Digital Video

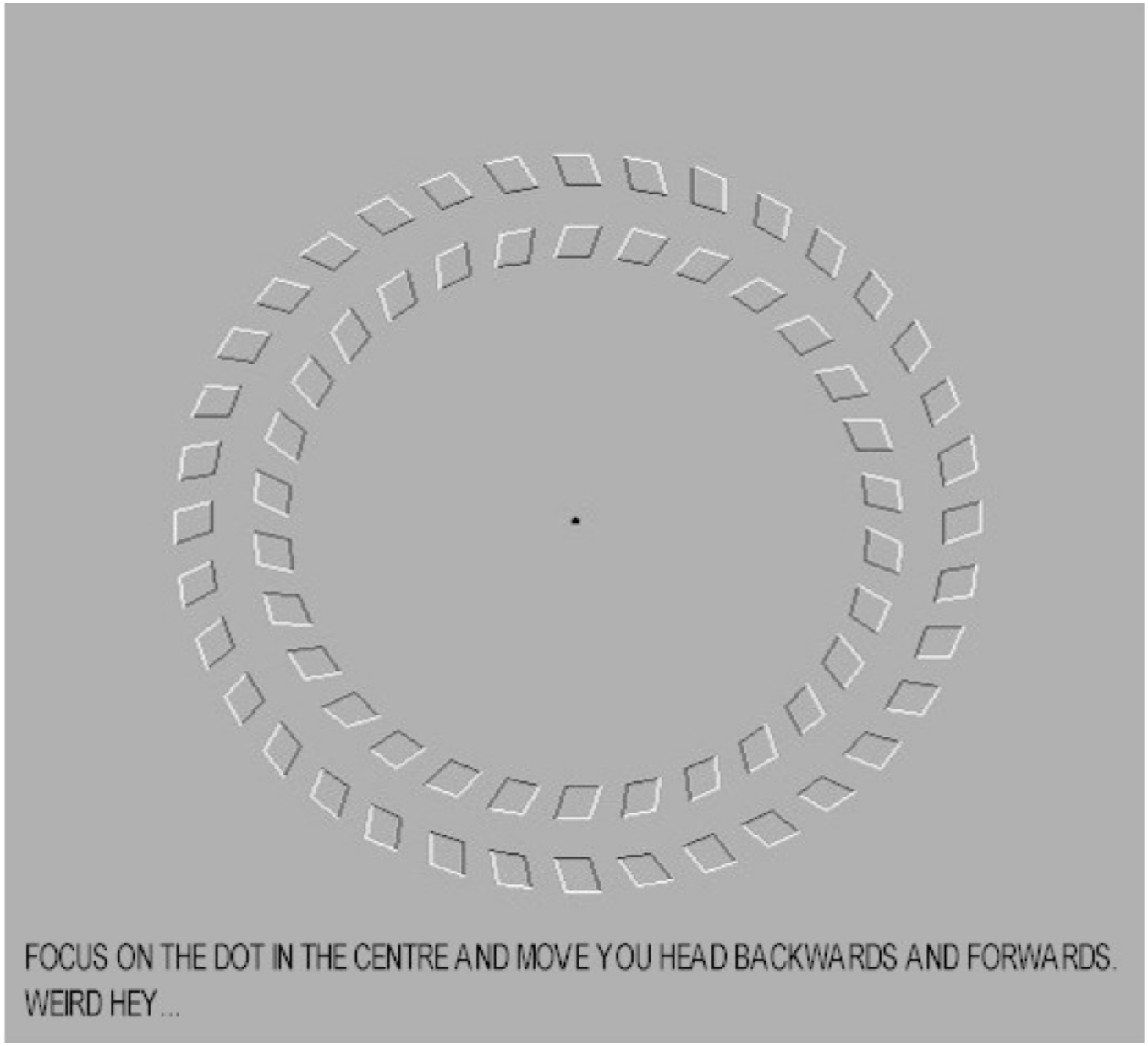
EE6310: Image and Video Processing, Spring 2023

March 30, 2023

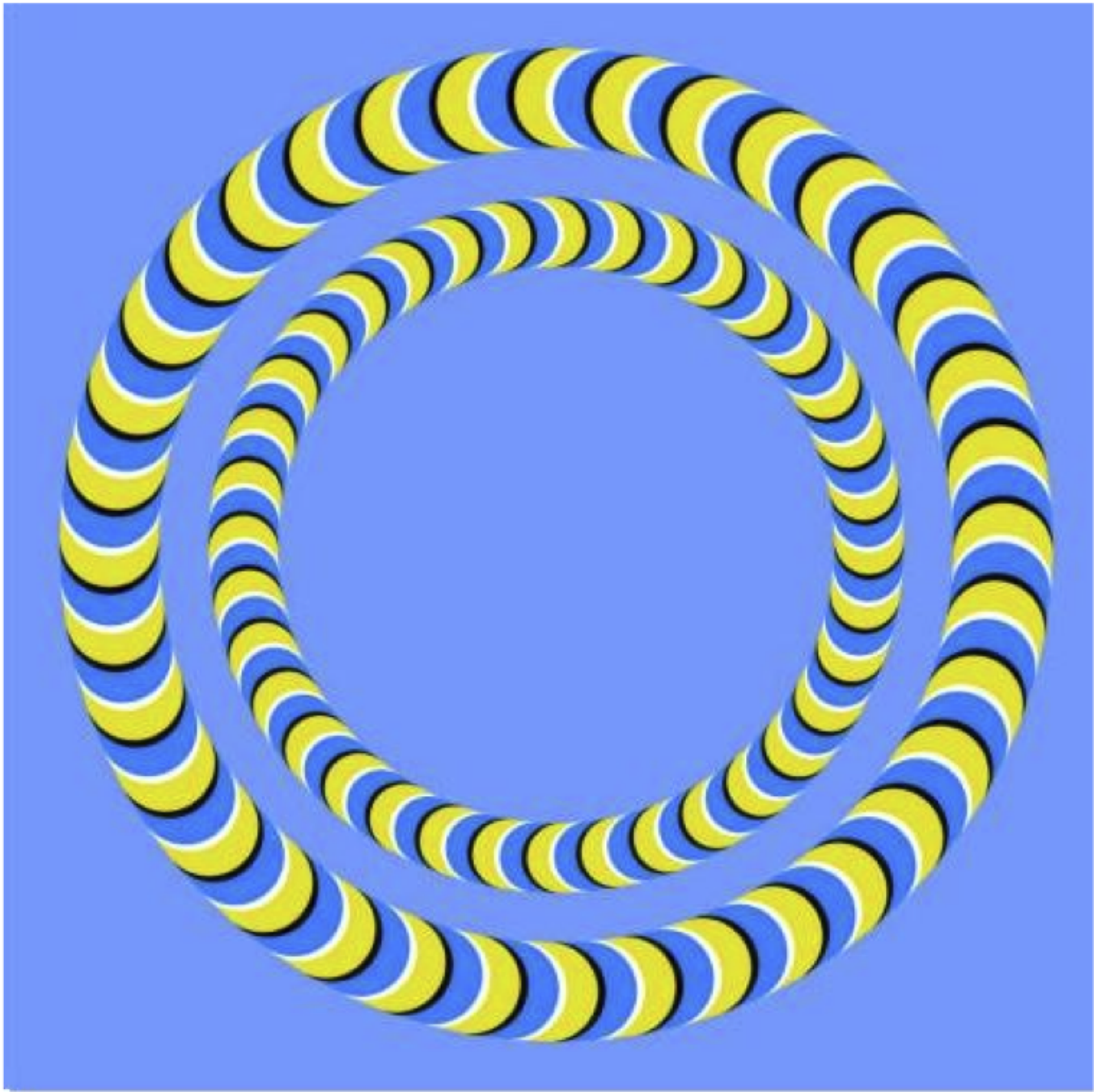
Digital Video

- Generic Video Codec
- Block Motion Estimation
- Interframe Coding and Motion Estimation
- Video Compression Standards: MPEG-x/H.264
- Optical Flow

