Multimedia Software Systems CS4551

Basic Video Compression Technique

What is Video Compression?

- Image compression
  - to reduce spatial redundancy
- Video compression
  - In a video, spatial redundancy exists in each frame like images.
  - In addition, temporal redundancy (redundancy between frames) exists and can be exploited for compression reasons.
  - Video compression is to reduce spatial redundancy within a frame AND temporal redundancy between frames.
  - Each video frame may be encoded differently depending on whether to use spatial or temporal redundancies.

Modalities of Video Compression

- Depending on which mode (spatial or temporal) you want to exploit for compression reasons, you have two modes of compression for video
- Intraframe (I frame)
  - Each frame is encoded as an individual entity (like an "image")
  - Uses Image Compression techniques (eg DCT)
- Interframe (P frame, B frame, or multiframe)
  - Predictive Encoding but for the temporal domain
  - Instead of encoding the current frame directly,we encode the *difference* between the current frame and a *prediction* based on previous frames
  - Term Used –*Motion Compensation*

Intraframe Coding

- These frames are compressed using a combination of a Lossy Scheme such as Transform Coding or Subsampling/Quantization and Lossless Entropy Coding such Huffman or Arithmetic.
- For example, the MPEG/ITU Standard compresses these frames as discussed in JPEG image compression
  - Get 8x8 blocks (for each component)
  - DCT on each block
  - Quantization of all coefficients
  - AC zigzag ordering
  - DC=>DPCM=>(size) (value)
  - AC=>Runlength Encoded=>(runlength, size) (value)
  - Both AC and DC Coefficients get Huffman encoded to form a bit stream

Changes from Frame to Frame

- What can happen to a pixel (or pixel region) from one frame to another?
  - Nothing! –Like an unchanging background –so do we need to encode information here?
  - Changes (slight) due to quantization and noise
  - Changes due to the motion of the object
  - Changes due to motion of the camera
  - Changes due to environment and lighting
- If nothing has changed –no need to encode
- If changes in pixel color or pixel value due to motion of object or camera, maybe we can predict how the pixels have moved, thereby needing to encode only the change vector

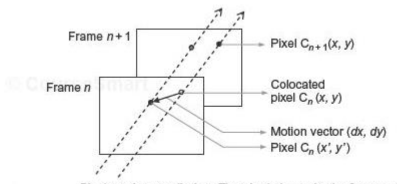
Motion Vector

- Let's assume that that if a pixel has moved from frame to frame, the color of the pixel has not changed.
- In other words: A point at *(x,y)* in frame *n+1* with color $c_{n+1}(x,y)$ corresponds to some point *(x',y')* in frame *n* if $c_{n+1}(x , y)$ is same or very similar to $c_n(x' , y')$
- If we call this *displacement* or *motion vector* $d = (d_x, d_y)$ then we have

  $d = (d_x, d_y) = current – previous$

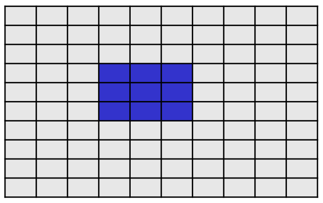
  $= (x, y) – (x', y')$

  $=> d_x = x-x'$ and $d_y = y-y'$

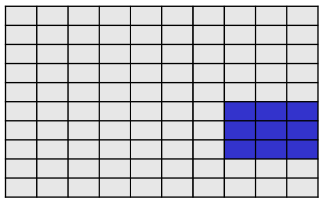


Pixel motion prediction. The pixel shown in the frame n has moved to a new location in frame n + 1. Consequently, $C_{n+1}(x, y)$ in frame n + 1 is not the same as $C_n(x, y)$ but is offset by the motion vector (dx,dy).

