Image Data Compression

Image compression formats

We use many different image and video formats!

Standards of single image compression

Many common computer image file formats: PNG, BMP, JPG, TIFF, RAW, PSD, BPG, …

ISO standards:

- Joint Photographic Experts Group (JPEG)
  - JPEG [ISO/IEC 10918]
  - JPEG-LS [ISO/IEC 14495]
  - JPEG 2000 [ISO/IEC 15444]
- Joint Bi-level Image Experts Group (JBIG)
  - JBIG [ISO/IEC 11544]
  - JBIG-2 [ISO/IEC 14492]

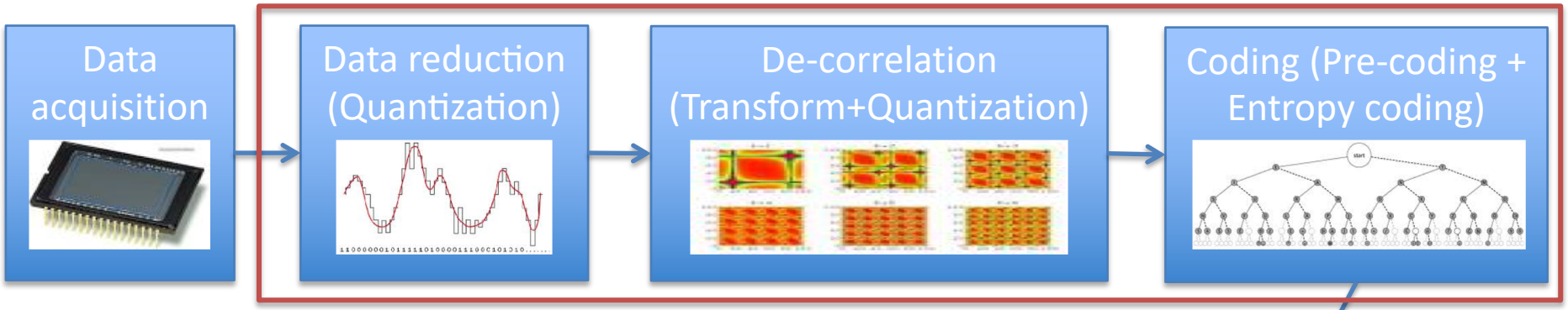
International Organization for Standardization

ISO standards define baseline and extended versions, provide implementation guidelines, specify alternative algorithms, reference implementations, standard tables, file storage formats, …

Many ways to classify algorithms:

- **Treating irrelevant information:** lossy / lossless / near-lossless
- **Treating color channels:** multi-color / gray-level / bi-level
- **Data-dependent adjustments:** adaptive / non-adaptive encoding
- **Size vs quality trade-off:** flexible quality parameterization
- **Simultaneous decoding while loading:** progressive / hierarchical / sequential
- **Primary purpose:** compression / delivery / storage
- …

Image signal path: overview



Output data:
Continuous multi-dimensional analog signal

Compression:
Physical sensor limitations, DAQ settings

Output data:
Correlated fully digital multi-dimensional symbol stream

Compression:
Spatio-temporal quantization, value quantization

Output data:
Linear stream of de-correlated symbols

Compression:
Truncation of irrelevant signal parts, value quantization

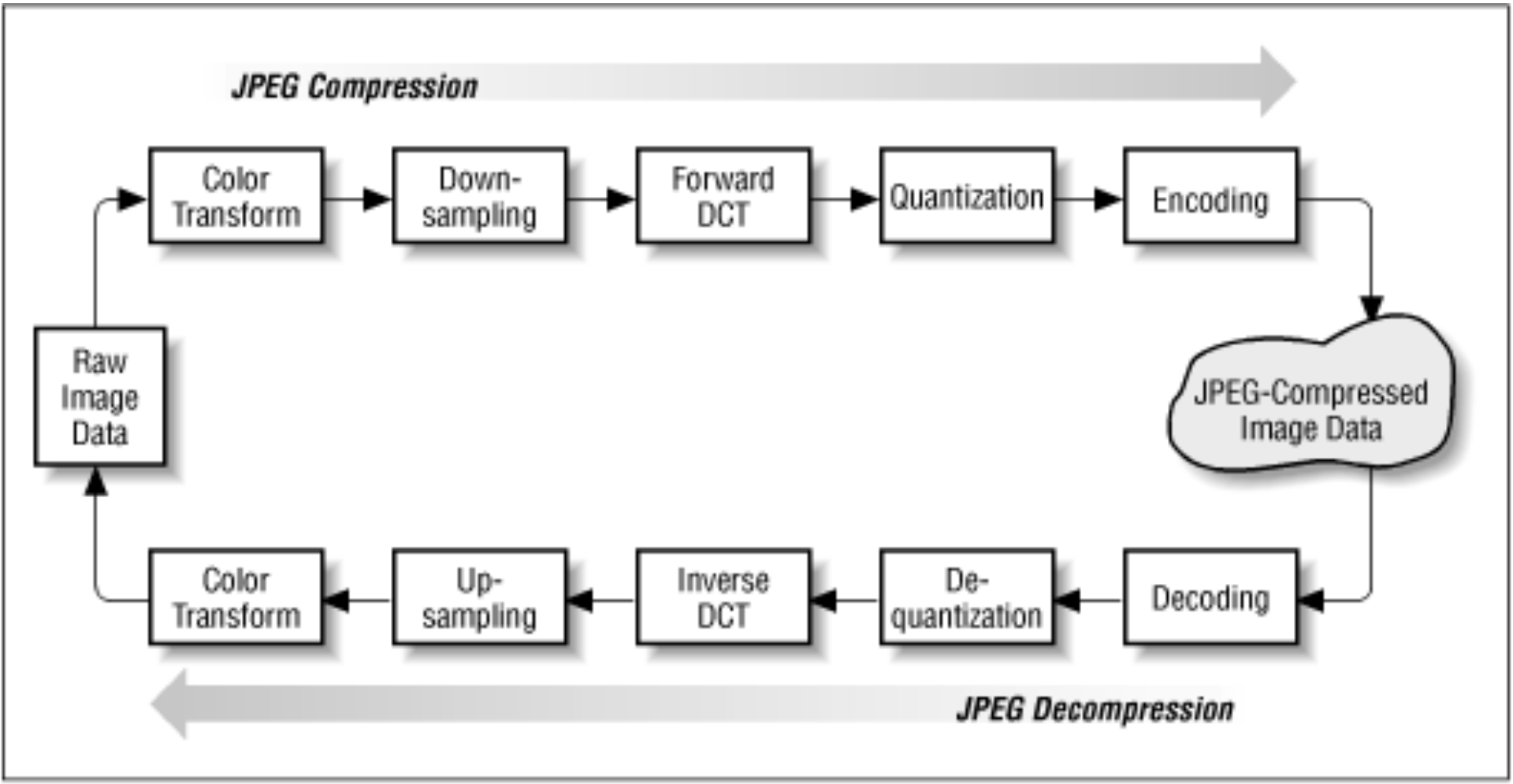
Output data:
Linear stream of bits

Compression:
Reversible suppression of the redundant information

...