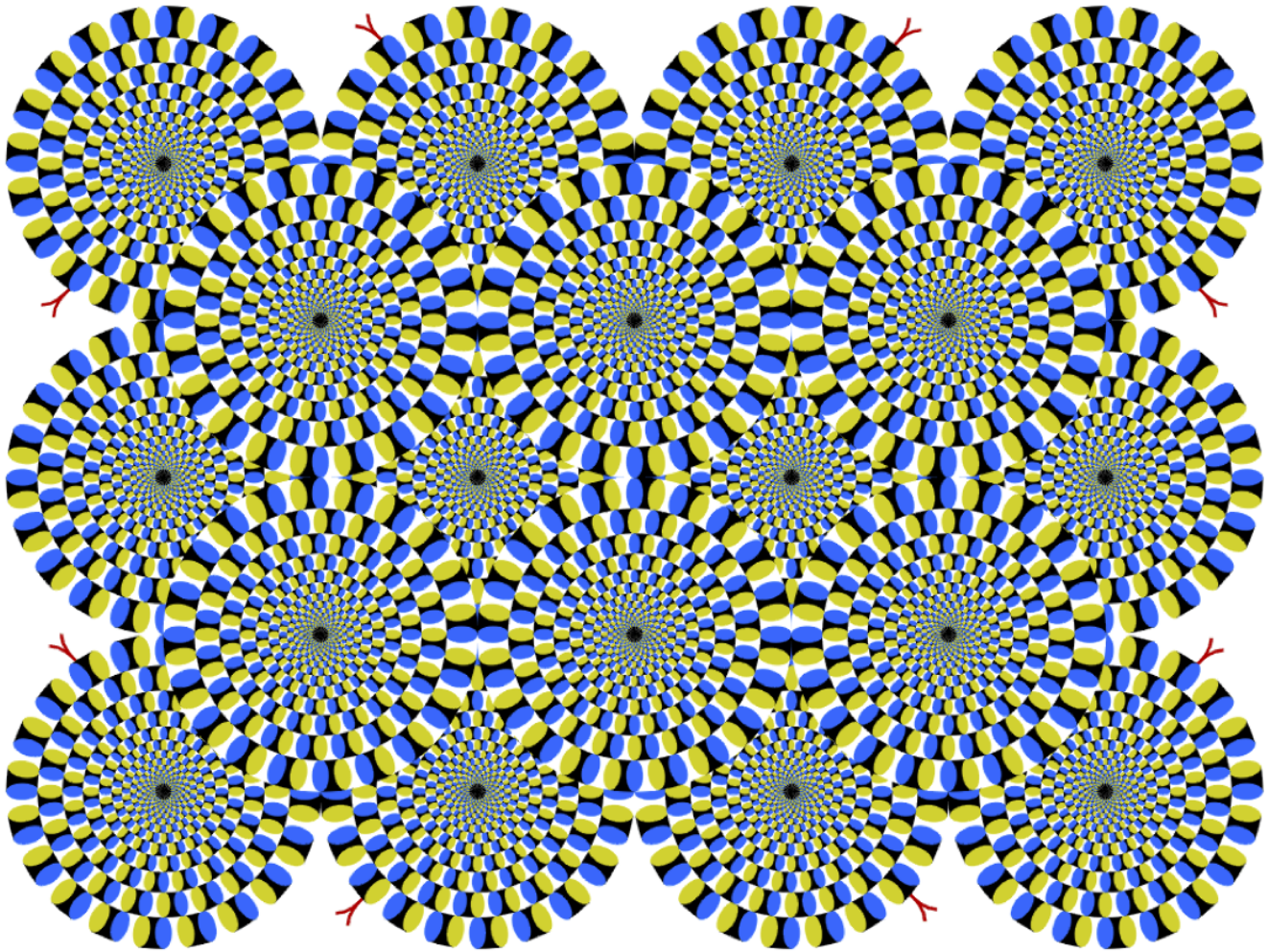
Induced Motion Effects



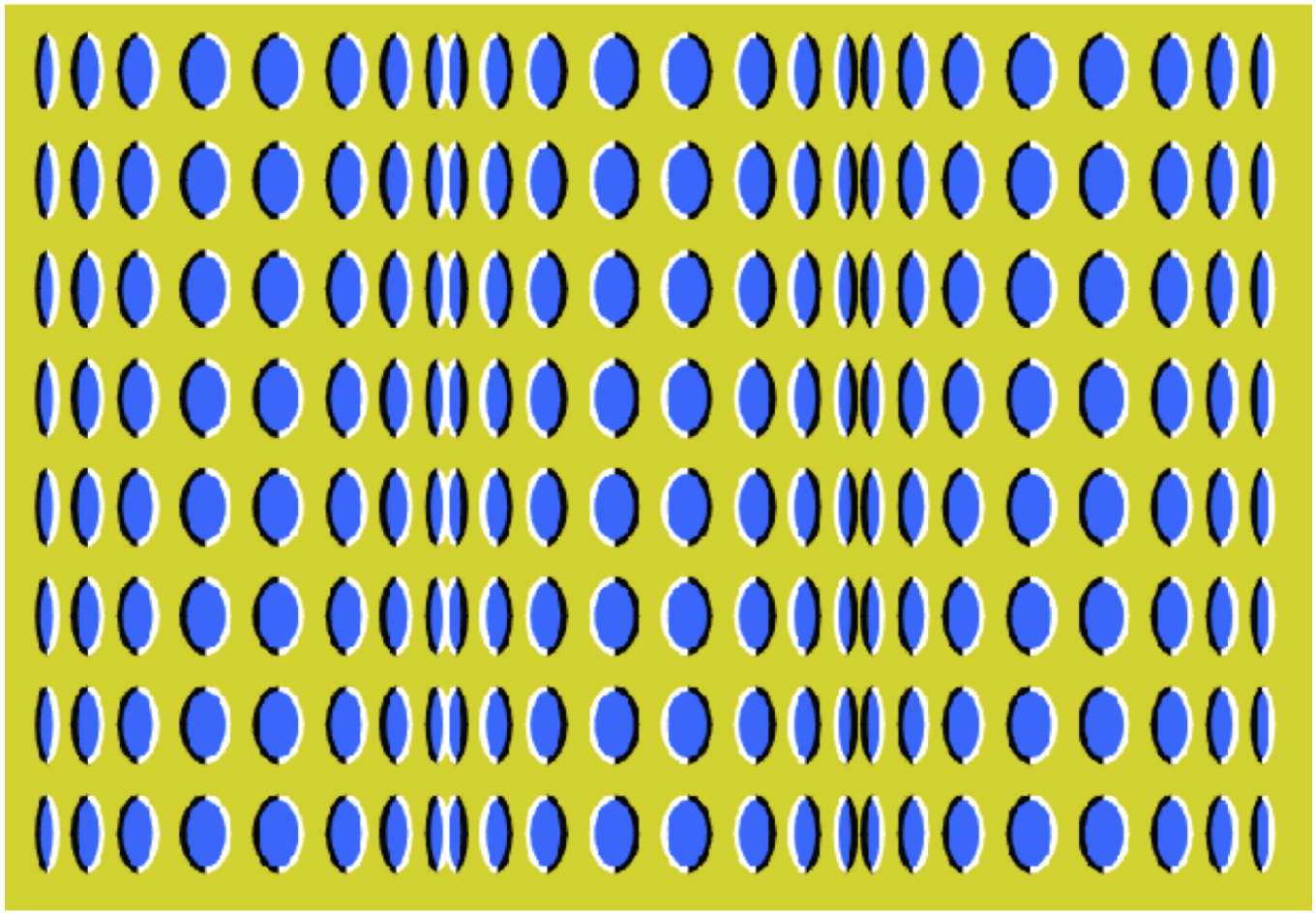
Induced Motion Effects



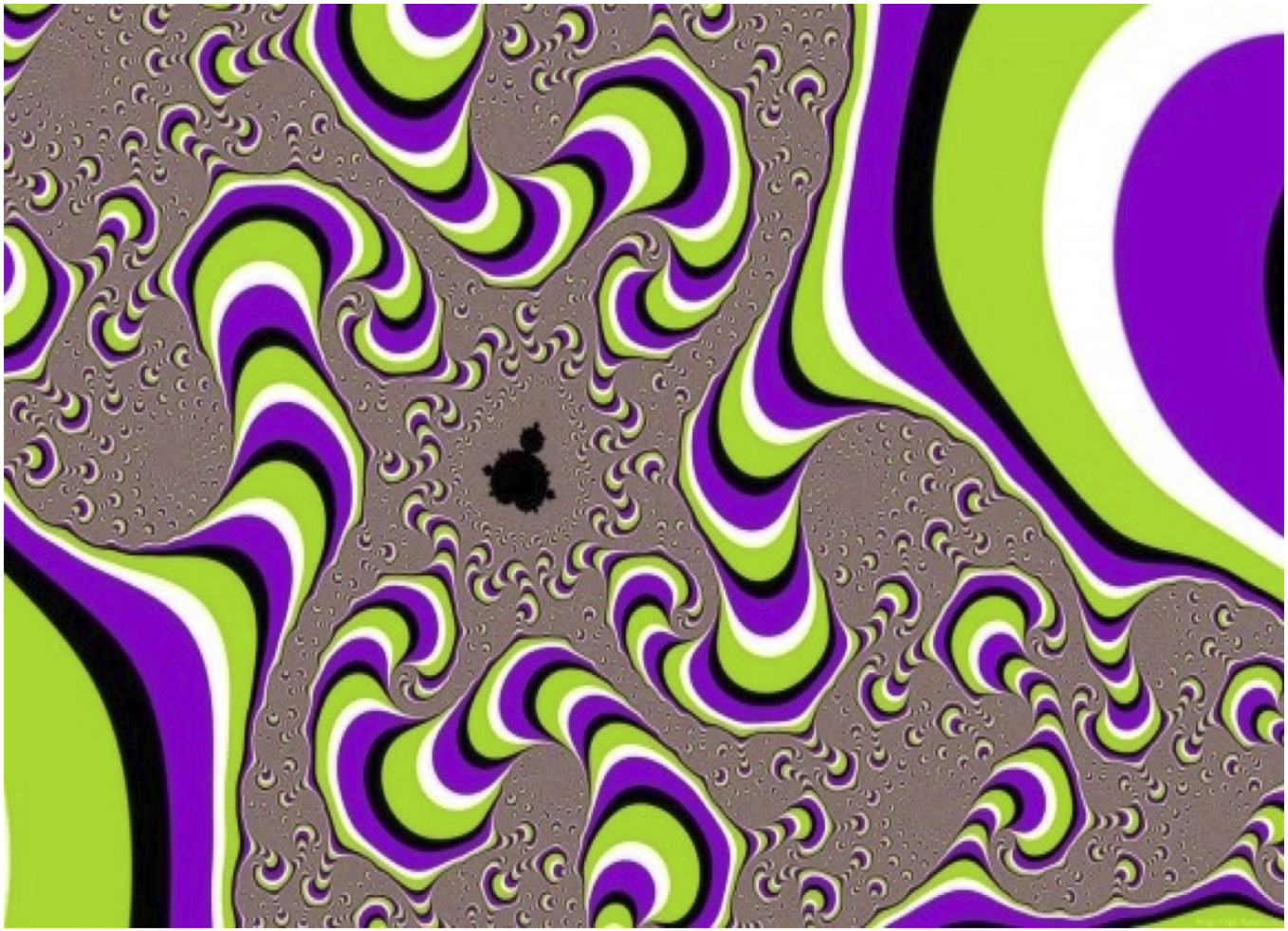
Induced Motion Effects



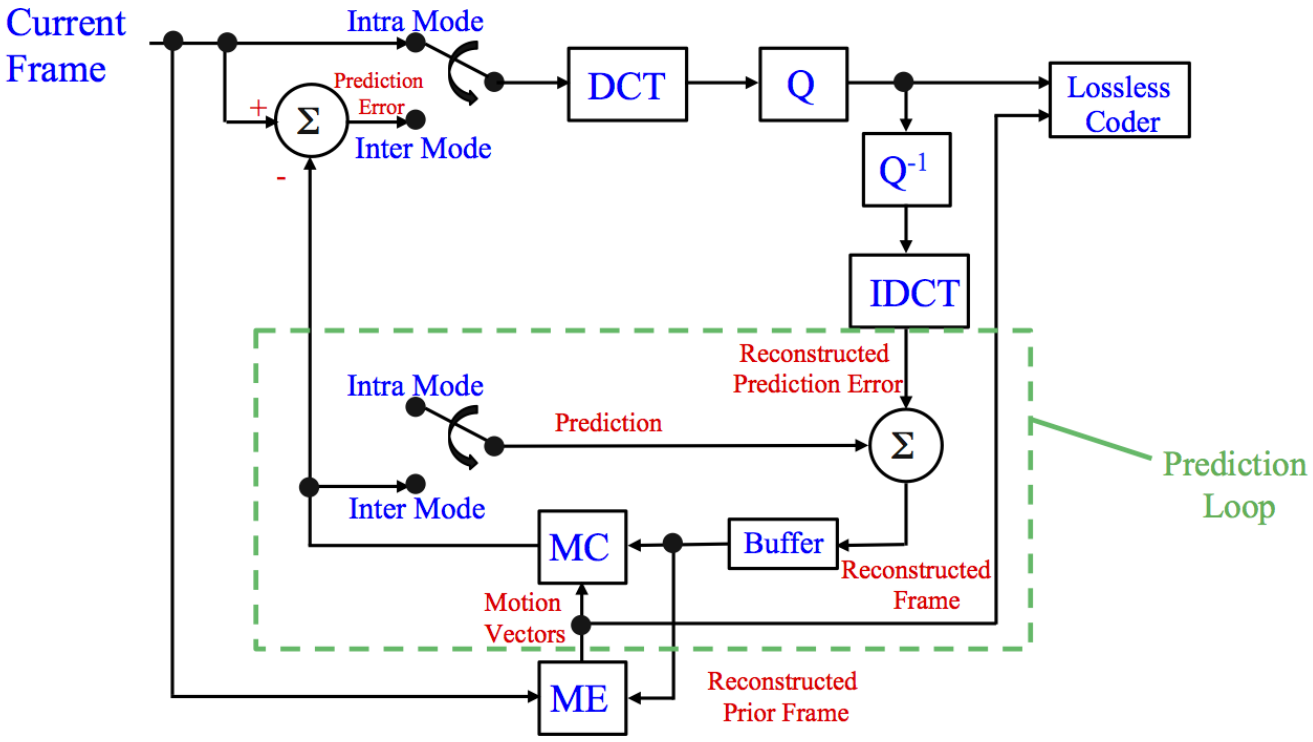
Induced Motion Effects



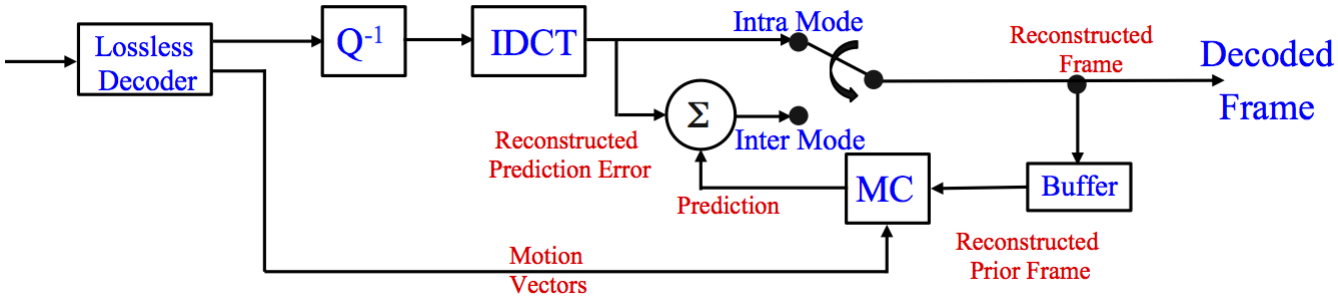
Induced Motion Effects



Generic Video Encoder Diagram



Generic Video Decoder Diagram



Lossless Coding

- **Lossless** techniques achieve compression with **no loss of information**
- The true image can be reconstructed **exactly** from the coded image
- Lossless coding doesn't usually achieve **high compression** but has applications such as:
  - In **combination** with lossy compression, multiply gains
  - In applications where information loss is **unacceptable**
- Lossless compression ratios usually in the range:
  $2:1 \leq \text{CR} \leq 3:1$ but can vary from image to image

Methods for Lossless Coding

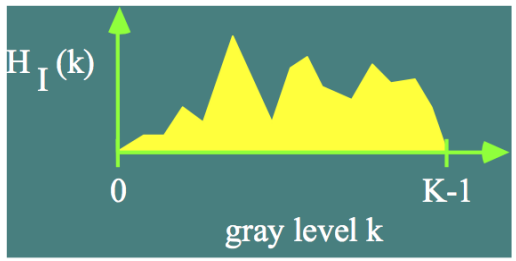
- Basically amounts to clever rearrangement of data
- This can be done in many ways and many domains (DFT, DCT, wavelet etc)
- The most popular methods use **variable length coding (VLC)**

Variable Length Coding (VLC)

- Idea: use **variable length codewords** to encode gray levels
- Assign **short lengths** to gray levels that occur **frequently**
- Assign **long lengths** to gray levels that occur **infrequently**
- On average, the **bits per pixel (BPP) will be reduced**

Image Histogram and BPP

- Recall the image histogram $\mathbf{H_I}$:



- If $B(k)$ is the number of bits used to code gray-level k, then $\text{BPP}(\mathbf{I}) = \frac{1}{NM} \sum_{k=0}^{K-1} B(k) H_I(k)$
- **BPP is the common measure** for VLC

Image Entropy

- Recall the **normalized histogram** values
  $p_I(k) = \frac{1}{NM} H_I(k); k = 0, \ldots, K-1$
- $p_I(k)$ is the probability of gray level k
- The **entropy** of image **I** is then:
  $$\mathrm{E}[\mathbf{I}] = -\sum_{k=0}^{K-1} p_I(k) \log_2(p_I(k))$$
- **Entropy** is a **measure of information**
- Provides a lower bound on VLCs:
  $\mathrm{BPP}(\mathbf{I}) \geq \mathrm{E}(\mathbf{I})$

Optimal VLC

- Recall

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

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

- Comparing equations, if $B(k) = -\log_2(p_I(k))$, optimum code found - lower bound attained!