Motion Vector



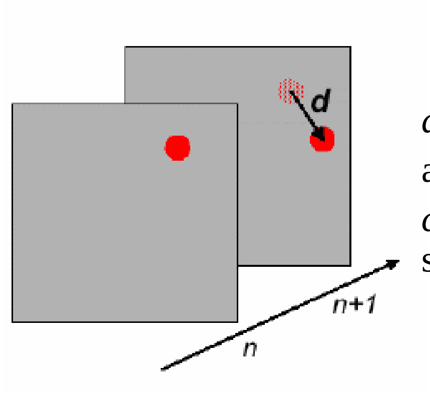
Previous



Current

For the blue block, what is d = (dx, dy) = ?

Motion Vector Example



d = (dx, dy) = (x, y)–(x', y') assuming that $c_{n+1}(x, y)$ is same or very similar to $c_n(x', y')$

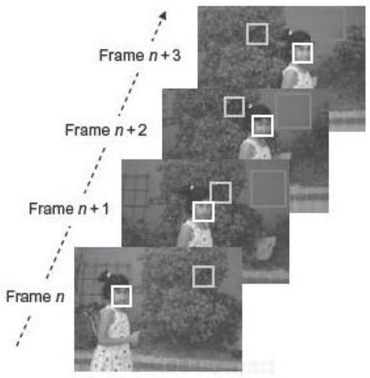
Motion Compensation

- If we compute motion vector for pixels of the current frame with respect to the previous frame, it means that $c_{n+1}(x,y)$ can be predicted from $c_n(x',y')$ => $c_{n+1}(x,y) = c_n(x-d_x, y-d_y)$.
- So instead of storing/transmitting $c_{n+1}(x,y)$ for the current frame, we can store the following two things
  - the motion vectors (d_x, d_y)
  - residual error, $e(x,y) = c_{n+1}(x,y) - c_n(x-d_x, y-d_y)$
- Then, to recover $c_{n+1}(x,y)$, we use the pixel from the previous frame $c_n(x',y')$ which can be computed by $c_n(x-d_x, y-d_y)$, the motion vector and the residual error.
  - $c_{n+1}(x,y) = c_n(x-d_x, y-d_y) + e(x,y)$

Motion Compensation - MacroBlocks

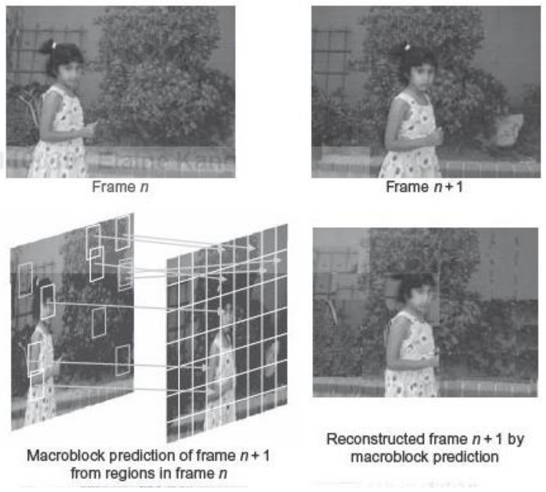
- Do we need to *compute* and *transmit* one motion vector *d* per pixel?
  - Computationally intensive!
  - Lots of data to send
- Instead, transmit only 1 motion vector per groups of pixels called *macroblock* (e.g., 16x16 pixels)
- Advantage & Disadvantages:
  - Fewer motion vectors to be transmitted
  - Faster computationally
  - Less precise motion prediction if motion is not constant within the macroblock (e.g., if macroblock covers the edge of a moving object)

Motion Compensation - MacroBlocks



Temporal redundancy in video. Four frames of a video sequence are shown. The girl in the foreground is moving, whereas the background objects are static; however, the camera is also moving. Although each frame is different, there are areas of the background and the foreground that remain the same or are very similar.

