10 Basic Video Compression Techniques

As discussed in Chap. 7, the volume of uncompressed video data could be extremely large. Even a modest CIF video with a picture resolution of only 352 $\times$ 288, if uncompressed, would carry more than 35 Mbps. In HDTV, the bitrate could easily exceed 1 Gbps. This poses challenges and problems for storage and network communications.

This chapter introduces some basic video compression techniques and illustrates them in standards H.261 and H.263—two video compression standards aimed mostly at videoconferencing. The next two chapters further introduce several MPEG video compression standards and the latest, H.264 and H.265.

Tekalp [1] and Poynton [2] set out the fundamentals of digital video processing. They provide a good overview of the mathematical foundations of the problems to be addressed in video. The books by Bhaskaran and Konstantinides [3] and Wang et al. [4] include good descriptions of early video compression algorithms.

10.1 Introduction to Video Compression

A video consists of a time-ordered sequence of frames—images. An obvious solution to video compression would be predictive coding based on previous frames. For example, suppose we simply created a predictor such that the prediction equals the previous frame. Then compression proceeds by subtracting images: instead of subtracting the image from itself (i.e., use a spatial-domain derivative), we subtract in time order and code the residual error.

And this works. Suppose most of the video is unchanging in time. Then we get a nice histogram peaked sharply at zero—a great reduction in terms of the entropy of the original video, just what we wish for.

However, it turns out that at acceptable cost, we can do even better by searching for just the right parts of the image to subtract from the previous frame. After all, our naive subtraction scheme will likely work well for a background of office furniture and sedentary university types, but wouldn't a football game have players zooming around the frame, producing large values when subtracted from the previously static green playing field?

So in the next section we examine how to do better. The idea of looking for the football player in the next frame is called *motion estimation*, and the concept of shifting pieces of the frame around so as to best subtract away the player is called *motion compensation*.

10.2 Video Compression Based on Motion Compensation

The image compression techniques discussed in the previous chapters (e.g., JPEG and JPEG2000) exploit *spatial redundancy*, the phenomenon that picture contents often change relatively slowly across images, making a large suppression of higher spatial frequency components viable.

A video can be viewed as a sequence of images stacked in the *temporal* dimension. Since the frame rate of the video is often relatively high (e.g., $\geq$15 frames per second) and the camera parameters (focal length, position, viewing angle, etc.) usually do not change rapidly between frames, the contents of consecutive frames are usually similar, unless certain objects in the scene move extremely fast or the scene changes. In other words, the video has *temporal redundancy*.

Temporal redundancy is often significant and it is indeed exploited, so that not every frame of the video needs to be coded independently as a new image. Instead, the difference between the current frame and other frame(s) in the sequence is coded. If redundancy between them is great enough, the difference images could consist mainly of small values and low entropy, which is good for compression.

All modern digital video compression algorithms (including H.264 and H.265) adopt this *Hybrid coding* approach, i.e., predicting and compensating for the differences between video frames to remove the temporal redundancy, and then transform coding on the residual signal (the differences) to reduce the spatial redundancy.

As we mentioned, although a simplistic way of deriving the difference image is to subtract one image from the other (pixel by pixel), such an approach is ineffective in yielding a high compression ratio. Since the main cause of the difference between frames is camera and/or object motion, these motion generators can be "compensated" by detecting the displacement of corresponding pixels or regions in these frames and measuring their differences. Video compression algorithms that adopt this approach are said to be based on motion compensation (MC). The three main steps of these algorithms are:

1. Motion estimation (motion vector search)
2. Motion compensation-based prediction
3. Derivation of the prediction error—the difference.

For efficiency, each image is divided into *macroblocks* of size $N \times N$. By default, $N = 16$ for luminance images. For chrominance images, $N = 8$ if 4:2:0 chroma subsampling is adopted. Motion compensation is not performed at the pixel level,

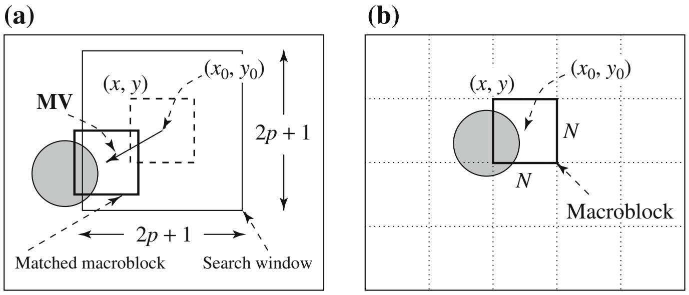


Fig. 10.1 Macroblocks and motion vector in video compression: **a** reference frame; **b** target frame

nor at the level of *video object*, as in later video standards (such as MPEG-4). Instead, it is at the macroblock level.

