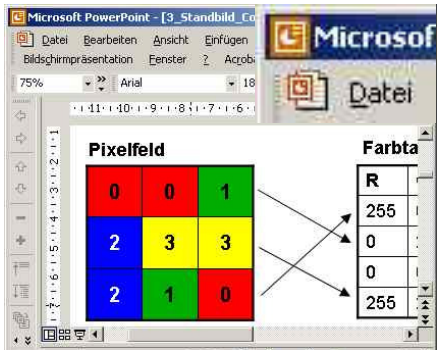
Image compression in practice
Lossless vs. lossy compression (1)

For images with just a few colors and hard edges (screenshots, business graphics), a lossless method for colormapped images like GIF or PNG is often better suited than a lossy method like JPEG or JPEG2000!

- Examples: screen shots, business graphics



GIF, 64 colors, 16010 bytes, 0.77 bpp



JPEG, q=35%, 24569 bytes, 1.19 bpp

Image compression in practice
Lossless vs. lossy compression (2)

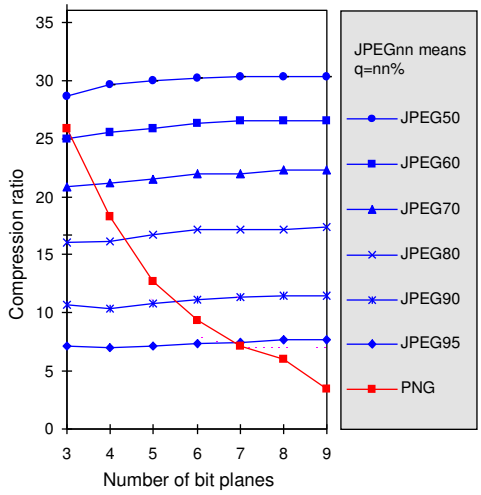
For photographic images, JPEG/JPEG2000 are better suited than GIF or PNG!

- Color quantization in GIF is a lossy operation → "banding" in color gradients
- If lossless compression is required, use PNG, LZW-TIFF or JPEG-LS

GIF, 64 colors, 21416 bytes, 2.61 bpp

JPEG, q=75%, 11001 bytes, 1.34 bpp

Image compression in practice
Lossless vs. lossy compression (3)



Number of colors vs. compression ratio

- The compression ratio of JPEG is (averaged over a set of images) independent from the number of colors; it just depends on "q" parameter setting
- The compression ratio of PNG and GIF decreases as the number of colors increases

Multimedia Coding
Part 3: Still Image Coding

Software tools

- **netpbm**
- **ImageMagick**
- **Independent JPEG**
- **Kakadu**
- **LuRaWave**
- **libTIFF**
- **libPNG**

Software tools: netPBM Overview

- **Idea behind netPBM: create a very simple image format as „Intermediate language“ for image processing**
  - image processing with command line tools possible
  - 3 image classes, each can be stored in ASCII or binary format
    - bilevel (portable bitmap, PBM, type code P1/P4)
    - greylevel (portable graymap, PGM, type code P2/P5)
    - 24bit RGB (portable pixmap, PPM, type code P3/P6)
  - File structure
    - Header (separated by whitespace)
      - Type code (Pn; 1<=n<=6)
      - num_rows num_cols
      - maximum pixel value per channel
      - linefeed
    - Pixel values: ASCII numbers separated by whitespace if ASCII format is used (1<=n<=3); binary values otherwises

Software tools: netPBM Components

- **Tools**
  - Software library to read and write the format
  - Large variety of command line tools fo specific functions
    - import and export filters
    - image processing filters
    - complex image processing tasks can be solved on command line using pipes
    - „speaking names“ of the tools, e.g. giftoppm or pnmscale
  - PNM: Portable AnyMap; no image format but an „interface placeholder“
    - „pnm“ in the name of a tool means that it reads/writes PPM, PGM and/or PBM
- **Example**

  giftoppm picture.gif | pnmscale –xscale 0.5 –yscale 0.5 | ppmquant 256 | ppmtogif > scaled.gif

  → reads a GIF image, scales it (with interpolation), reduces the number of colors by color quantization, writes result as GIF
- **Source**
  - http://netpbm.sourceforge.net
  - Open Source