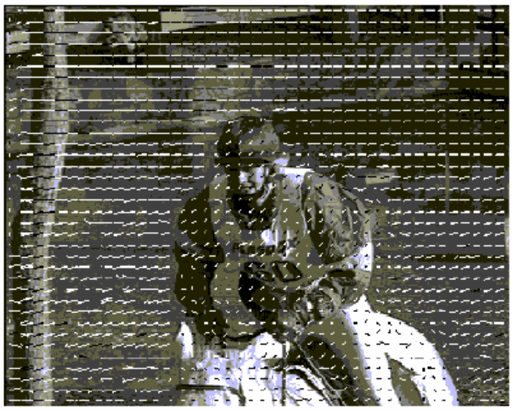
Motion Compensation - MacroBlocks


Frame n
Frame $n+1$
Macroblock prediction of frame $n+1$ from regions in frame n
Reconstructed frame $n+1$ by macroblock prediction

MacroBlock Motion Vectors - Example

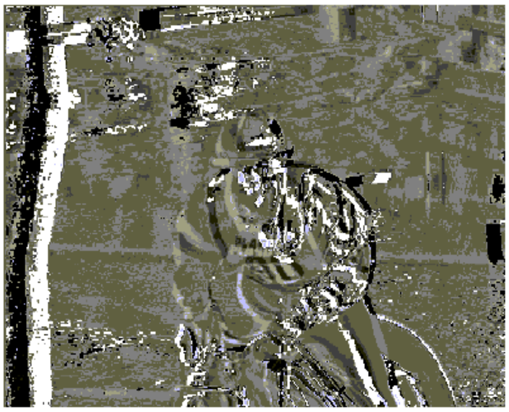
(n-1)th frame

MacroBlock Motion Vectors - Example



nth frame showing motion vector

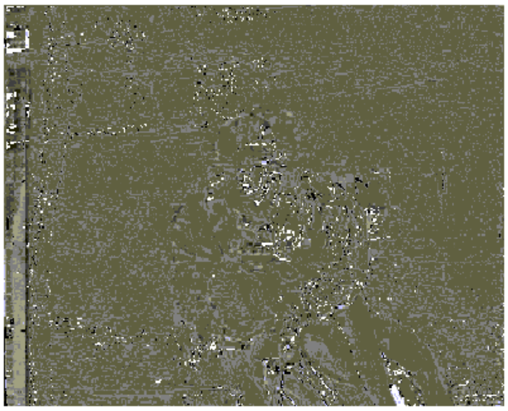
MacroBlock Motion Vectors - Example



Difference between frames *n* and
***n-1*, without motion compensation**

CSULA CS4551 Multimedia Software Systems by Eun-Young Kang

MacroBlock Motion Vectors - Example



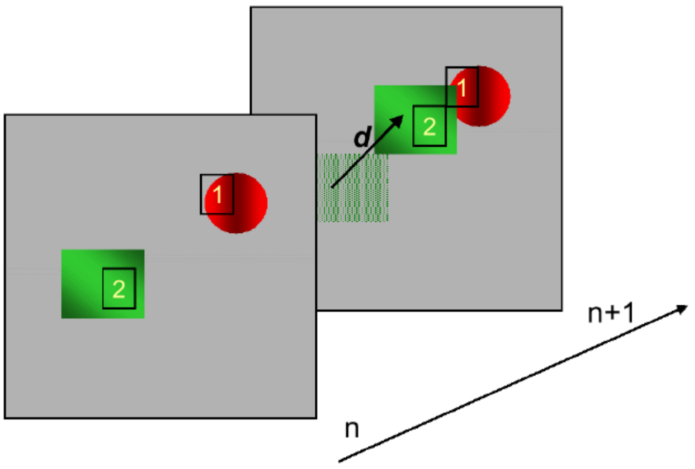
Difference between frames *n* and
***n-1*,** *with motion compensation*

CSULA CS4551 Multimedia Software Systems by Eun-Young Kang

A Note on Frame Segmentation

- This also refers to deciding a *size* and *shape* of these macroblocks that are used in computing motion
- Normally a frame is divided in non-overlapping blocks of a certain size *B*(16x16) and shape (square). Block size and shape affects the performance of compression
- Issues
  - Large *B* => fewer blocks to search =>fewer motion vectors to encode
  - For large *B*, the movement of objects do not coincide with boundaries of *B* => larger errors or residuals that need to be encoded
- Thus block size represents a trade off between minimizing the number of motion vectors and maximizing the quality of the matching blocks