Baseline sequential JPEG encoding: outline

JFIF = JPEG File Interchange Format (i.e. data stream, not file format)



Baseline sequential JPEG encoding: outline

1. Convert colors to **YCrCb** space
2. Down-sample **Cr** and **Cb** components
3. Cut image into independent **8x8** blocks
4. Apply DCT transform (result: **8x8** coefficients X_{ij} per block)
5. Quantize coefficients with a pre-defined quantization table Q_{ij}:

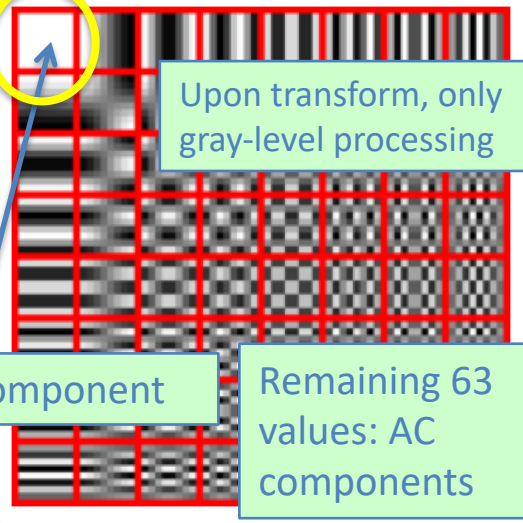
$$\begin{pmatrix} Y \\ Cr \\ Cb \end{pmatrix} = \begin{pmatrix} x & x & x \\ x & x & x \\ x & x & x \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} x \\ x \\ x \end{pmatrix}$$

- Basic tables Q_{ij} (Y, Cr) suggested by JPEG
- Table values scaled acc. to quality settings
- Tables are always saved for the decoder!

$$q_{ij} = \left\lfloor \frac{X_{ij}}{Q_{ij} \cdot f} + 0.5 \cdot \mathrm{sgn}\left(X_{ij}\right) \right\rfloor$$

6. **Encode the average value (DC component):**
   1. Find the difference **d** from the previous block's DC value
   2. Find the corresponding interval (from a standard table)
   3. Huffman-encode the interval index **i**
   4. Encode **d** with the bit depth depending on **i**
7. **Encode remaining (AC) components:**
   1. Select interval corresponding to coefficient value **c**
   2. Run-length encode leading zeros + interval index **i**:
      …, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, -7, 5, 0, … ➔ …, (7/3), …
   3. Huffman-encode the resulting (run-length/index) symbols with the standard codebook
   4. Encode **c** with bit depth depending on **i**

DCT 8x8 basis vectors:

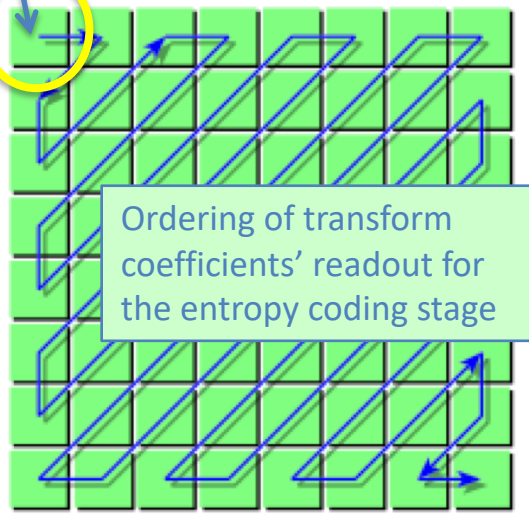


Upon transform, only gray-level processing

DC component

Remaining 63 values: AC components

Frequency-based coefficient ordering:



Ordering of transform coefficients' readout for the entropy coding stage

Baseline JPEG - examples

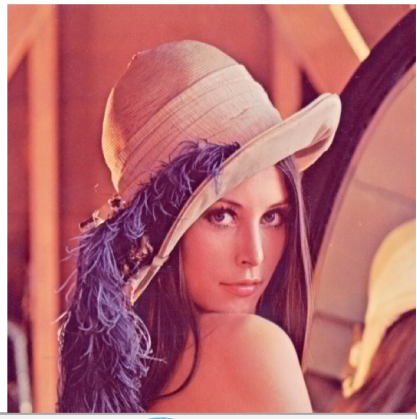
Lena.bmp: 786K

Lena.jpg: 23K

A few leading DCT coefficients as images:

Y, Cr, Cb components:

No chroma AC:



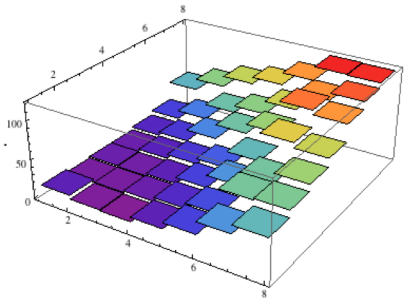
JPEG limitations and visual artifacts

- Quantize using a "quality factor" **f**:

$$q_{ij} = \left\lfloor \frac{X_{ij}}{Q_{ij} \cdot f} + 0.5 \cdot \mathrm{sgn}\left(X_{ij}\right) \right\rfloor$$

- JPEG table: human sensitivity studies

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



- Typical real-world compression before visible quality loss about **10:1**
- Works best for images with smooth variations of tone and color; large artifacts for line drawings, text, etc.
- Max image size: **65535x65535** pixels