The Huffman Code

- The **Huffman algorithm** yields an **optimum code**
- For a set of gray levels $\{0, \ldots, K-1\}$ it gives a set of **code words** $c(k)$ such that
  $\mathrm{BPP}(\mathbf{I}) = \sum_{k=0}^{K-1} p_I(k) L(c(k))$ is the **smallest possible**

The Huffman Algorithm

- Form a binary tree with branches labeled by the gray-levels k_m and their probabilities $p_I(k_m)$:
  1. Eliminate any k_m where $p_I(k_m) = 0$
  2. Find 2 **smallest probabilities** $p_m = p_I(k_m), p_n = pI(kn)$
  3. Replace by $p_{mn} = p_m + p_n$ to form a node; reduce list by 1
  4. Label the branch for k_m with (e.g.) '1' and for k_n with '0'
  5. Until list has only 1 element (root reached), return to (2)
- In step (4), values '1' and '0' are assigned to element pairs (k_m, k_n), element triples, etc. as the process progresses

Huffman Example

- There are $K = 8$ values $0, \ldots, 7$ to be assigned codewords: $p_I(0) = 0.5, p_I(1) = p_I(2) = p_I(3) = 0.125, p_I(4) = 0.0625, p_I(5) = p_I(6) = 0.03125, p_I(7) = 0$
- The process creates a **binary tree** with values '1' and '0' placed on the top and bottom branches at each stage
- Solution on board - make note

Huffman Decoding

- The Huffman code is a **uniquely decodable code**. There is **only one interpretation** for a series of codewords (bits)
- Decoding progresses as follows:
  - Starting at tree root, traverse tree using coded bits until a leaf is found. The symbol at the leaf is output
  - Return to tree root and repeat above step until all bits are exhausted

Arithmetic Coding

- Assigns a **single arithmetic code word** to an entire sequence of source symbols – creates a **mapping** between source symbol sequence and real numbers in the interval $[0, 1)$
- Achieves **higher compression efficiency** than Huffman codes - no need to map source symbols to integral number of code symbols
- Achieves Shannon's noiseless source coding bound

Arithmetic Coding Algorithm

- Divide the interval $[0, 1)$ according to PMF
- For e.g., if: $p(a) = p(e) = 0.25, p(i) = p(o) = 0.2, p(u) = 0.1$

Symbol	a	e	i	o	u
Range	$[0, 0.25)$	$[0.25, 0.5)$	$[0.5, 0.7)$	$[0.7, 0.9)$	$[0.9, 1)$

- Initialize $h = 1, l = 0$
- Loop over all source symbols
  - $r = h - l$
  - $h = l + r * h_s$
  - $l = l + r * l_s$
- Final output is any real number in the interval $[h, l)$

Arithmetic Decoding Algorithm

- Initial r to encoded number
- Loop over sequence length or until EOF symbol
  - Find interval where r lands - output corresponding source symbol s
  - $r = \frac{r-l_s}{h_s-l_s} = \frac{r-l_s}{p(s)}$
- Example on board ...

Lossy Coding – Goals

To optimize and balance the following:

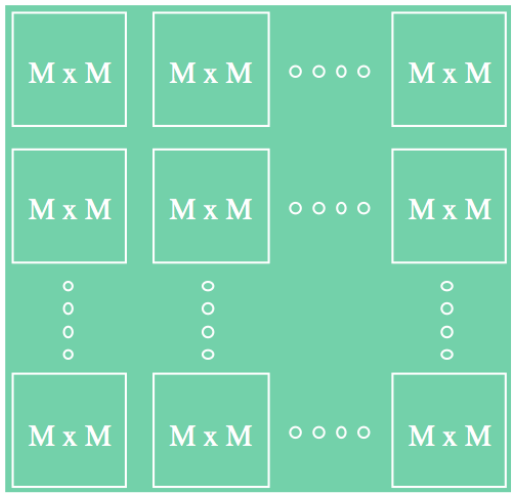
- **Compression** achieved by coding
- **Computation** required to code and decode
- **Quality** of the decompressed data

Lossy Coding – Broad Methodology

- **Many** methods proposed
- The successful methods broadly and loosely follow:
  - Transform the image to another domain and/or extract features
  - Quantize in this image or those features
  - Efficiently organize and/or entropy code the quantized data

Lossy Coding – Block Coding

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



Lossy Coding – Why Block Coding?

- **Reason:** Images are **highly non-stationary: different areas** of an image may have **different properties**
- E.g., more high or frequencies, more or less detail etc.
- **Local coding** is thus **more efficient**
- Wavelet methods provide localization without blocks
- Typical block sizes: $\mathbf{4 \times 4}, \mathbf{8 \times 8}, \mathbf{16 \times 16}$

Lossy Coding – Karhunen-Loeve Expansion

- Thm 8.5-1 in Stark and Woods, stated here for completeness
- **Optimal** decorrelating transform in a **probabilistic** framework
- Theorem: Let $X(t)$ be a zero-mean, second-order random process defined over $[-T/2, T/2]$ with continuous covariance function $K_{XX}(t_1, t_2) = R_{XX}(t_1, t_2)$, then
  $$X(t) = \sum_{n=1}^{\infty} X_n \phi_n(t) \text{ for } |t| \leq T/2$$
- $$X_n \equiv \int_{-T/2}^{+T/2} X(t)\phi_n^*(t)dt$$
- The set of functions $\{\phi_n(t)\}$ is a complete orthonormal set of solutions to the integral equation
  $$\int_{-T/2}^{+T/2} K_{XX}(t_1, t_2)\phi_n(t_2)dt_2 = \lambda_n \phi_n(t_1) \text{ for } |t_1| \leq T/2$$
- The coefficients X_n are statistically orthogonal
  $$E[X_n X_m^*] = \lambda_n \delta_{mn}$$

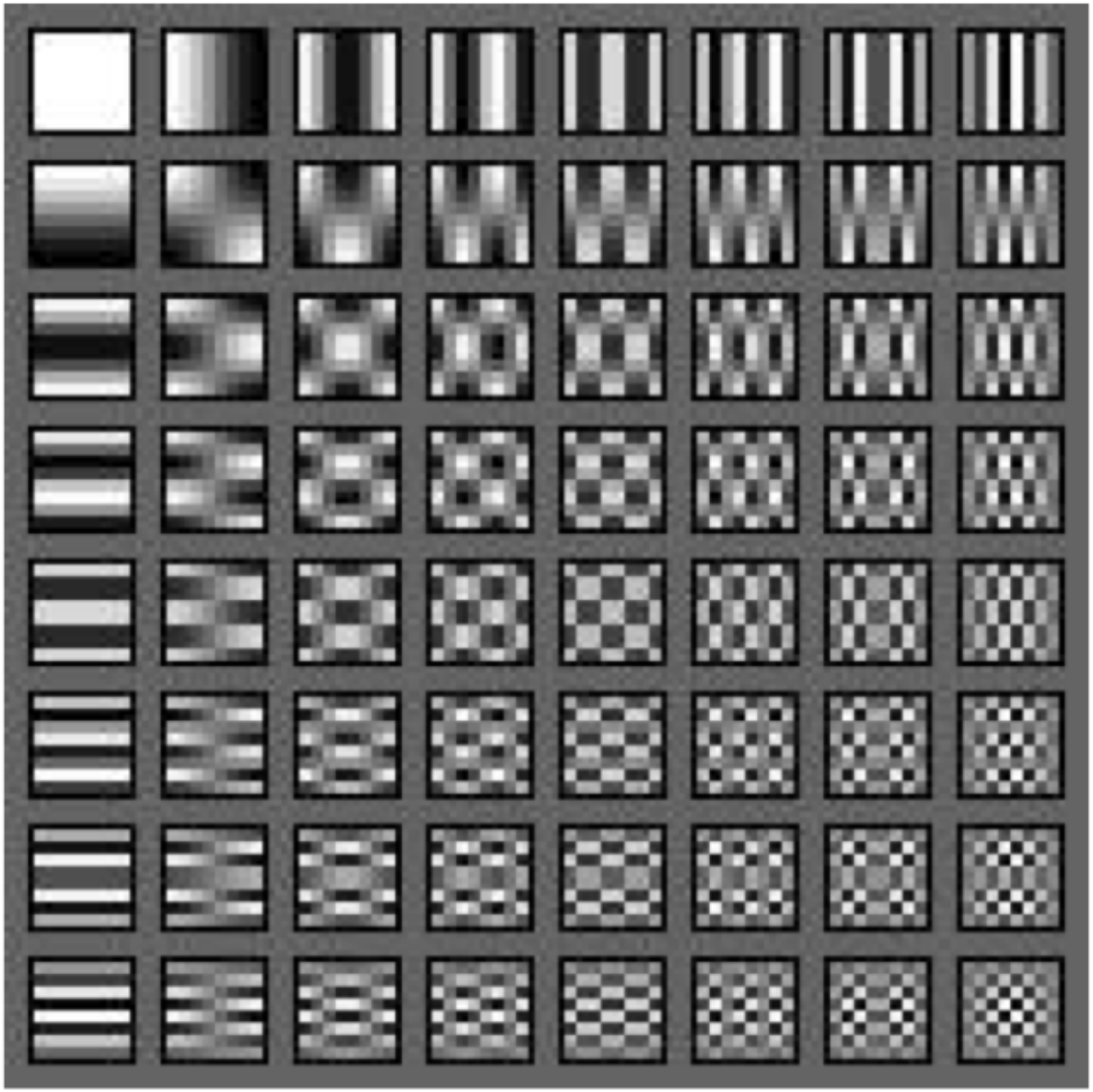
Lossy Coding – Principal Components Analysis (PCA)

- **Optimal** decorrelating transform in a **deterministic** framework
- Given an $m \times n$ matrix $\mathbf{X}$ of observations of an unknown system or a physical process
- Find a **linear** transform matrix $\mathbf{P}$ of size $m \times m$ such that $\mathbf{Y} = \mathbf{PX}$
- The transform should be such that:
  - The covariance matrix $C_{\mathbf{Y}}$'s off-diagonal elements must be 0
  - Successive dimensions in $\mathbf{Y}$ must be rank-ordered according to variance
- Note: $C_{\mathbf{X}} \equiv \frac{1}{n}\mathbf{X}\mathbf{X}^{\mathbf{T}}$
- Proof outline on board ...