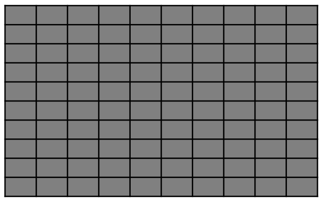
MacroBlocks at Motion Boundary



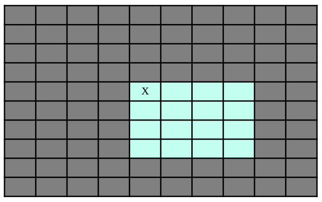
Search and Block Matching

- If the difference between the target block and the candidate block at the same position in the past frame is below some threshold then no motion, else search
- Block matching is the most time consuming –for accurate results you need to do an exhaustive search which is - given a block *B* in the current frame search for a match block *A* in the entire previous frame. This may be further optimized: Instead of entire frame,
  - Limit search to a region
  - Logarithmic Search
  - Hierarchical search
- In practice, this search can be limited to a range around the "target block"

Calculating a Motion Vector per a Target Macroblock



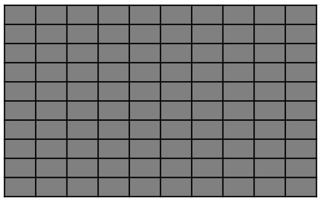
The Previous frame

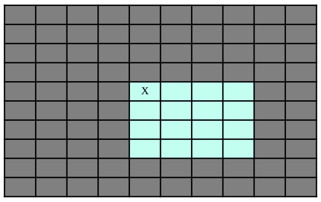


A target macro block at (4,4) at the current frame

Find the best matching block for the target block and calculate the motion vector d = (dx, dy)

For Search Window Size p=1

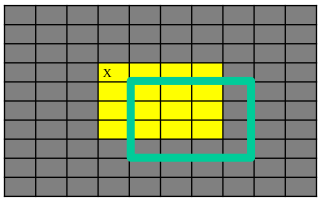

The Previous frame



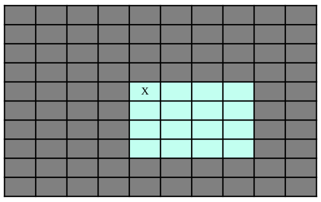
A target macro block at (4,4)
at the current frame

For p=1,
(2p+1)*(2p+1) = 9 matches will be performed.

For Search Window Size p=1



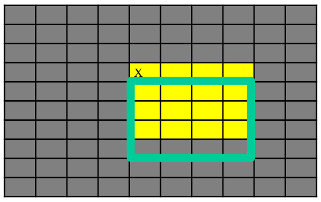
The Previous frame



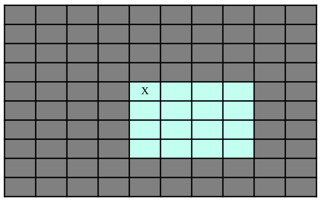
A target macro block at (4,4)
at the current frame

First, match a candidate block at d=(-1,-1)
Computer MSE and store

For Search Window Size p=1



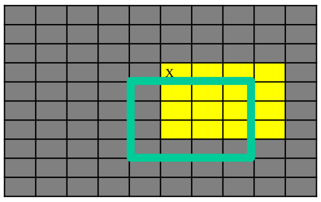
The Previous frame



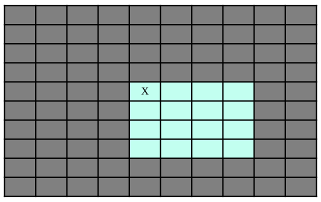
A target macro block at (4,4)
at the current frame