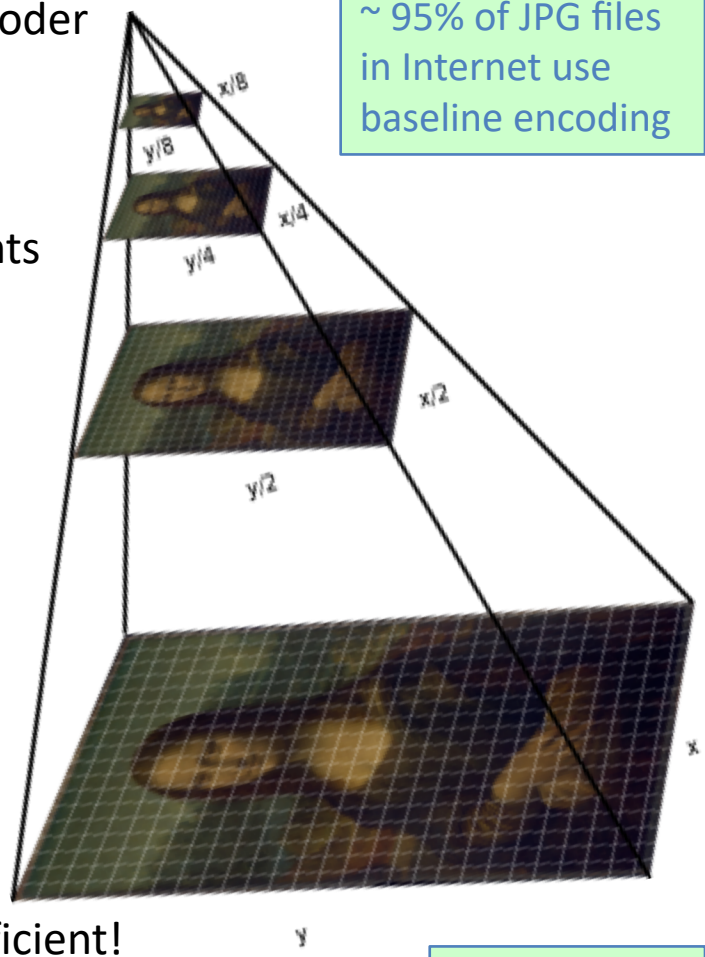
- "Blockiness" related to **8x8** block size
- Larger errors on contours / edges
- Loss of quality due to cropping, shifting, or repeated editing

JPEG – further options

- **Baseline process** must be supported by any compliant decoder

~ 95% of JPG files in Internet use baseline encoding

- **Extended DCT-based processing**
  - Up to 12-bit accuracy
  - Custom Huffman code tables for AC and DC coefficients
  - Arithmetic coding instead of Huffman coding
- **Lossless JPEG processing**
  - Predictive coding of pixel values (no DCT)
  - Use 2 to 16 bits per pixel
  - Up to 4 coding tables (context switching)
- **Hierarchical processing**
  - Encode pyramid of images with increasing resolution
  - First encode coarser versions, then finer ones
  - Up-scaled decoded image - predictor for finer image