Lossy Coding – Discrete Cosine Transform

- The DCT of an $N \times M$ image or sub-image is defined as:
$$\tilde{I}(u,v) = \frac{4C_N(u)C_M(v)}{NM} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} I(i,j)\cos[\frac{(2i+1)u\pi}{2N}]\cos[\frac{(2j+1)v\pi}{2M}]$$
- The inverse DCT of is defined as:
$$I(i,j) = \sum_{u=0}^{N-1} \sum_{v=0}^{M-1} C_N(u)C_M(v)\tilde{I}(u,v)\cos[\frac{(2i+1)u\pi}{2N}]\cos[\frac{(2j+1)v\pi}{2M}]$$
- $C_N(u) = \begin{cases} \frac{1}{\sqrt{2}}, & \text{if } u = 0 \\ 1, & u = 1, \ldots, N-1 \end{cases}$

Lossy Coding – Discrete Cosine Transform



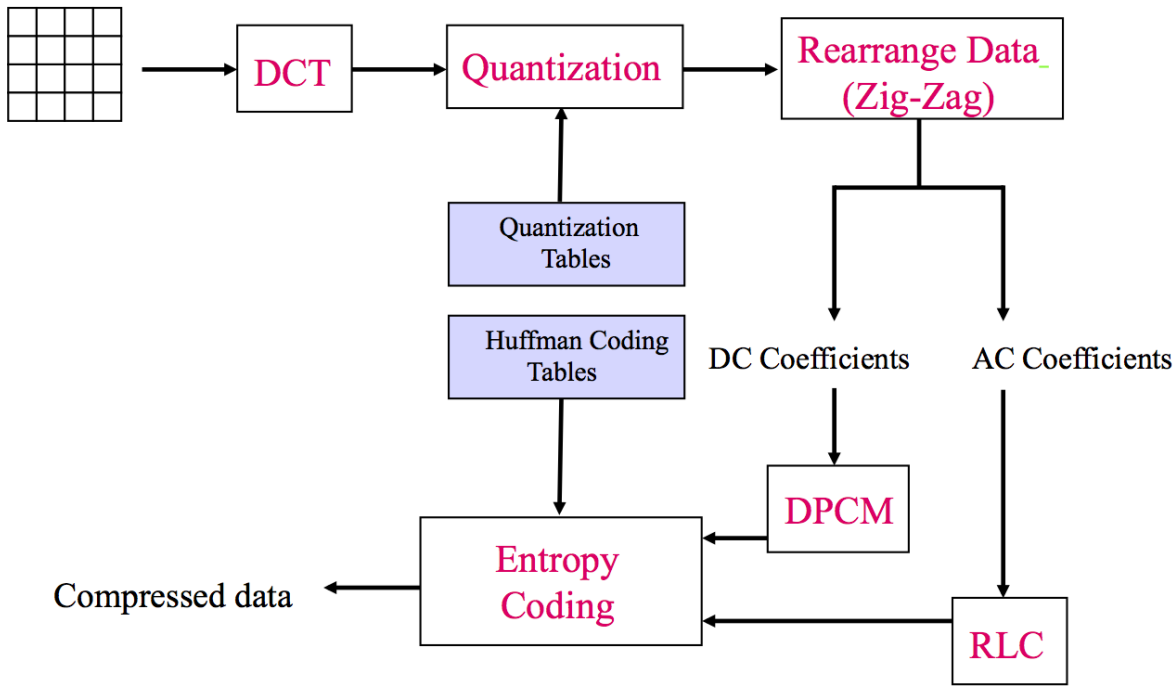
Lossy Coding – Discrete Cosine Transform

- Good decorrelating properties
- **Non-adaptive** orthonormal basis
- Seperable, fast implementation
- Adopted by JPEG and MPEG standards

Lossy Coding – Overview of JPEG

- The commercial industry standard - formulated by the CCIT **Joint Photographic Experts Group (JPEG)**
- Uses **DCT** as the central transform
- Standard is **quite complex** - will only discuss outline here

Lossy Coding – JPEG Block Diagram



Lossy Coding – JPEG Baseline Algorithm

- Partition image into 8×8 blocks and apply DCT to each block to get $\tilde{I}_k(u, v)$
- Pointwise divide each block by an 8×8 **user-defined normalization array** $Q(u, v)$
- $Q(u, v)$ designed using sensitivity properties of **human vision**
- **Uniformly quantize** the result
  $\tilde{I}_k(u, v) = \text{INT}[\frac{\tilde{I}_k(u,v)}{Q(u,v)} + 0.5]$

Lossy Coding – JPEG Baseline Example

- A block DCT (integer only algorithm) $\tilde{I}_k =$

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

- JPEG Normalization Array $Q =$

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

Lossy Coding – JPEG Baseline Example

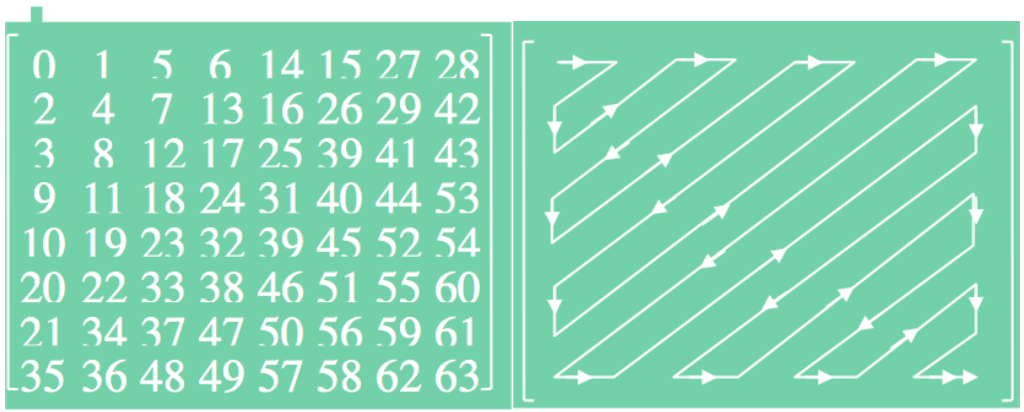
- A block DCT (integer only algorithm) $\tilde{I}_k =$

$$\begin{bmatrix} 79 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

- Notice all the **zeros** - due to DCT's **good energy compaction**

Lossy Coding – JPEG Data Rearrangment

- Rearrange quantized AC coefficients
- This array contains **mostly zeros**, especially at high frequencies
- So, rearrange into a 1-D array using **zig-zag ordering**



Reordered quantized block is:
$[79, 0, -2, -1, -1, -1, 0, 0, -1, (55\ 0's)]$

Lossy Coding – JPEG DC Coefficient Handling

- **Simple DPCM** applied to **DC values** $\tilde{I}_k(0,0)$ between adjacent blocks to reduce entropy
- **Difference** between current block and left-adjacent block is found
  $e(k) = \tilde{I}_k(0,0) - \tilde{I}_{k-1}(0,0)$
- $e(k)$ losslessly encoded with Huffman coder
- First column of DC values retained to allow reconstruction

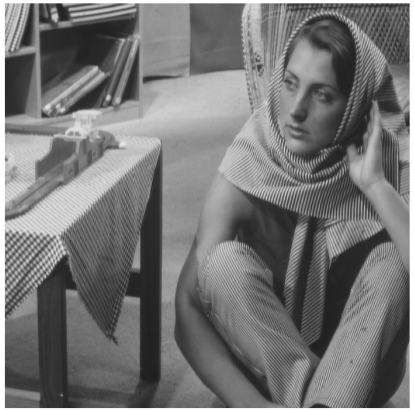
Lossy Coding – JPEG AC Coefficient Handling

- AC vector contains **many** zeros
- Using Run Length Coding (RLC) results in considerable compression
- The AC vector is converted into 2-tuples (skip, value) where
  - **Skip** = number of zeros preceding a non-zero value
  - **Value** = the following non-zero value
- The AC pairs are then Huffman coded

Lossy Coding – JPEG Decoding

- **Decoding** is achieved by reversing Huffman coding, RLC and DPCM to recreate $\tilde{I}_k$
- Then **multiply by normalization array** to create lossy DCT $\tilde{I}_k^{lossy} = Q \otimes \tilde{I}_k$
- The decoded image block is the IDCT of the result $I_k^{lossy} = \text{IDCT}[\tilde{I}_k^{lossy}]$
- The **overall decoded image** is recreated by putting together the compressed 8×8 pieces: $I^{lossy} = [I_k^{lossy}]$

Lossy Coding – JPEG Example



(a) Original

(b) 16:1

(c) 32:1

(d) 64:1

Block Motion Estimation

- **Image motion estimation,** as in optical flow, is important for many video applications including video filtering, motion compensation, and video compression
- Practical systems usually use **block motion estimation** for **ease of implementation**
- This approach assumes that the video consists of images containing **moving blocks**
- The blocks are assumed to have simple **translational motion**. We can again use **windows** and **windowed sets** to express this

Video Notation

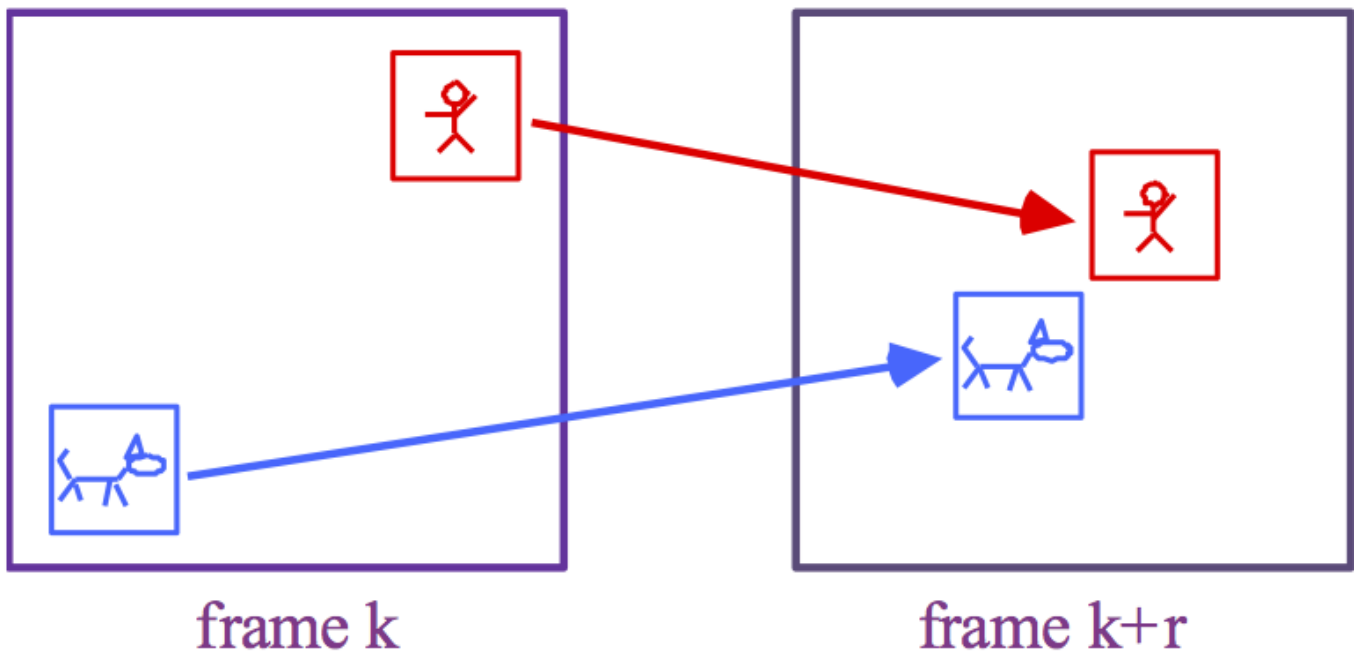
- A **video sequences I** is a 3-D array or signal:
  $\mathbf{I} = [I(i,j,k); 0 \leq i \leq N-1, 0 \leq j \leq M-1, 0 \leq k \leq K-1]$
- Here, we will regard **still images** as single images taken from a video. Thus a video consists of a sequence of still image:
  $\mathbf{I} = [\ldots \mathbf{I}_{k-1} \mathbf{I}_k \mathbf{I}_{k+1} \ldots]$

Video Windows

- A window $\mathbf{B}$ is a set of **2-D coordinate shifts** $\mathbf{B_i} = (m_i, n_i)$: $\mathbf{B} = \{\mathbf{B_1}, \ldots, \mathbf{B_{2M+1}}\} = \{(m_1, n_1), \ldots, (m_{2M+1}, n_{2N+1})\}$
- Given an image $\mathbf{I}_k$ and a window $\mathbf{B}$, the **windowed set** at (i, j, k) is
  $\mathbf{B} \diamond \mathbf{I}(i, j, k) = \mathbf{B} \diamond \mathbf{I}_k(i, j) = \{I(i - m, j - n, k); (m, n) \in \mathbf{B}\}$
  the **set of image pixels** covered by $\mathbf{B}$ at coordinate (i, j) at time k
- The windows used are **SQUARE** and **non-overlapping** for **ease of implementation**
- Standards typically use $\mathbf{16 \times 16}$

Translational Block Motion

- This assumes that at **some later time** $k + r$, each block or windowed set at time k has translated in the i and j directions.
  $\mathbf{B} \diamond \mathbf{I}(\mathbf{i}, \mathbf{j}, \mathbf{k}) = \mathbf{B} \diamond \mathbf{I}(\mathbf{i} + \mathbf{d_1}, \mathbf{j} + \mathbf{d_2}, \mathbf{k} + \mathbf{r})$
  for integer displacement (d_1, d_2) and time shift r, as depicted.