Software tools: ImageMagick

- **Portable library for image conversion and image compression**
- **Comparison with netPBM**
  - Similarly powerful but different approach
  - Library approach instead of toolchain approach; libraries available for C/C++, Java, Perl, PHP, …
  - Small number of command line tools vailable for
    - Converting,
    - Filterin,g
    - Combining of images
  - Pixel field is kept in memory instead of being transferred to the next tool via pipes and file I/O
  - NetPBM: better suited for prototyping and server-side scripts
  - ImageMagick: better suited to be compiled into software
- **Source**
  - http://www.imagemagick.org/
  - Open Source

Software tools: Independent JPEG

- **The free JPEG codec of the Independent JPEG Group is the most widely used JPEG implementation today**
  - it has greatly facilitated the adoption of JPEG
- **Components**
  - libjpeg: portable library supporting the baseline and progressive modes of JPEG
  - cjpeg: command line encoder
    - input PNM, output JFIF
    - Control of quantization via „q“ parameter – scaling the quantization tables
  - djpeg: command line decoder
    - input JFIF, output PPM
- **Source**
  - http://www.ijg.org/
  - Open Source
  - Current version is 6b

Software tools: Kakadu

- **JPEG2000 implementation of David Taubmann, inventor of EBCOT and one of the authors of the JPEG2000 standard**
  - C++ library and example applications; Java Native Interface available
  - largely platform independent
  - implements part 1, JPIP and the file formats JP2, JPX, MJ2
- **Tools**
  - kdu_show: Browser for JPEG2000 and MJPEG2000 as well as Client for JPIP
  - kdu_server: Server for JPIP
  - kdu_hyperdoc: Tool to create documentation and JNI classes from the C++ headers
  - kdu_{v_}compress / kdu_{v_}expand: Encoder/Decoder for still images and video
  - kdu_transcode: Transcoder (lossless operations in the compressed domain)
- **Source**
  - http://www.kakadusoftware.com/
  - License model for source code, non-commerciel licenses from ca. 150 EUR

Software tools: LuRaWave / LuRaDocument

- **Commercial implementation of JPEG2000 (LuRaWave) and JPM (LuRaDocument) from LuRaTech**
- **Componenten**
  - Decoder as standalone application and plugin for web browsers, Irfanview and Photoshop
  - Encoder als command line applicationo, Photoshop plugin or standalone GUI application
  - Libraries (C/JNI) available
- **Source**
  - http://www.luratech.de/
  - commercial software
  - decoder freeware (plugin for web browsers)

Software tools: libTIFF, libPNG

- **libTIFF**
  - http://www.libtiff.org/
  - reads and writes TIFF files
  - Open Source
  - implemented in C
- **libPNG**
  - http://www.libpng.org/
  - reads and writes PNG files
  - Open Source
  - implemented in C

Multimedia Coding
Part 3: Still Image Coding

Further information (1)

- **Color systems, Color quantization, Dithering**
  - J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics – Principles and Practice, Second Edition*. Addison-Wesley, 1990. ISBN 0-201-12110-7.
- **Image coding (in German)**
  - Tilo Strutz. *Bilddatenkodierung*. Vieweg Verlag, 2002.
  - Rauschenbach, U.: "Bedarfsgesteuerte Bildübertragung mit Regions of Interest und Levels of Detail für mobile Umgebungen", Dissertation, Universität Rostock, Ingenieurwissenschaftliche Fakultät, Mai 2000. http://wwwicg.informatik.uni-rostock.de/~urausche/MoVi/Publications/dissRauschenbach/
- **JPEG**
  - W.B. Pennebaker and J.L. Mitchell. *JPEG Still Image Data Compression Standard*. Van Nostrand Reinhold, New York, 1993.

Further information (2)

- **JPEG2000**
  - David S. Taubman and Michael W. Marcellin. *JPEG 2000: Image Compression Fundamentals, Standards and Practice*. Kluwer International Series in Engineering and Computer Science, Secs 642, Kluwer Academic Publishers, 2001.
  - Signal Processing: Image Communication, Special Issue on JPEG 2000, Elsevier Science, 17(1), January 2002
  - Official JPEG2000 page of the JPEG Committee: http://www.jpeg.org/jpeg2000/index.html
- **PNG**
  - http://www.png.org/pub/png