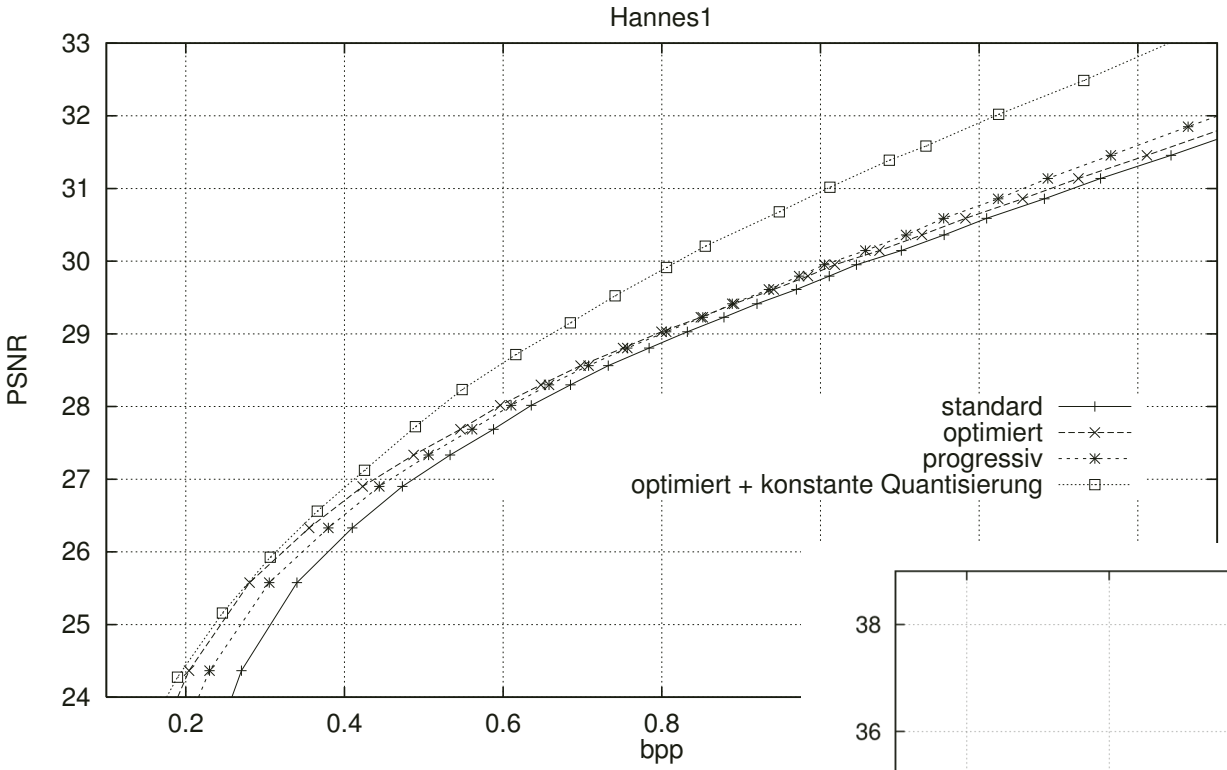


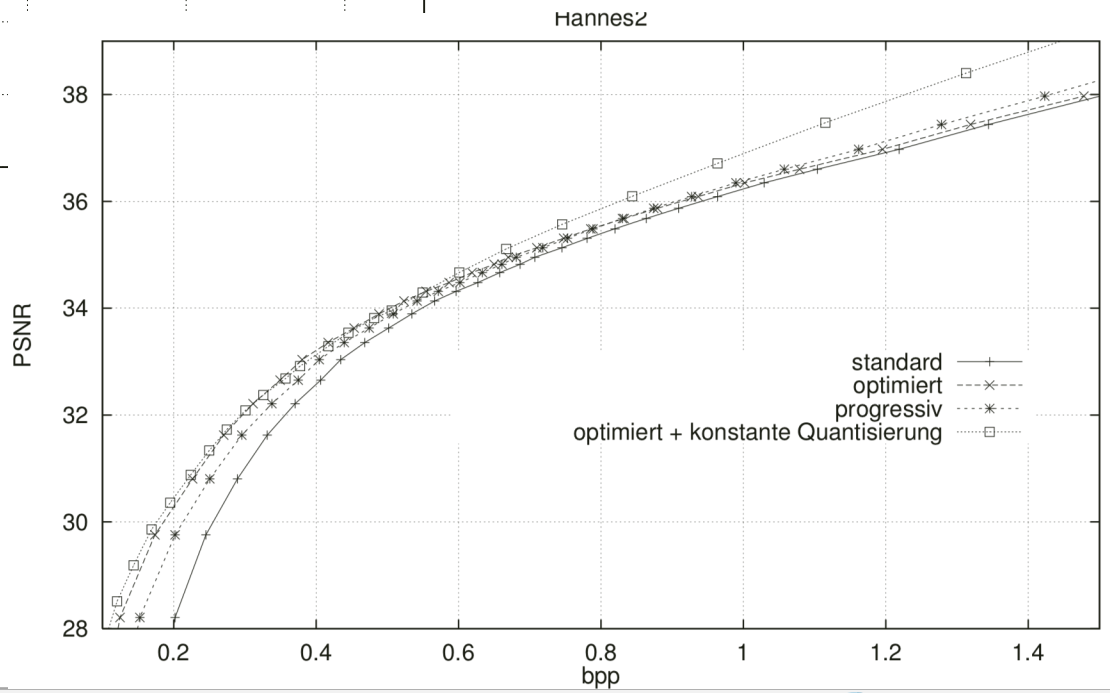
- **Sequential processing (as in baseline)**
  - Encode and send data block-wise: very simple and efficient!
- **Progressive processing**
  - While encoding, send all DC coefficients, then all S01 coefficients, etc.
  - Alternative: send first the most significant bits of all coefficients, then further bits
  - Allows gradual image refinement; may be combined with hierarchical process

AKA spectral selection

JPEG – “objective” performance benchmarks

[Strutz 2009]





Metric:
PSNR = peak signal-to-noise ratio,
bpp = bits-per-pixel, characterizes compression rate

A (much much) better modern standard: JPEG 2000

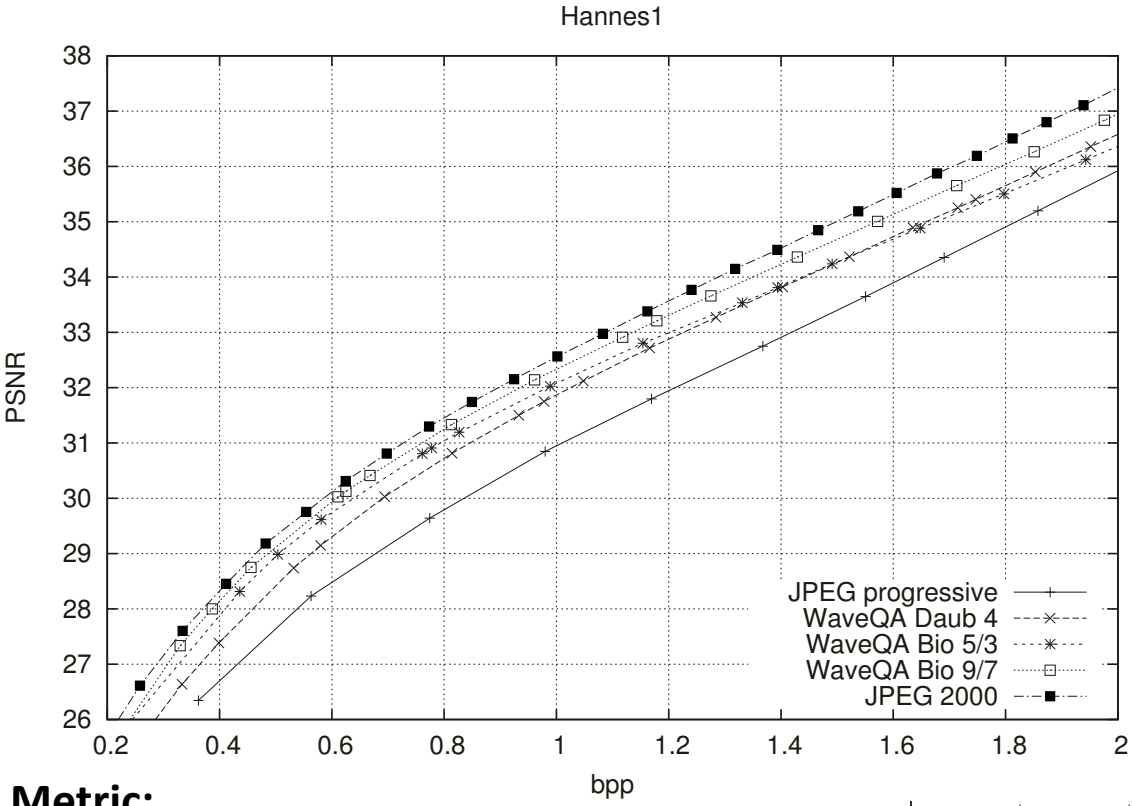
Allows up to 2^{14} image components (color layers)
Supports lossy, lossless, and nearly-lossless compression [= guaranteed max pixel value error]

- De-correlate color layers (standard JPEG functions + extensions)
- Break a large image into tiles, encoded/decoded independently
- Perform a **discrete wavelet transform (DWT):**
  - 9/7 bi-orthogonal filters for lossy compression,
  - 5/3 integer-coefficient lifting schema for lossless transform
  - Extensions of standard: may also use other wavelets
- **Quantization (lossy version):**
  - Uses a uniform quantizer with a dead zone; coefficients are re-weighted according to human sensitivity tables
  - Exact steps and dynamic range depend on original layer bit depth
- **Encoding of wavelet sub-bands:** hierarchical tree-based bit-marking algorithm, 3 steps:
  - Significance propagation (determine non-zero bits in each bit layers)
  - Magnitude refinement (update magnitudes of significant coefficients)
  - Cleanup pass (encode remaining insignificant [zero] coefficients)
- Finally, adaptive arithmetic encoder

Data granularity levels:
- Image
- Tile
- DWT sub-band
- Precincts
- Code-block
- Bit layer
- Amplitude bit