Translational Block Motion

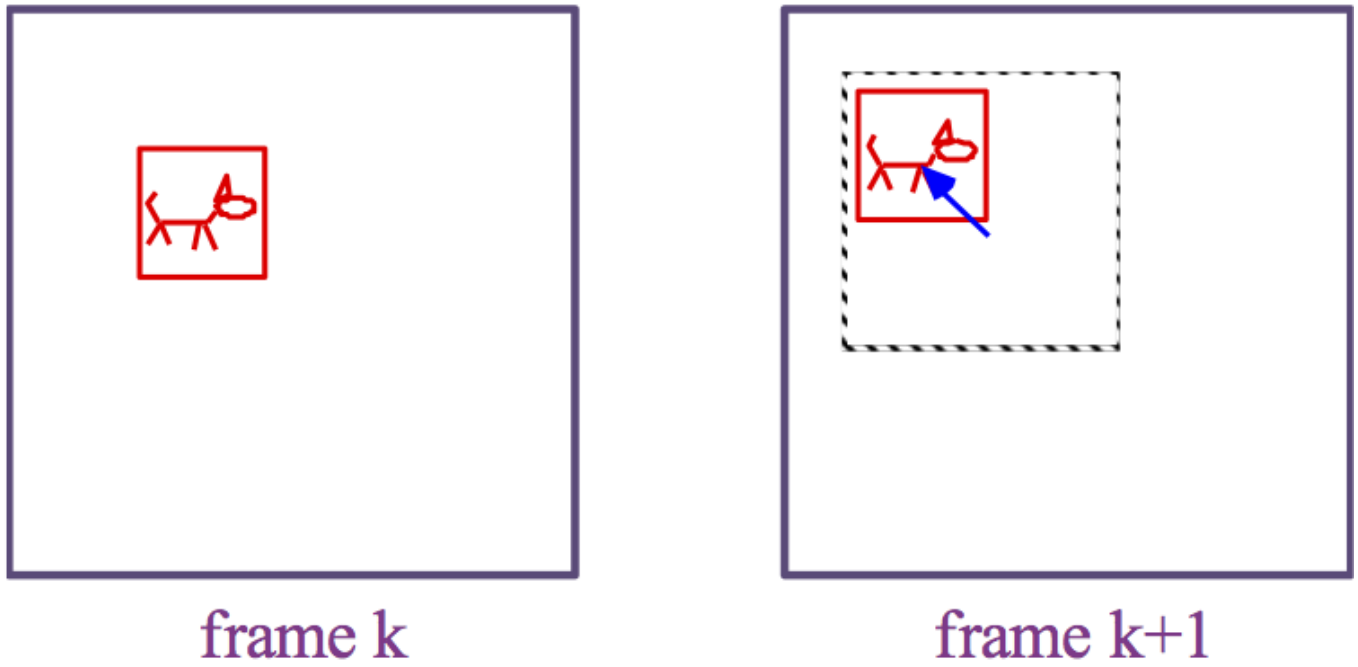
- **Advantages** of translational block models:
  - Only one motion vector needed per block
  - Ease of hardware implementation
- **Disadvantages** of translational block models:
  - Inaccurate for other motion types: zoom, rotation, bending etc.
  - Leads to visual "blocking artifacts" at low bitrates
- It is possible to estimate other motion types, but at much higher cost. Standards use the simple model

Block Matching

- **Goal:** Estimate (d_1, d_2) by **block matching** - the **simplest** method for estimating block motion
- Involves a **simple search** to find the best-fitting translational motion for each block
- Method: **for each block** $\mathbf{B} \diamond \mathbf{I}(\mathbf{i}, \mathbf{j}, \mathbf{k})$ in video signal $\mathbf{I}$ at time k, **search** for the **best-fitting block of the same size** at time $k + 1$
- The blocks that are found at time $k + 1$ **may overlap**

Search Space

The search is conducted over a neighborhood centered around the location (i, j) of the original block:



Block Match Measures

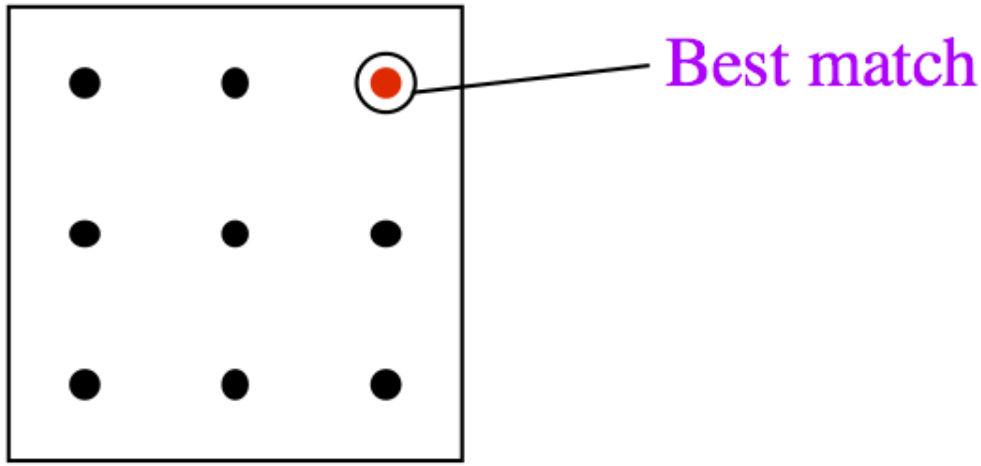
- **Goal:** Find the block with the **minimum error** with respect to the original block:
  FIND:$\min_{(d_1,d_2)} ||\mathbf{B} \diamond \mathbf{I}(i,j,k) - \mathbf{B} \diamond \mathbf{I}(i+d_1, j+d_2, k+1)||$
  where $||.||$ is an error metric such as (assume $P \times Q$ blocks):
  $\text{MSE}(d_1,d_2) = \frac{1}{PQ} \sum \sum_{(m,n)\in\mathbf{B}} [I(i-m,j-n,k) - I(i-m+d_1, j-n+d_2, k+1)]^2$
  $\text{MAD}(d_1,d_2) = \frac{1}{PQ} \sum \sum_{(m,n)\in\mathbf{B}} |I(i-m,j-n,k) - I(i-m+d_1, j-n+d_2, k+1)|$
- **MAD** is commonly used in practice - no computation of squares:
  $(d_1^*, d_2^*) = \arg\min_{(d_1,d_2)} \text{MAD}(d_1,d_2)$

Block Searching

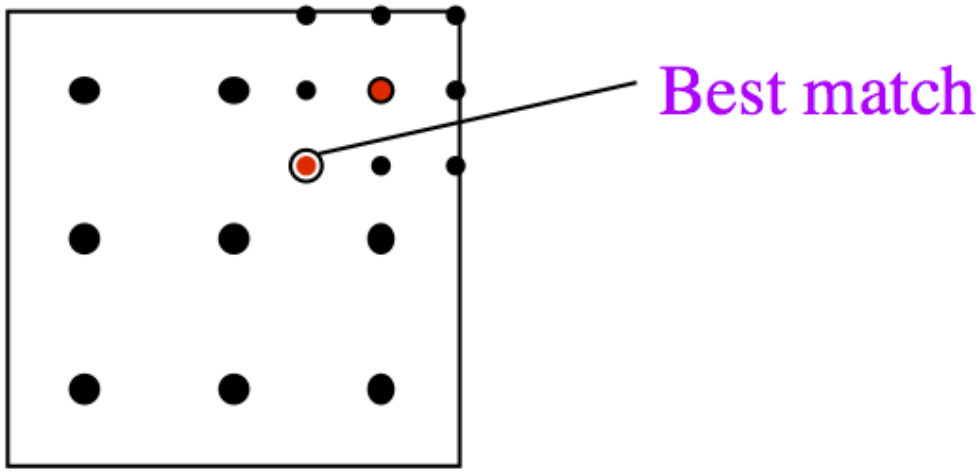
- In practice it is far too time consuming to check for **all** possible matches. Instead, a **subset** of matches is checked
- First, the amount of translation is always limited: $-M \leq d_1, d_2, \leq M$
- **Three-step search** is a **typical** strategy that is used. It involves narrowing down the best location using a directed search
- However, sub-optimal

Three-Step Search

- Step 1: Compute error at $d_1 = d_2 = 0$ at 9 **equally-spaced pixels:**

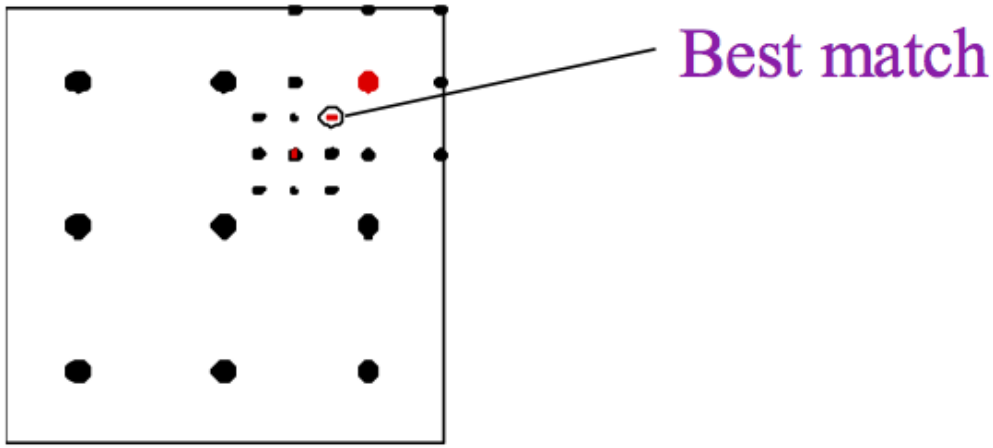


- Step 2: Localize the search near the best match from Step 1:

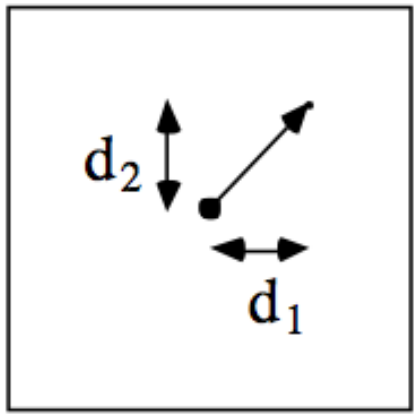


Three-Step Search

- Step 3: Localize the search near the best match from Step 2:



- The **motion estimate** is then simply **the displacement** between the current block (time k) and the best match (time $k+1$)