Second, match a candidate block at d=(0,-1)
Computer MSE and store

For Search Window Size p=1



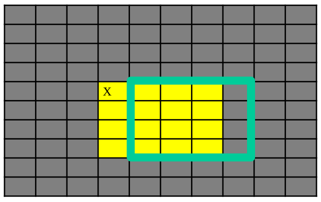
The Previous frame



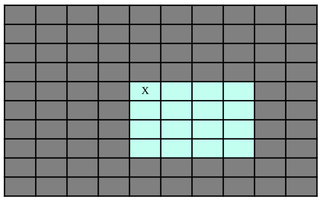
A target macro block at (4,4)
at the current frame

Third, match a candidate block at d=(1,-1)
Computer MSE and store

For Search Window Size p=1



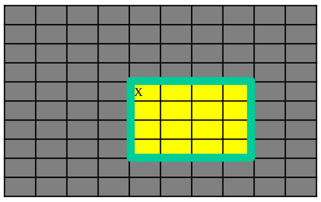
The Previous frame



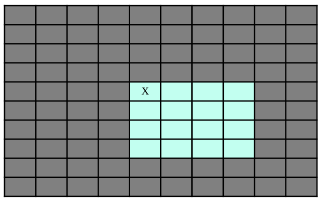
A target macro block at (4,4)
at the current frame

Fourth, match a candidate block at d=(-1,0)
Computer MSE and store

For Search Window Size p=1



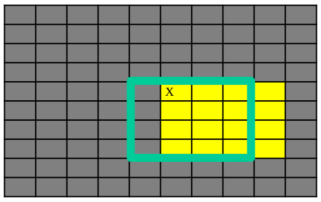
The Previous frame



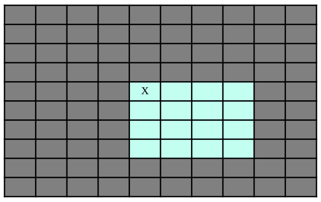
A target macro block at (4,4)
at the current frame

Firth, match a candidate block at d=(0,0)
Computer MSE and store

For Search Window Size p=1



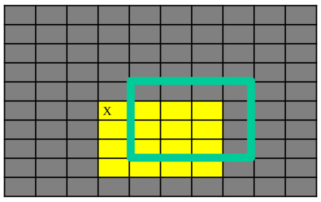
The Previous frame



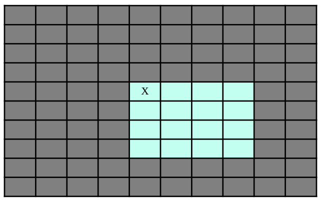
A target macro block at (4,4)
at the current frame

Sixth, match a candidate block at d=(1,0)
Computer MSE and store

For Search Window Size p=1



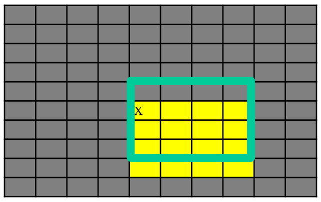
The Previous frame



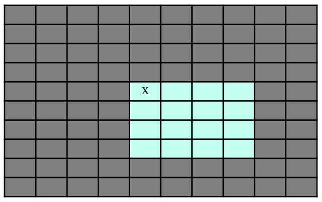
A target macro block at (4,4)
at the current frame

Seventh, match a candidate block at d=(1,-1)
Computer MSE and store

For Search Window Size p=1



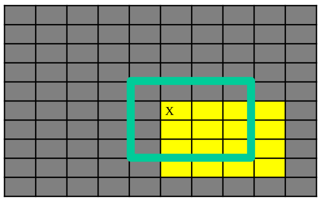
The Previous frame



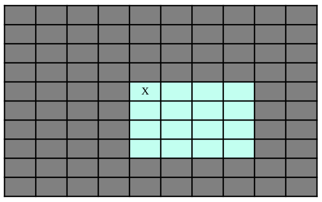
A target macro block at (4,4)
at the current frame