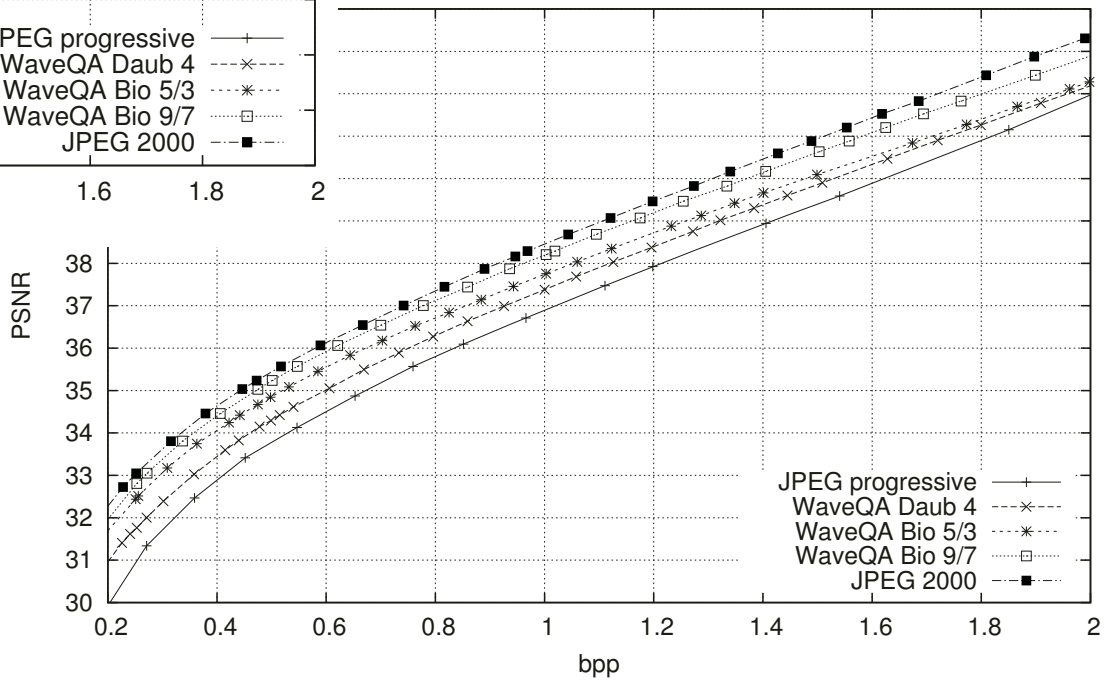
- Multiple options of progressive encoding, bit marking, contexts, etc.
- **A much** more complicated standard than JPEG!
- Extensions such as visual masking, ROI (region of interest), non-linear transforms, etc.

JPEG 200 – objective performance



[Strutz 2009]

Hannes2



Metric:

PSNR = peak signal-to-noise ratio, bpp = bits-per-pixel, characterizes compression rate

JPEG 2000 typically beats JPEG by about 20% in compression ratio at the same quality

Specialized image compression standards: JBIG-2

Application: structured documents with tables, text paragraphs, lo-res background or foreground images (primarily used in scanners and for fax transmission)

First stage: segmentation of image into text, halftone, and generic regions

- Similar to OCR, but can be faster and less accurate
- Several algorithms, mostly ad hoc (e.g. [Tompkins, Kossentini '99])

Encoding "text" image data:

- Maintain a dictionary of shapes (e.g. letters)
- Use pattern matching to find identical (or nearly identical) shapes
- Save the position on page and a dictionary index

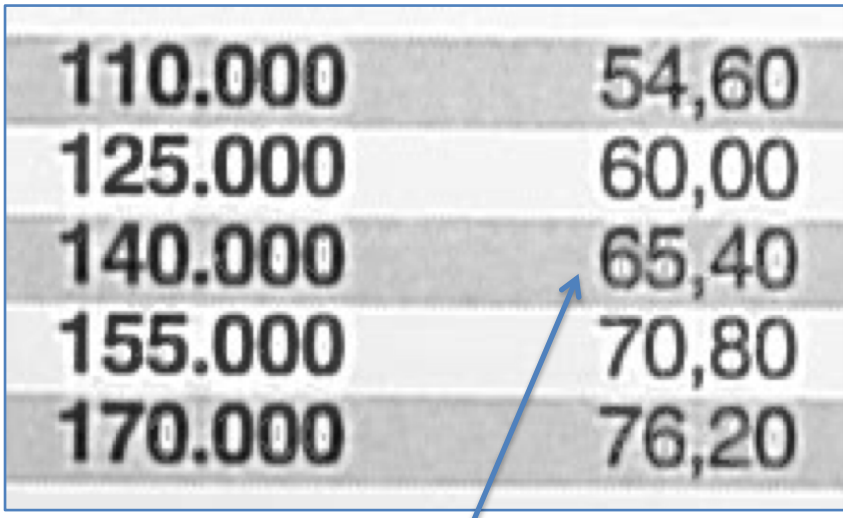
Halftone images:

- context-based prediction (based on pixel neighborhood) or dictionary of small gray-level bitmaps

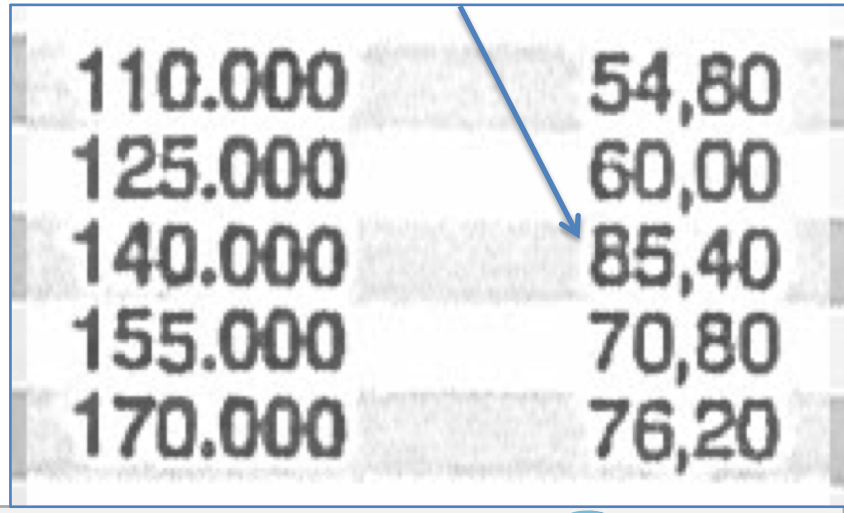
Unrecognized regions: pass values directly to coder