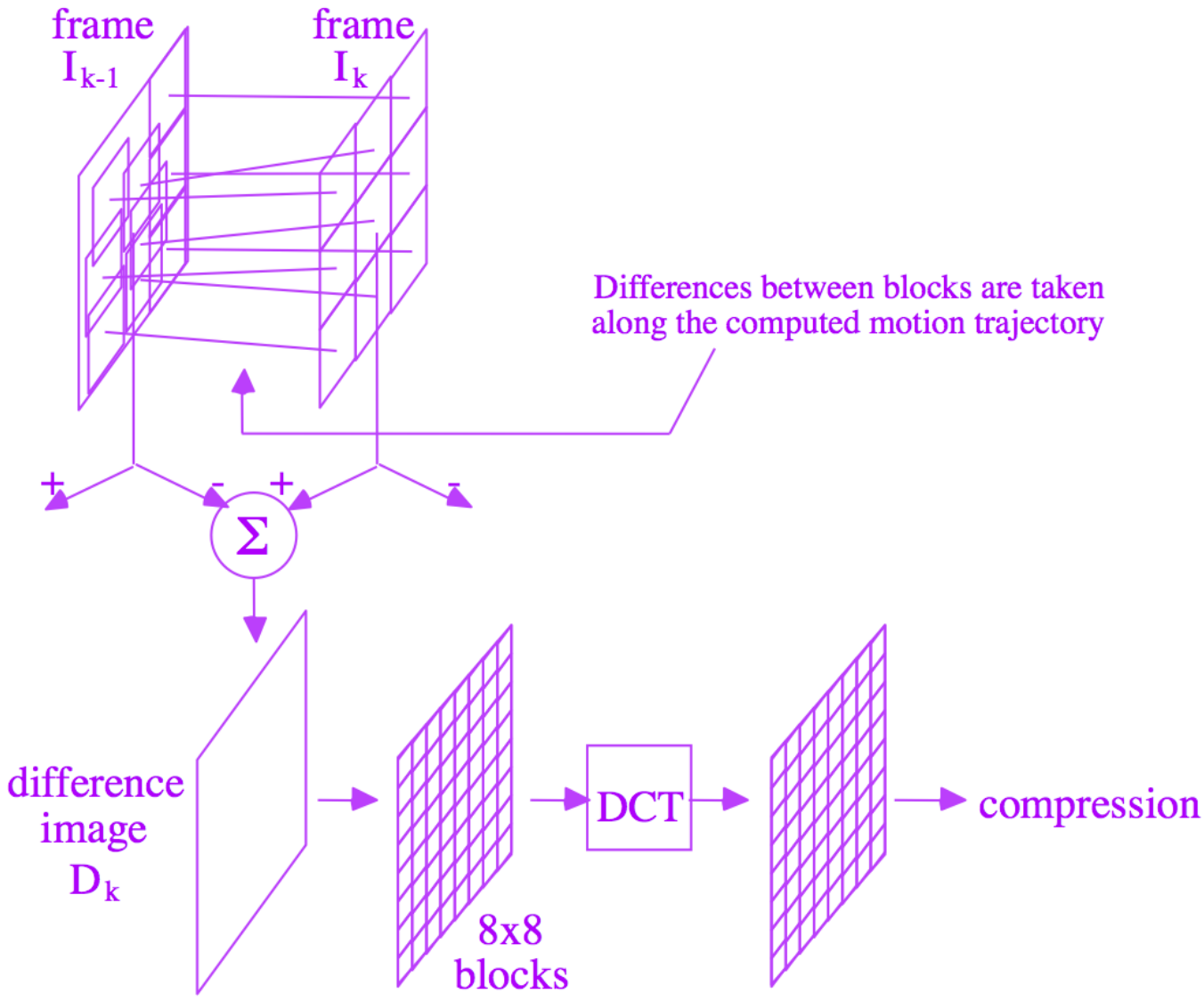
Comments on Match Search

- Discussed **forward** motion estimation between current frame k and next frame $k+1$
- **Backward** motion estimation between current frame k and previous frame $k-1$ common
- Block-based motion estimation is really a form of **optical flow**
- Current (H.264, H.265) video coding standards use block-based motion estimation

Motion-Compensated Transform Coding

- Compute block motion displacement vectors using **loopback** from frame I_k to frame I_{k-1}. Usually 16×16 blocks. Blocks are non-overlapping in frame I_k. This is referred to as **interframe coding**
- Compute **motion-compensated difference image** D_k by differencing each 16×16 block in I_k with its corresponding displaced block in I_{k-1}
- Subdivide difference image D_k into sub-blocks (usually 8×8) and code using **JPEG-like algorithm**
- The first frame is coded like an image. This is referred to as **intraframe coding**

Motion-Compensated Transform Coding



Comments on Motion Compensation

- **Motion compensation (MC)** is highly effective for:
  - **Increasing** compression efficiency
  - Reducing "**ghosting**" artifacts in compressed video. Very fast movements result in large differences between blocks. Without MC, leads to "**motion ghosts**"
  - Accomodating compression to **temporal aliasing**
- Current standards (H.264, H.265) use MC

Practical Video Codec – The Basics

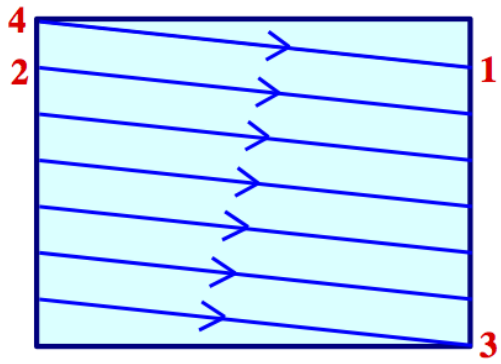
- Earlier, defined **video** as **time-indexed images**
- Can **sample** in all three dimensions, yielding **discrete video**. This is always **quantized**, which is **digital video**
- In principle, **analog video** is continuous in all three dimensions
- In **practice**, analog video is sampled along one spatial dimension and along time dimension

Practical Video Codec – Analog Video

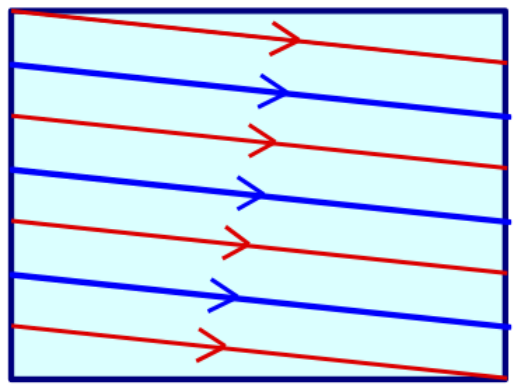
- An **optical** analog video signal is a function $I_c(x, y, t)$ of space and time
- Practical video systems, such as **television** and monitors, represent analog video as a **one-dimensional electrical signal** $V(t)$
- A 1-D signal $V(t)$ is obtained by sampling $I_c(x, y, t)$ along the vertical (y) direction and along time (t) direction. This is called **scanning** and the result is a series of **scan lines**

Analog Video Sampling

- **Progressive Analog Video** involves sampling row after row at intervals Δy and each frame at intervals Δt



- **Interlaced Analog Video** involves sampling even and odd rows alternately



Digital Video Sampling

- Digital video is obtained either by sampling an analog video signal or by directly sampling the 3-D intensity distribution
- If **progressive analog video** is sampled, or if digital video is **directly sampled**, then the sampling is **rectangular** and properly indexed
- If **interlaced analog video** is sampled, then the digital is interlaced also and must be re-indexed

Practical Video Codec – TV Standards

- **NTSC (National Television Systems Committee)**
  - 2:1 interlaced
  - 525 lines per frame (262.5 / refresh) - 485 active, 40 blank
  - 60 refreshes / second
  - Used heavily in Japan and North America
- **PAL (Phase Alternation Line)**
  - 2:1 interlaced
  - 625 lines per frame
  - 50 refreshes / second
  - Used heavily in Europe and Asia (including India)
  - Older tube TVs used this format

Practical Video Codec – Aspect Ratio

Aspect ratio: The ratio of the width of a video frame to its height

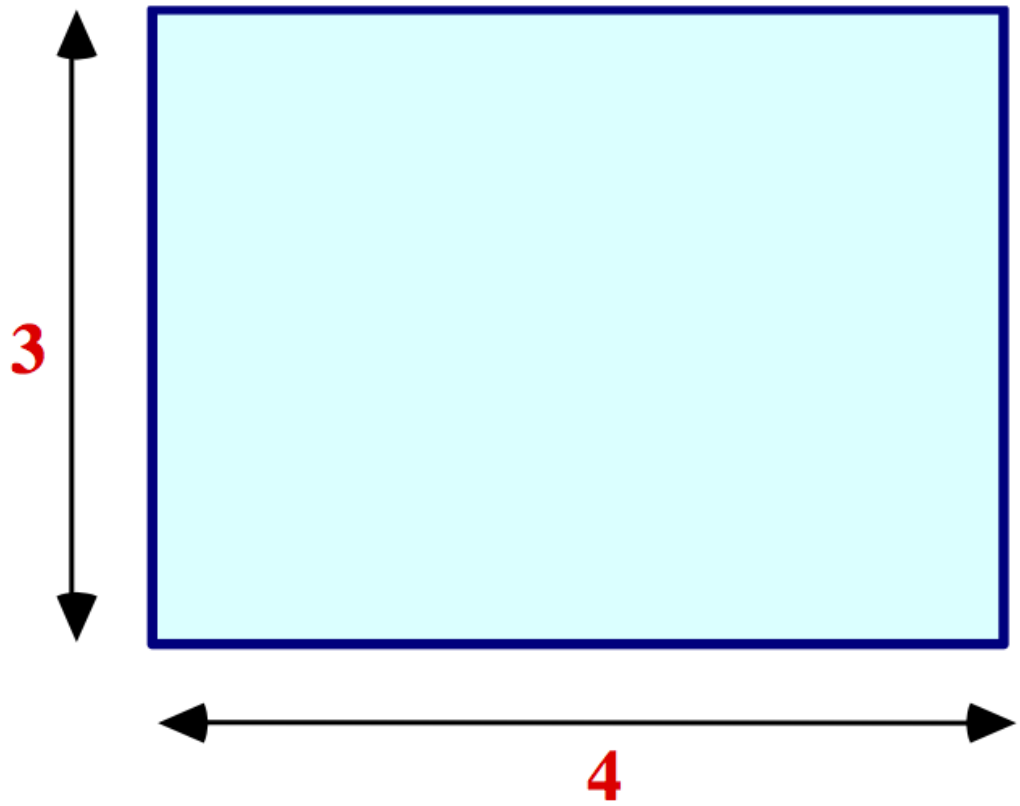


Figure: Analog formats used 4:3 aspect ratio

Practical Video Codec – Color Basics

- Any color can be represented as a combination of Red (R), Blue (B), and Green (G)
- **RGB representation** codes a color video as three separate signals
- The **YIQ representation** combines the information according to perceptual criteria:
  $Y = 0.299R + 0.587G + 0.114B$ (luminance)
  $I = 0.596R + 0.275G - 0.321B$ (chrominance)
  $Q = 0.212R - 0.523G + 0.311B$ (chrominance)
- **Alternately, YCbCr** chrominance representation:
  $Cr = R - Y$ (chrominance)
  $Cb = B - Y$ (chrominance)
- Why bother? **Compression! Chrominance** information can be sent in a **fraction** of information required for luminance information

Practical Video Codec – Resolution

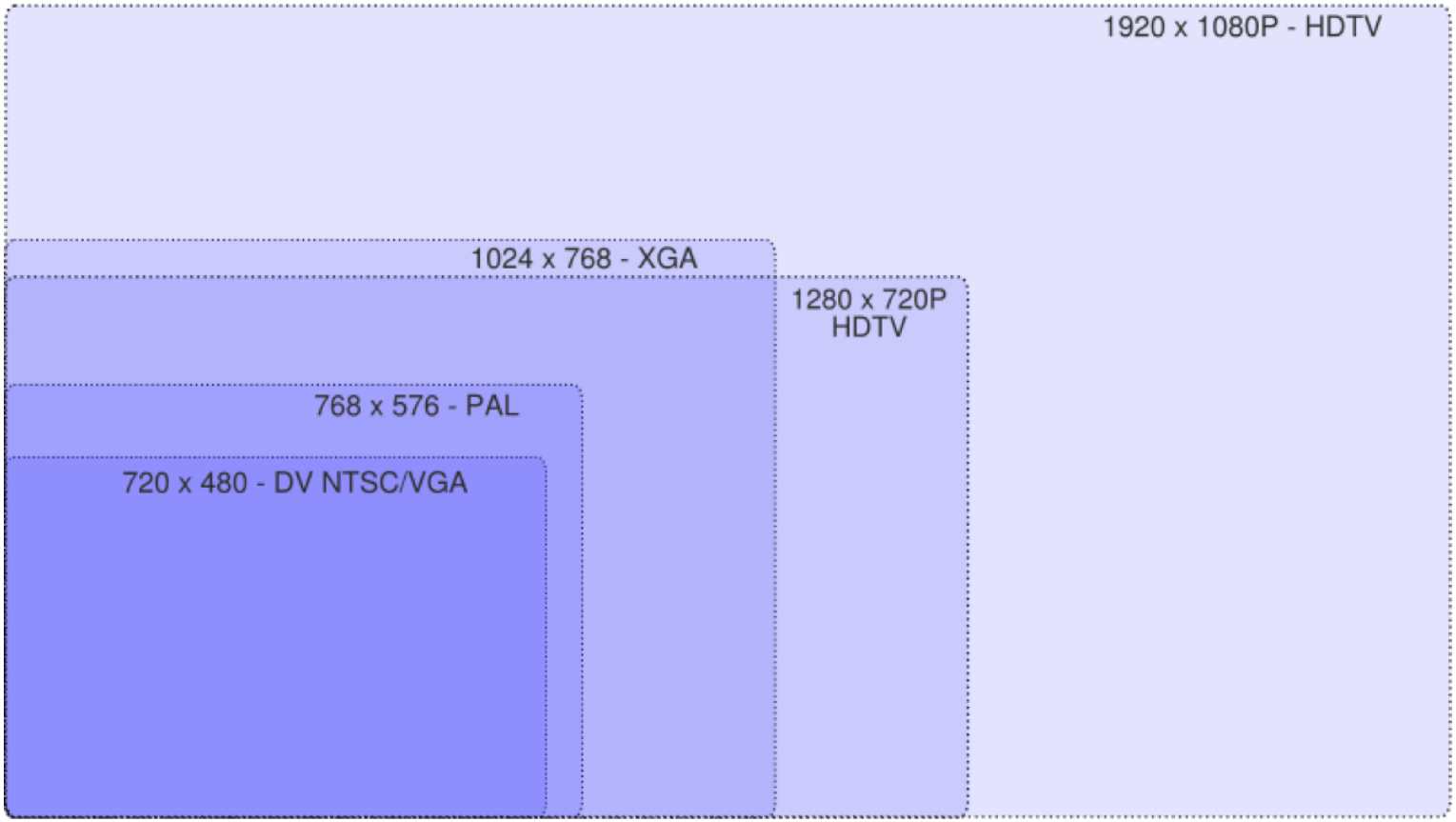


Figure: Typical modern video resolutions

Practical Video Codec – HDTV Resolutions

The HDTV format:

- Has **interlaced** and **progressive** modes
- **720p:** progressive, 1280×720 pixels, 60 frames per second (fps). Raw BW (24 bits/pixel): **1.3 Gbps**
- **1080i:** interlaced, 1920×1080 pixels, 50 fields (25 frames) per second (fps). Raw BW (24 bits/pixel): **1.2 Gbps**
- **1080p:** progressive, 1920×1080 pixels, 59.94 fps. Raw BW (24 bits/pixel): **2.98 Gbps**
- Aspect ratio: **16:9**
- Typically **compressed:** MPEG-2 or H.264

Practical Video Codec – Group of Pictures (GOP)

- Specifies the order in which Intra (I) and Inter (P, B) frames are arranged in a video sequence

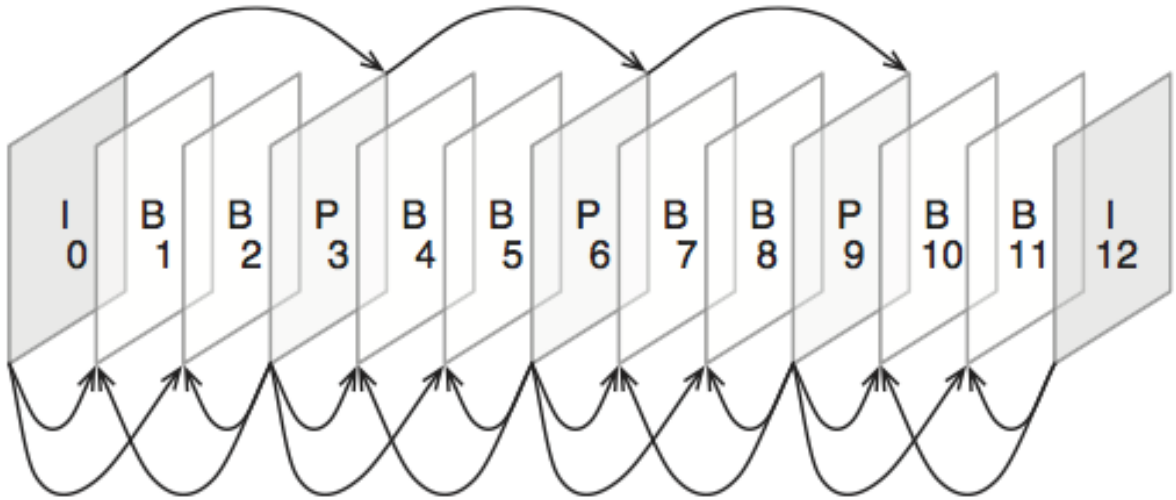


Figure: Traditional GOP structure

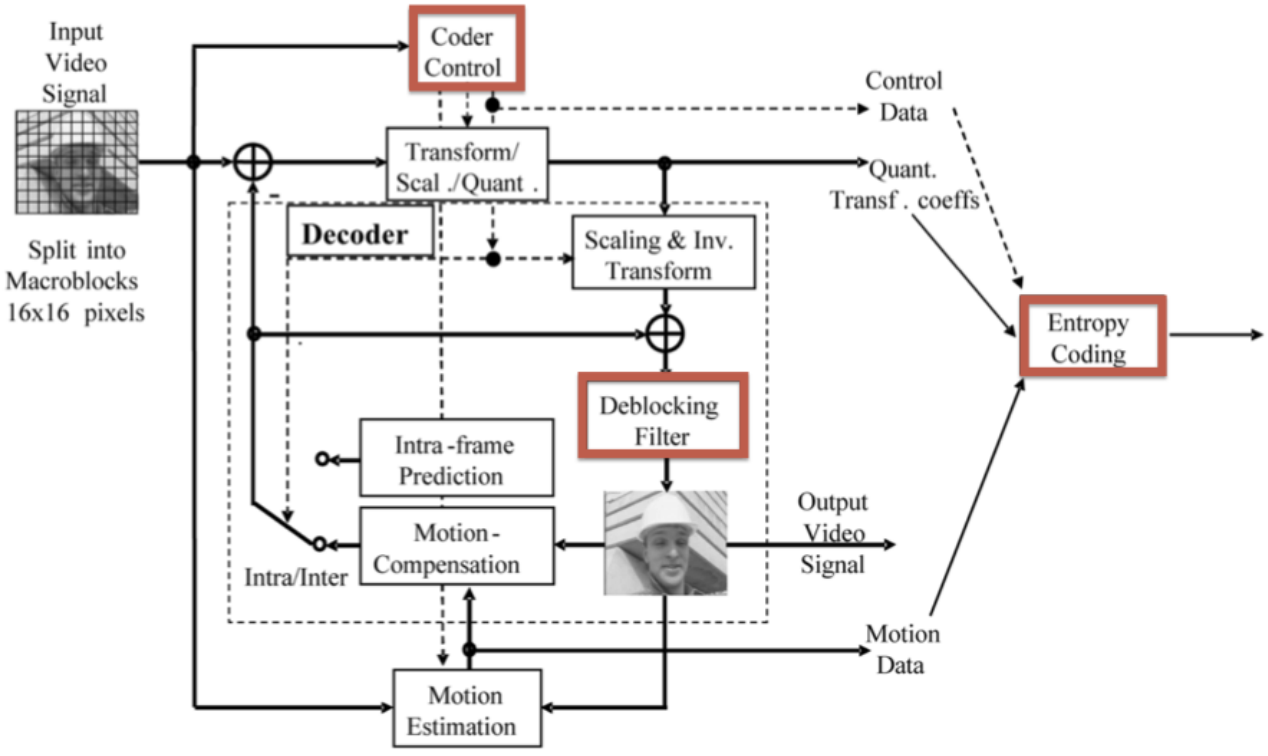
Video Compression Standard - H.264

- Standardized in 2003
- A large, **complex** video standard
- High-level overview here
- Good reference: "The H.264 Advanced Video Compression Standard" by Iain E. Richardson, Wiley, 2010

H.264 - The Highlights

- Variable macroblock sizes - better motion estimation
- In-loop filtering - reduces blocking artifacts
- Integer transform - efficient implementation
- Improved lossless coding - CABAC
- Network Abstraction Layer (NAL) units - facilitates network transport

H.264 - The Highlights



Optical Flow

- Fundamental to the concept of motion, but nevertheless different, is **optical flow**
- Optical flow is the **instantaneous motion of image intensities**. This is **not** the same as the motion of the objects being imaged: **image motion is not object motion**
- Examples:
  - An "off-camera" variable light source illuminating a **stationary object**. A case of **image motion without object motion**
  - A mirrored sphere that is spinning. A case of **object motion without image motion**
- Still, optical flow is all the motion information that the image supplies! So, most methods of **motion estimation, motion compensation,** etc. depend on it

Optical Flow – Continuous Formulation

- The image intensity at a point in space and time is $I(x, y, t)$
- After a sufficiently **small** time interval Δt, the intensity at (x, y) will move to a point $(x + \Delta x, y + \Delta y)$. In other words: $I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t)$
- Illustration on board ...
- This assumes that the intensity does not change, just its position

Optical Flow – Taylor Expansion

- Expanding the LHS in a Taylor's series:
  $I(x + \Delta x, y + \Delta y, t + \Delta t) =$
  $I(x, y, t) + \Delta x \frac{\partial I}{\partial x} + \Delta y \frac{\partial I}{\partial y} + \Delta t \frac{\partial I}{\partial t} +$ higher order terms
- So that
  $I(x, y, t) + \Delta x \frac{\partial I}{\partial x} + \Delta y \frac{\partial I}{\partial y} + \Delta t \frac{\partial I}{\partial t} +$ higher order terms $=$
  $I(x, y, t)$
- Letting higher order terms to 0 (assuming small time and motion), cancelling $I(x, y, t)$ and dividing by Δt
  $\frac{\Delta x}{\Delta t} \frac{\partial I}{\partial x} + \frac{\Delta y}{\Delta t} \frac{\partial I}{\partial y} + \frac{\partial I}{\partial t} = 0$

Optical Flow Constraint Equation

- Taking the limit $\Delta t \to 0$ yields:
  $\frac{\partial x}{\partial t}\frac{\partial I}{\partial x} + \frac{\partial y}{\partial t}\frac{\partial I}{\partial y} + \frac{\partial I}{\partial t} = 0$
- The **optical flow** components are:
  $u(x, y, t) = \frac{\partial x}{\partial t}(x, y, t), v(x, y, t) = \frac{\partial y}{\partial t}(x, y, t)$
- Putting this together gives the **optical flow constraint equation** or **OFCE**:
  $I_x u + I_y v + I_t = 0$
- So-called since it does not solve for optical flow - only **constrains** the optical flow vector (u, v) to lie on a line

Optical Flow – The Aperture Problem

- Even knowing I_x, I_y, I_t does not solve optical flow
- This is the **aperture problem**
- Image being able to view only a **small region** of the image that is in motion:

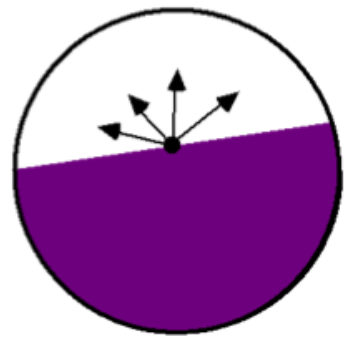


Figure: Aperture problem

- If the edge is sensed to be moving "up", the true motion could actually be in any of the shown directions
- In order to solve for optical flow, some other **physically meaningful** constraints must be found or assumed

Smooth Optical Flow

- The assumption that is usually made is that optical flow is **smooth**. By smooth is meant that the derivatives of u and v have small magnitudes
- Solution involves minimizing the overall **departure** from smoothness: $E_{smooth} = E_s = \int \int_{image} [u_x^2 + u_y^2 + v_x^2 + v_y^2] dxdy$
- We also want the overall OFCE error to be small: $E_c = \int \int_{image} [I_x u + I_y v + I_t]^2 dxdy$

A Minimization Problem

- Minimize the **weighted sum:**
  $E = E_s + \lambda E_c$
- A solution will always **exist** and be **unique**
- For larger λ, the solution will track OFCE closely
- For smaller λ, the solution will be forced smoother
- Picking λ is a **hard problem** – not discussed here
- We will not try to solve the **continuous** problem

Discrete Optical Flow

- **Approximations of derivatives** of flow:
  $u_x \approx [u(i+1,j) - u(i,j)]/2, u_y \approx [u(i,j+1) - u(i,j)]/2$
  $v_x \approx [v(i+1,j) - v(i,j)]/2, v_y \approx [v(i,j+1) - v(i,j)]/2$
- Then
  $$E_s = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \{[u(i+1,j) - u(i,j)]^2 + [u(i,j+1) - u(i,j)]^2 +$$
  $$[v(i+1,j) - v(i,j)]^2 + [v(i,j+1) - v(i,j)]^2\},$$
  $$E_c = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} [I_x(i,j)u(i,j) + I_y(i,j)v(i,j) + I_t(i,j)]^2$$
- The estimates $I_x(i,j), I_y(i,j), I_t(i,j)$ will be discussed soon ...