Eighth, match a candidate block at d=(0,1)
Computer MSE and store

For Search Window Size p=1



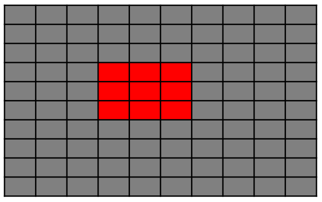
The Previous frame

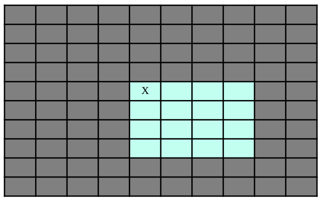


A target macro block at (4,4)
at the current frame

Last, match a candidate block at d=(1,1)
Computer MSE and store

For Search Window Size p=1

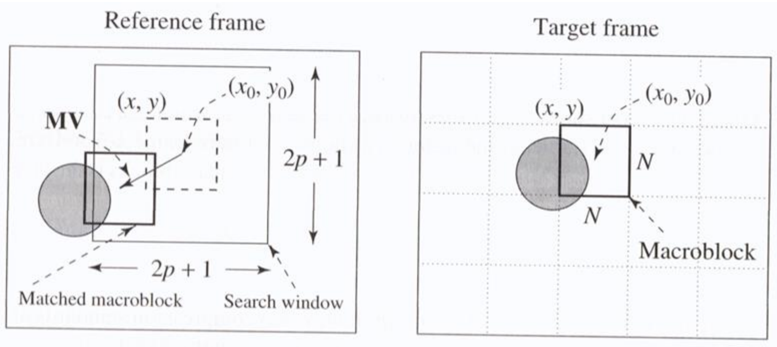

The Previous frame



A target macro block at (4,4) at the current frame

Search window size in red.
Find the best matching block using MSE.
Let's assume that the best matching candidate block is at (3,4) then, the motion vector for the target block is d=(-1, 0).

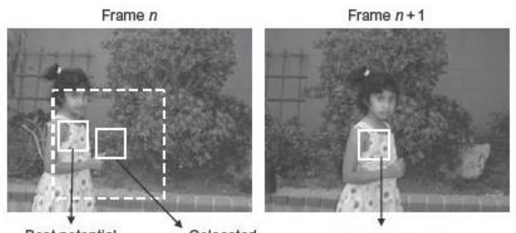
Search for Motion Vectors

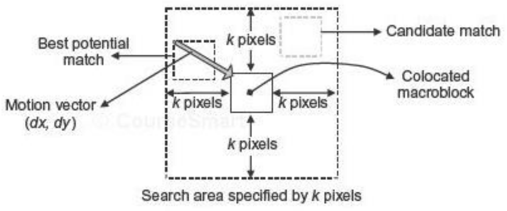


Macroblocks and motion vector in video compression: (a) reference frame; (b) target frame.

Search for Motion Vectors

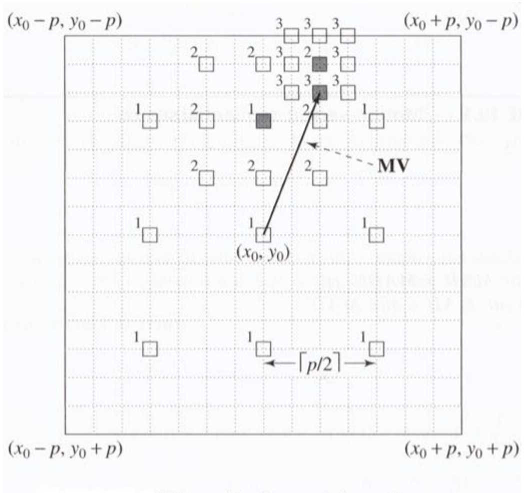
- Sequential Search (a.k.a. full search) – Sequentially search the whole (2k+1)x(2k+1) search window area
- Very costly.
- Not good for real-time encoding.