All data eventually encoded with arithmetic coder

A typical bi-level image:



Some Xerox scanners JBIG2 compression too aggressively [Kriesel, 2013]:



Specialized image compression standards: DJVu

Delivery format (allows various internal algorithms) [Bottou et al '98]

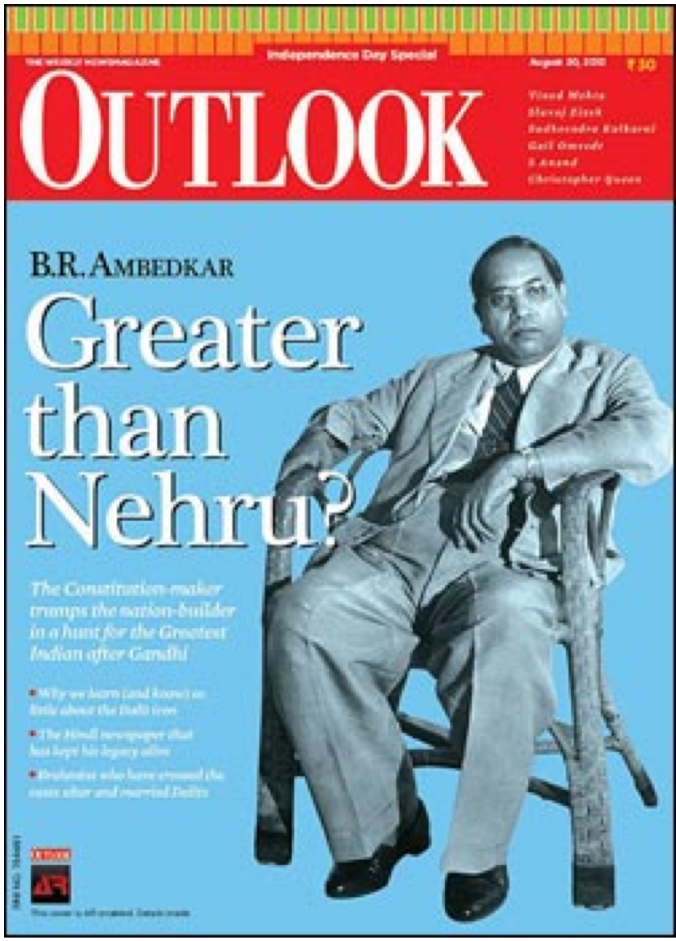
Created and fine-tuned for a specific content type!

Application:
High-quality, high-resolution images of scanned documents in color: digital books, magazine pages

Separation of layers:

- **Mask** - bi-level high-frequency data (text and drawings, typical resolution 300dpi), single bitmap compressed with JB-2 (similar to JBIG-2)
- **Background** – smooth colored images, wavelet-encoded (typical resolution 100 dpi, uses IW44 or JPEG 2000)
- **Foreground** – color of text and drawings, wavelet-encoded (typical resolution 100dpi or less)

Entropy coding: custom arithmetic coder (ZP-coder)



Text characters and line drawings need high spatial resolution, but limited color resolution

Major step: foreground / background separation!

DJVu: layer separation with K-means clustering, MRFs

Task: separate foreground from background
Simple bi-color document: two peaks in histogram

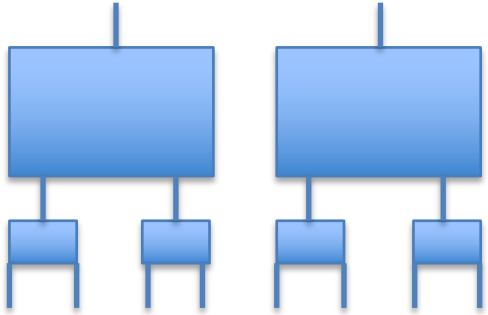
1. Initialize reference BG color to white, FG color to black
2. Classify each pixel as BG or FG based on distance to reference
3. Update BG and FG reference as average of classified pixels
4. Repeat steps 2 and 3 until convergence

Select BG vs FG: higher luminance, higher peak, topology, …

Realistic documents: multiple colors, different BG and FG in different document parts

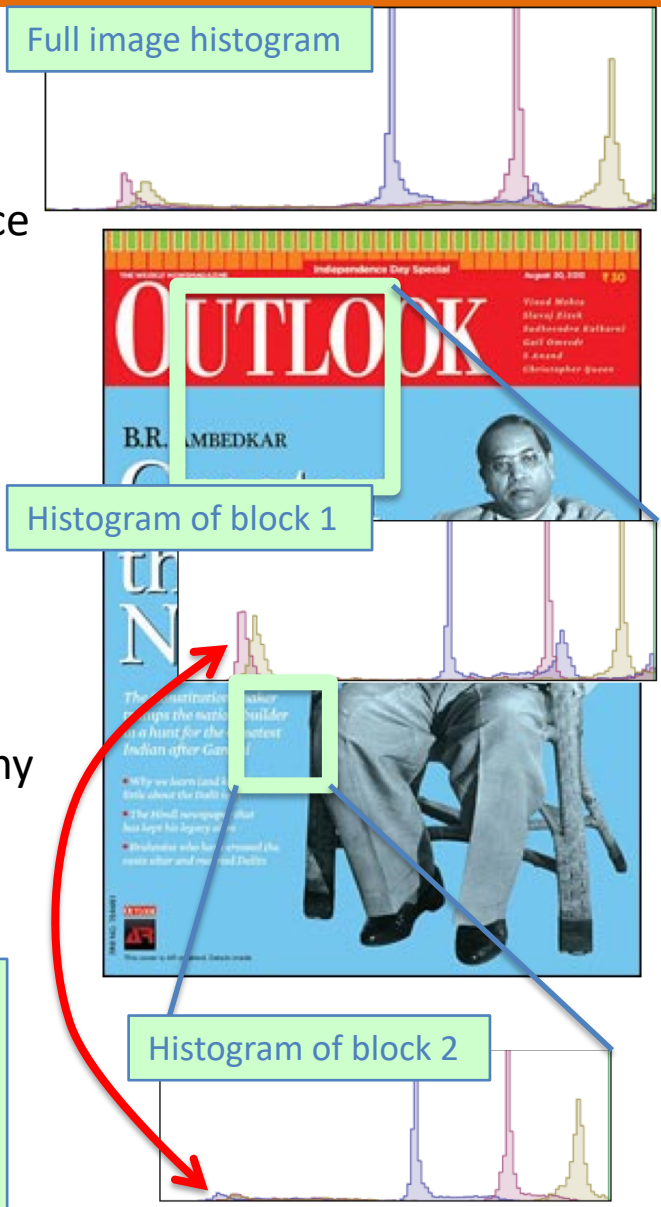
1. Prepare several grids of decreasing rectangular block size
2. Break document into blocks according to each grid
3. Run clustering for each block, initializing with coarser block
4. Update FG and BG colors in blocks up and down the hierarchy
5. Repeat 3 and 4 until convergence

Results filtered to avoid artifacts (inversions, holes, etc.)