Discrete Optimization

- The goal is to **minimize:** $E = E_s + \lambda E_c$
- Take derivatives w.r.t. $u(i,j), v(i,j)$ for $0 \leq i \leq N-1, 0 \leq j \leq M-1$:
  $\frac{\partial E}{\partial u(i,j)} = 2[u(i,j) - u_{ave}(i,j)] + 2\lambda[I_x u(i,j) + I_y v(i,j) + I_t]I_x$
  $\frac{\partial E}{\partial v(i,j)} = 2[v(i,j) - v_{ave}(i,j)] + 2\lambda[I_x u(i,j) + I_y v(i,j) + I_t]I_y$
- The local **4-averages** are:
  $u_{ave}(i,j) = \frac{1}{4}[u(i+1,j) + u(i-1,j) + u(i,j+1) + u(i,j-1)]$
  $v_{ave}(i,j) = \frac{1}{4}[v(i+1,j) + v(i-1,j) + v(i,j+1) + v(i,j-1)]$

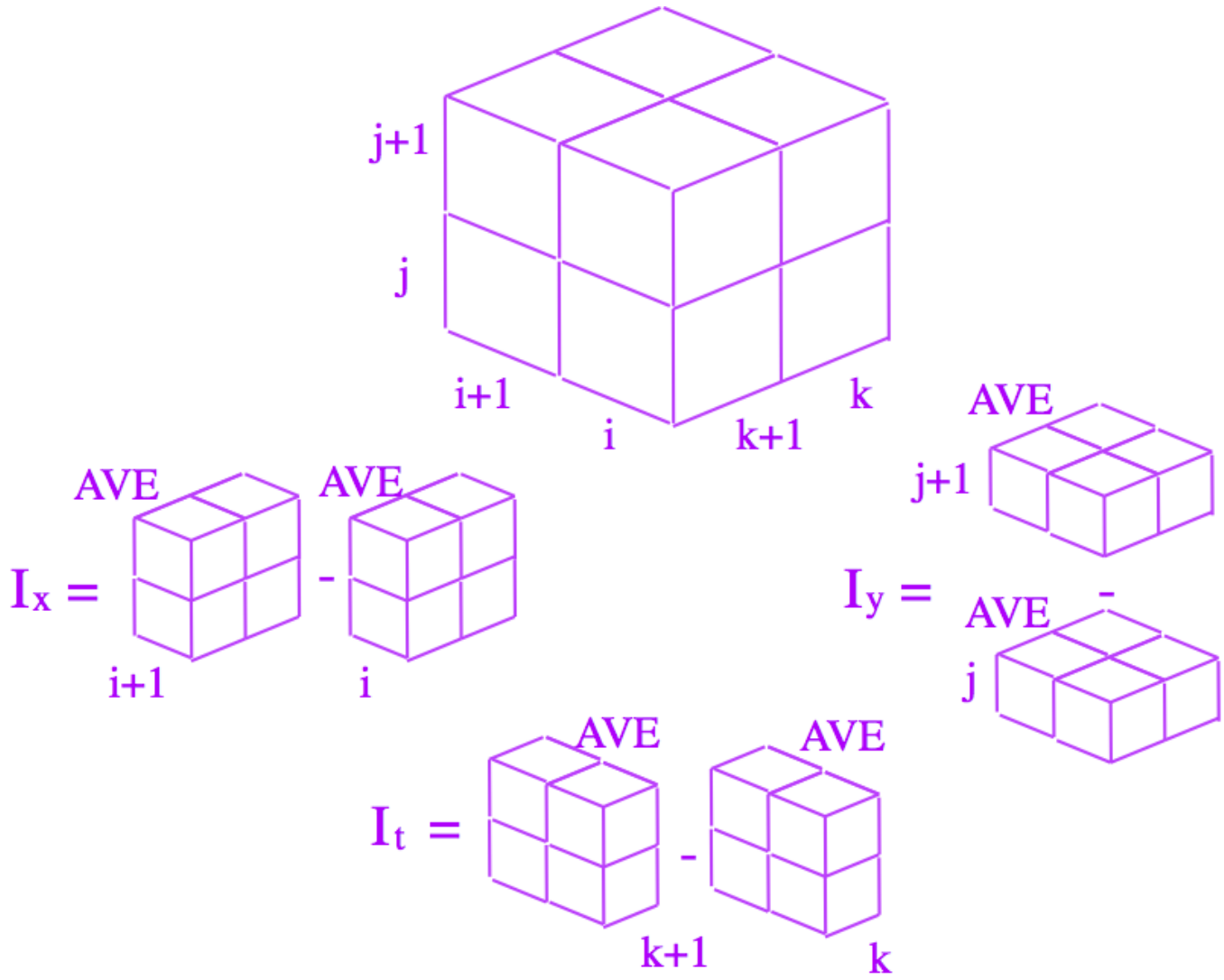
Discrete Solution

- The minima occur when the derivatives are zero:
  $\frac{\partial E}{\partial u(i,j)} = \frac{\partial E}{\partial v(i,j)} = 0$
- This results in:
  $(1 + \lambda I_x^2)u(i,j) + \lambda I_x I_y v(i,j) = v_{ave}(i,j) - \lambda I_x I_t$
  $(1 + \lambda I_y^2)v(i,j) + \lambda I_x I_y u(i,j) = v_{ave}(i,j) - \lambda I_y I_t$
- Solving for $u(i,j), v(i,j)$ yield:
  $u(i,j) = u_{ave}(i,j) - \lambda \frac{I_x u_{ave}(i,j) + I_y v_{ave}(i,j) + I_t}{1+\lambda(I_x^2+I_y^2)} . I_x$
  $v(i,j) = v_{ave}(i,j) - \lambda \frac{I_x u_{ave}(i,j) + I_y v_{ave}(i,j) + I_t}{1+\lambda(I_x^2+I_y^2)} . I_y$

Intensity Gradient Estimation

- The derivatives I_x, I_y, I_t can also be estimated as **differences-of-averages** across a $2 \times 2 \times 2$ data cube:
- $I_x \approx \frac{1}{4}[I(i+1,j,k)+I(i+1,j,k+1)+I(i+1,j+1,k)+I(i+1,j+1,k+1)]-[I(i,j,k)+I(i,j,k+1)+I(i,j+1,k)+I(i,j+1,k+1)]$
- $I_y \approx \frac{1}{4}[I(i,j+1,k)+I(i,j+1,k+1)+I(i+1,j+1,k)+I(i+1,j+1,k+1)]-[I(i,j,k)+I(i,j,k+1)+I(i+1,j+1,k)+I(i+1,j,k+1)]$
- $I_t \approx \frac{1}{4}[I(i,j,k+1)+I(i,j+1,k+1)+I(i+1,j,k+1)+I(i+1,j+1,k+1)]-[I(i,j,k)+I(i,j+1,k)+I(i+1,j,k)+I(i+1,j+1,k)]$

Intensity Gradient Estimation



Iterative Solution

- The solution for $u(i,j)$ and $v(i,j)$ suggests a numerical algorithm for actually computing them. The **relaxation algorithm** is:

$$u^{(p+1)}(i,j) = u^{p}_{ave}(i,j) - \lambda \frac{I_x u^{(p)}_{ave}(i,j) + I_y v^{(p)}_{ave}(i,j) + I_t}{1+\lambda(I_x^2+I_y^2)} . I_x$$

$$v^{(p+1)}(i,j) = v^{p}_{ave}(i,j) - \lambda \frac{I_x u^{(p)}_{ave}(i,j) + I_y v^{(p)}_{ave}(i,j) + I_t}{1+\lambda(I_x^2+I_y^2)} . I_y$$

- This technique of compute a "new" estimate from "old" estimates is a common technique in numerical analysis called **successive refinement**

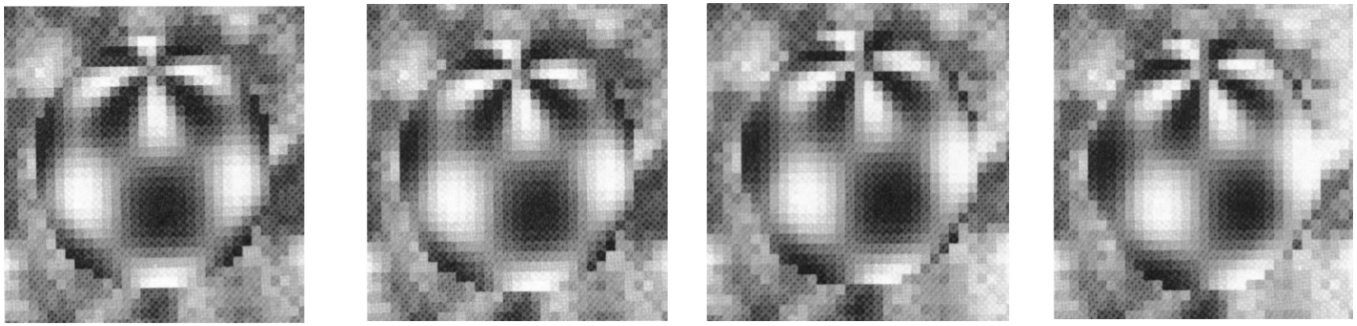
Initial Estimates

- The **initial estimates** $u^{(0)}(i,j), v^{(0)}(i,j)$ might be taken from some **independent estimate** of u, v or simply by taking $u^{(0)}(i,j) = v^{(0)}(i,j) = 0$ which gives
$$u^{(1)}(i,j) = -\lambda \frac{I_t I_x}{1+\lambda(I_x^2+I_y^2)}$$
$$v^{(1)}(i,j) = -\lambda \frac{I_t I_y}{1+\lambda(I_x^2+I_y^2)}$$

Iteration Limit

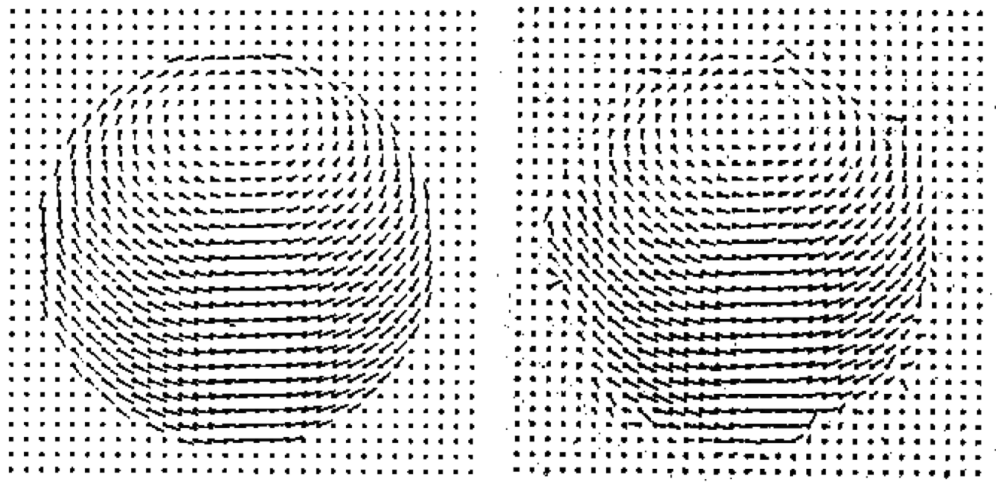
- The **iteration** are continued either:
  - for a prescribed number P of iterations
  - until iterating doesn't change the solution much e.g., $\max_{(i,j)}|u^{(p+1)}(i,j) - u^{(p)}(i,j)| < \epsilon$ where ϵ is a tolerance threshold
- Although in **principle** it could take N iterations for the constraints to propagate across the image domain, in **practice** it take just a few iterations due to the **localness** of image motion

Optical Flow Example



$t = t_0$ $t = t_0 + \Delta t$ $t = t_0 + 2\Delta t$ $t = t_0 + 3\Delta t$

Optical Flow Example



(a) Original Flow (b) Estimated Flow

Needle diagram: Arrow direction indicates flow direction and its length indicates flow magnitude

Optical Flow Example

- Results are accurate in most places but **errors occur near the sphere boundary**
- Error occur near **flow discontinuities** - the smoothness conditions is inaccurate
- This is the **Horn-Schunk Algorithm,** the first and still classic approach
- Many sophisticated techniques exist e.g., the attempt to **find flow discontinuites**, then **disable** the **smoothness** constraint there