Search for Motion Vectors

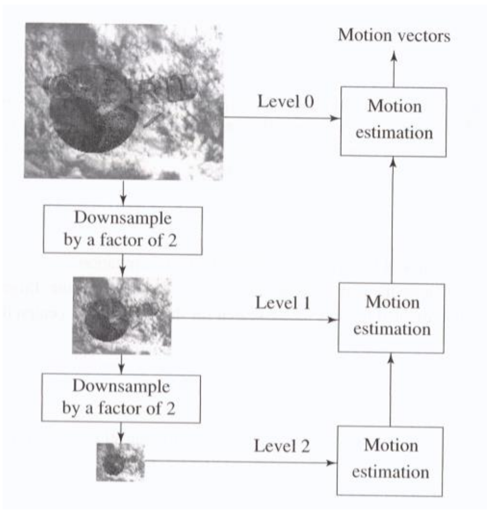
- 2D Logarithmic Search – Acts like binary search. More efficient than sequential search



2D Logarithmic search for motion vectors.

Search for Motion Vectors

- Hierarchical Search. The initial motion vector is obtained from images with a significantly reduced resolution and refined.



Matching Criteria

- Various Criteria are used to decided whether the current block matches a target block –
  - Mean Absolute Difference (MAD)
  - Mean Square Difference (MSD)
  - Pel Difference Classification (PDC)
  - Integral Projection

Matching Criteria

- Mean Absolute Difference (MAD)
  - The mean absolute difference (MAD) is the most popular block matching criterion. Corresponding pixels from each block are compared and their differences summed, as described by this equation.

$$\frac{1}{mn}\sum_{p=1}^{m}\sum_{q=1}^{n}\left|A[p,q]-B[p,q]\right|$$

  - Mean Absolute Difference function for two blocks A and B of size *n* x *m*. A[p,q] is the value of the pixel in the p^{th} row and q^{th} column of block A.
  - The lower the MAD the better the match and so the candidate block with the minimum MAD should be chosen. The function is alternatively called Mean Absolute Error (MAE).

Matching Criteria

- Mean Square Difference (MSD)
  - The mean square difference function is similar to the mean absolute difference function, except that the difference between pixels is squared before summation.

$$\frac{1}{mn}\sum_{p=1}^{m}\sum_{q=1}^{n}\left(A[p,q]-B[p,q]\right)^2$$

Mean Squared Difference function

  - The mean square difference is more commonly called the mean square error (MSE) and the lower this value the better the match. This criterion is said to result in better matches than the MAD criterion

Matching Criteria

- Pel Difference Classification (PDC)
  - The Pel Difference Classification (PDC) function compares each pixel of the target block with its counterpart in the candidate block and classifies each pixel pair as either matching or not matching. Pixels are matching if the difference between their values is less than some threshold and the greater the number of matching pixels the better the match.

$$\sum_{q=1}^{n}\sum_{p=1}^{m}\left[ord\left(\left|A[p,q]-B[p,q]\right|\le t\right)\right]$$

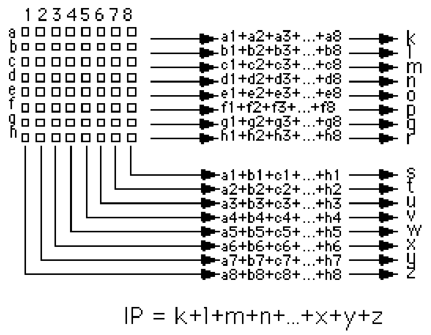
  - Pel Difference Classification function. Ord(*e*) evaluates to 1 if *e* is true and 0 otherwise.

Matching Criteria

- Integral Projection
  - Integral projections are calculated by summing the values of pixels from each column and each row of a block. The most attractive feature of this criterion is that values calculated for a particular candidate block can be reused in calculating the integrals for overlapping candidate blocks. This feature is of particular value during an exhaustive search, but less useful in the case of sub-optimal searches. The inventors claims that this choice of matching criterion is less susceptible to noise than others.