Implementation: simple version of a Markov Random Field (MRF)-based relaxation algorithm.

Resolution-independent, simple, fast.
May implement arbitrary interactions!



DJVu vs JPEG

JPEG at 100dpi only.

[Bottou et al '98]

DjVu.

Figure 6: *Comparison of JPEG at 100dpi (left) with quality factor 30% and DjVu (right). The images are cropped from* **hobby002**. *The file sizes are 82 KBytes for JPEG and 67 KBytes for DjVu*

Image Data Compression

Basics of image sequence (video) coding

Coding video sequences vs coding of individual images

Compressing 3D data (movies) is very similar to compressing 2D images, but:

- In addition to spatial correlations, we need to account for temporal correlations
- In common sequences (at 25 fps) there is not much change from frame to frame
- This means, we have redundancy and irrelevance between subsequent frames!

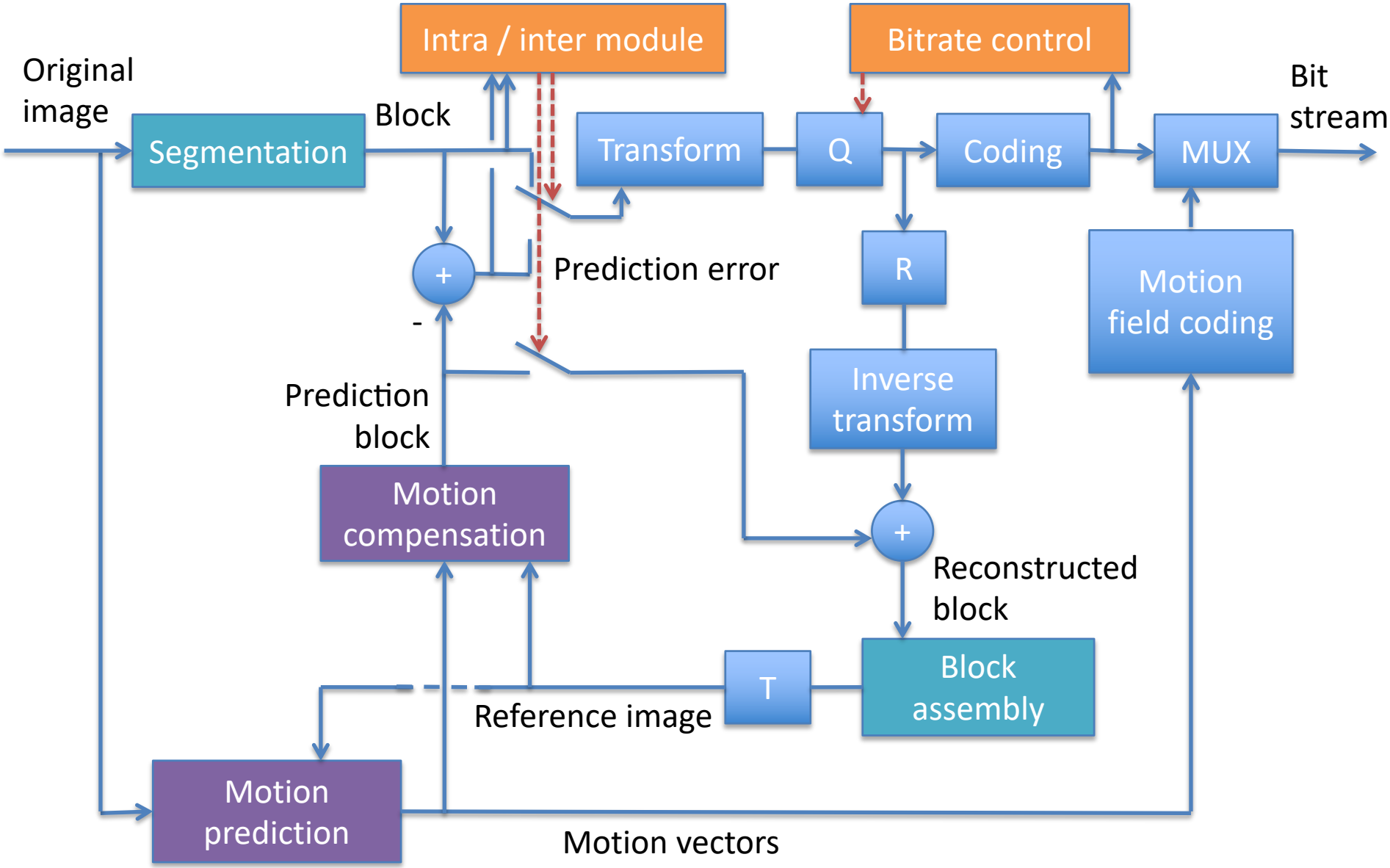
Can in principle extend common 2D techniques: transform, quantization, etc., but:

- 3D wavelets need at least 6 parameters to specify the position and frequency
- Temporal pixel value correlations vanish rapidly when objects move (a very common case!)
- Some authors claim success in combining segmentation with a transform [Schwarz '00]

Alternative [commonly accepted] way:

- Implement additional modules that model and exploit temporal correlations corresponding to the most common types of frame changes: object motion, camera "eigenmotion", static background, etc.
- Mostly prediction-based algorithms
- Even today, still need to save computational resources, accept many ad hoc solutions!

Structure of a generic video codec



Motion prediction and compensation

Typical motions related to changes in the scene / object:

- Translations in frame, rotations, deformations, illumination changes, …

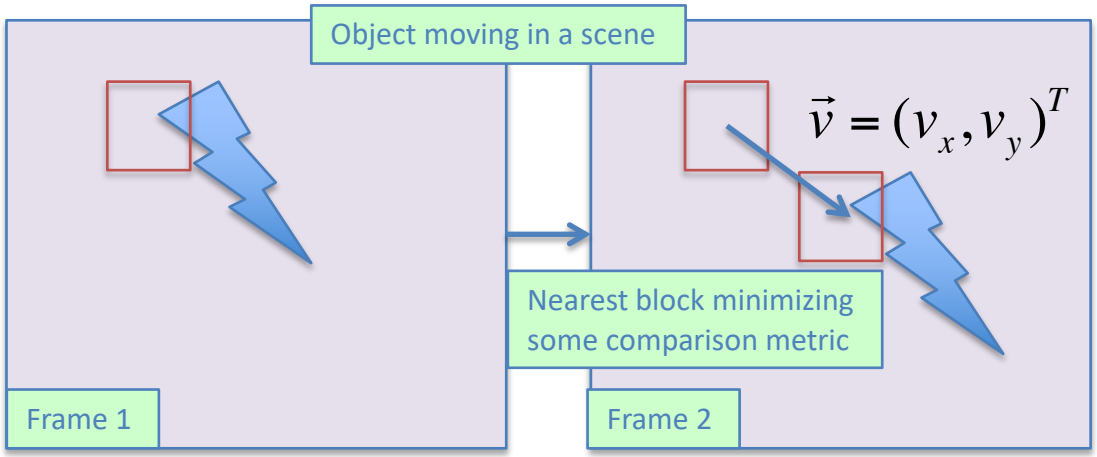
Some camera-associated motions:

- Camera shifts, rotations, re-focusing, zooming, …

In practice, algorithms mostly attempt to only compensate for translational motion of objects

- Prediction of pixel values or values of pixel groups (determined from local neighborhoods)
- Assumptions about the image contents (e.g., head-shoulder identification during video-calls)
- Object / region-based methods (based on smart segmentation of object contours).

The simplest practical method: block matching (usually performed only in the Y-channel)

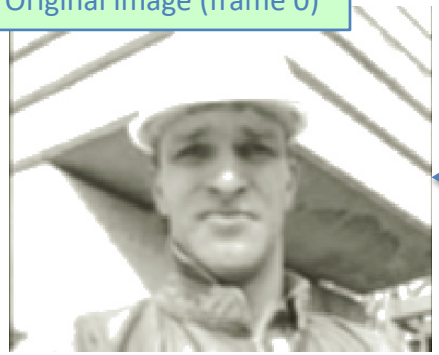


- Somewhat similar to Fractal Coding
- Balance block size vs matching accuracy
- In general, very large search space!
- N/A for new appearing content
- May reach sub-pixel accuracy
- Forward / backward estimation possible
- Hierarchical motion estimation
- Overlapping blocks, long-time storage

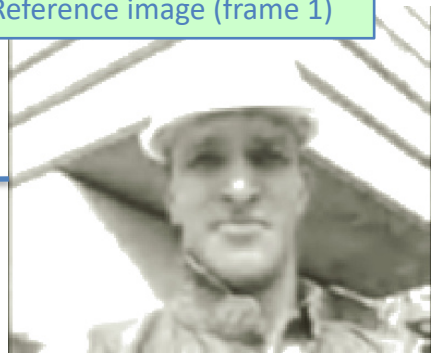
Closely related problem from Computer Vision: estimation of Optical Flow field

Block matching - examples

Original image (frame 0)



Reference image (frame 1)

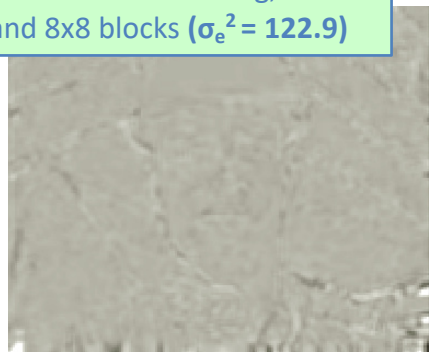


Pixel difference w/o motion compensation ($\sigma_e^2 = 1394.3$)

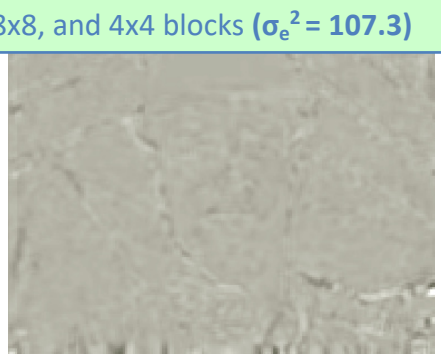
Searching in the neighborhood, integer vectors

Simple block matching, 16x16 blocks ($\sigma_e^2 = 153.1$)

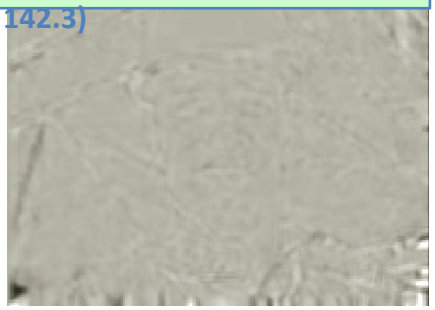
Hierarchical matching, 16x16 and 8x8 blocks ($\sigma_e^2 = 122.9$)



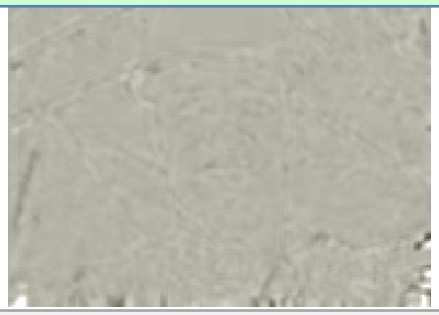
Hierarchical matching with 16x16, 8x8, and 4x4 blocks ($\sigma_e^2 = 107.3$)



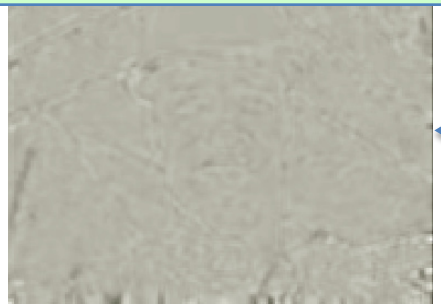
Simple matching, 16x16 blocks, 0.5-pixel acc ($\sigma_e^2 = 142.3$)



Hierarchical matching, 16x16 and 8x8 block, 0.5-pix acc ($\sigma_e^2 = 108.1$)



Hierarchical matching with 16x16, 8x8, and 4x4 blocks, 0.5-pix acc ($\sigma_e^2 = 87.6$)



Allowing half-pixel accuracy of motion vector components