Matching Criteria

- Integral Projection



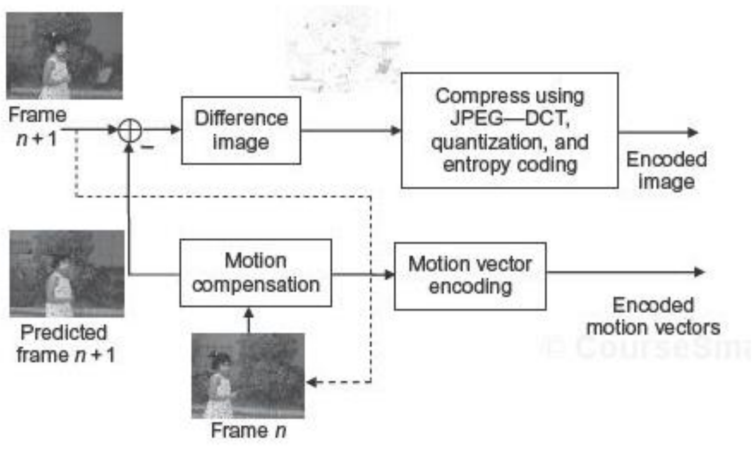
Integral Projection function

$$\sum_{p=1}^{m}\left|\sum_{q=1}^{n}A[p,q] \quad -\sum_{q=1}^{n}B[p,q]\right| \quad + \quad \sum_{q=1}^{n}\left|\sum_{p=1}^{m}A[p,q] \quad -\sum_{p=1}^{m}B[p,q]\right|$$

Integral Projection function

Motion Compensation - Algorithm

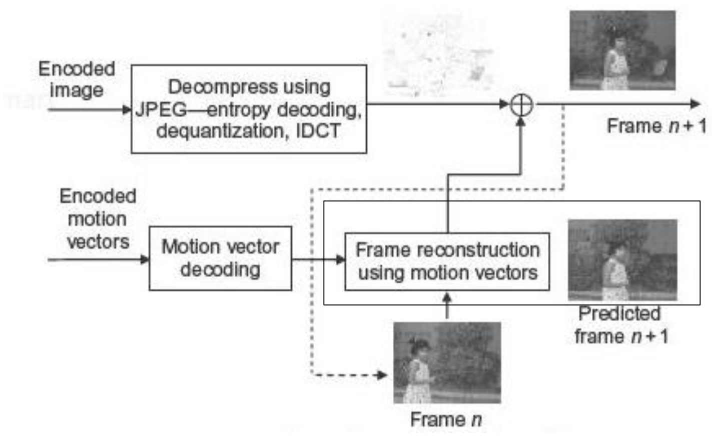
- Given a sequence of frames (each have macroblocks) –
- Encode first frame as IntraFrame
- For each corresponding macroblock of next frame and current frame, find the difference.
  – If difference less than threshold => no motion, find residual error
  – If difference above threshold => may be motion, look in search range to find a matching block using matching criteria discussed above. Note motion vector and residual error
  – If difference (or total residual error) is *too large* for a majority of macroblocks, and/or after regular intervals encode the current frame as an Intraframe and proceed to previous step



Motion Compensation - Encoding

- There are two things to encode here:
  - Motion Vector for every macroblock
  - Difference or residual for every macroblock
- Motion vectors are typically encoded losslessly (similar to JPEG lossless mode)
- The *residuals e(x, y)* are encoded lossy +lossless (DCT+Entropy) producing *variable bit rate (VBR).*
- If smooth motion or no motion:
  - Motion prediction is good (residuals are small)
  - Entropy coded with few bits
- If complex motion or change of scene:
  - Motion prediction is bad (residual are large)
  - Entropy coded with many bits

Motion Compensation - Decoding


Encoded image
Decompress using JPEG—entropy decoding, dequantization, IDCT
Frame n + 1
Encoded motion vectors
Motion vector decoding
Frame reconstruction using motion vectors
Predicted frame n + 1
Frame n