The current image frame is referred to as the *Target frame*. A match is sought between the macroblock under consideration in the Target frame and the most similar macroblock in previous and/or future frame(s) [referred to as *Reference frame(s)*]. In that sense, the Target macroblock is predicted from the Reference macroblock(s).

The displacement of the reference macroblock to the target macroblock is called a *motion vector* **MV**. Figure 10.1 shows the case of *forward prediction*, in which the Reference frame is taken to be a previous frame. If the Reference frame is a future frame, it is referred to as *backward prediction*. The *difference* of the two corresponding macroblocks is the prediction error.

For video compression based on motion compensation, after the first frame, only the motion vectors and difference macroblocks need be coded, since they are sufficient for the decoder to regenerate all macroblocks in subsequent frames.

We will return to the discussion of some common video compression standards after the following section, in which we discuss search algorithms for motion vectors.

10.3 Search for Motion Vectors

The search for motion vectors $\mathbf{MV}(u, v)$ as defined above is a matching problem, also called a *correspondence* problem [5]. Since MV search is computationally expensive, it is usually limited to a small immediate neighborhood. Horizontal and vertical displacements i and j are in the range $[-p, p]$, where p is a positive integer with a relatively small value. This makes a search window of size $(2p + 1) \times (2p + 1)$, as Fig. 10.1 shows. The center of the macroblock (x_0, y_0) can be placed at each of the grid positions in the window.

For convenience, we use the upper left corner (x, y) as the origin of the macroblock in the Target frame. Let $C(x + k, y + l)$ be pixels in the macroblock in the Target (current) frame and $R(x + i + k, y + j + l)$ be pixels in the macroblock in the

Reference frame, where k and l are indices for pixels in the macroblock and i and j are the horizontal and vertical displacements, respectively. The difference between the two macroblocks can then be measured by their *Mean Absolute Difference (MAD)*, defined as

$$MAD(i, j) = \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} |C(x+k, y+l) - R(x+i+k, y+j+l)|, \quad (10.1)$$

where N is the size of the macroblock.

The goal of the search is to find a vector (i, j) as the motion vector $\mathbf{MV} = (u, v)$, such that $MAD(i, j)$ is minimum:

$$(u, v) = [(i, j) \mid MAD(i, j) \quad \text{is minimum}, \quad i \in (-p, p), j \in (-p, p)] \quad (10.2)$$

We used the mean absolute difference in the above discussion. However, this measure is by no means the only possible choice. In fact, some encoders (e.g., H.263) will simply use the *Sum of Absolute Difference* (*SAD*). Some other common error measures, such as the *Mean Square Error* (*MSE*), would also be appropriate.

10.3.1 Sequential Search

The simplest method for finding motion vectors is to sequentially search the whole $(2p+1) \times (2p+1)$ window in the Reference frame (also referred to as *full search*). A macroblock centered at each of the positions within the window is compared to the macroblock in the Target frame, pixel by pixel, and their respective MAD is then derived using Eq. (10.1). The vector (i, j) that offers the least MAD is designated the $\mathbf{MV}(u, v)$ for the macroblock in the Target frame.

Procedure 10.1 (Motion vector: Sequential search)

```
BEGIN
   $min\_MAD = LARGE\_NUMBER$;      /* Initialization */
   for $i = -p$ to $p$
       for $j = -p$ to $p$
           {
               $cur\_MAD = MAD(i, j)$;
               if $cur\_MAD < min\_MAD$
                   {
                       $min\_MAD = cur\_MAD$;
                       $u = i$;          /* Get the coordinates for MV. */
                       $v = j$;
                   }
           }
END
```

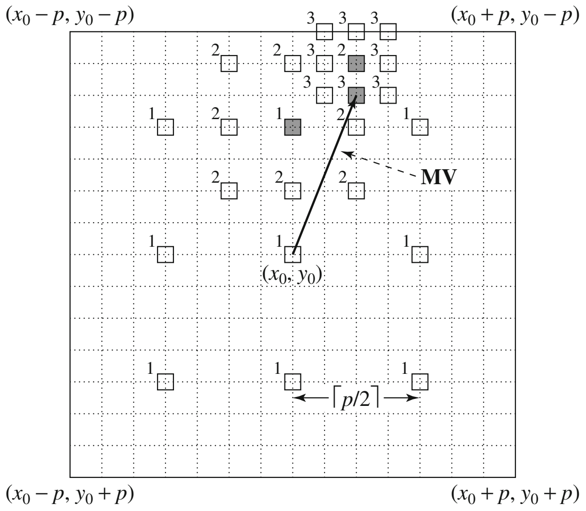


Fig. 10.2 2D Logarithmic search for motion vectors

Clearly, the sequential search method is very costly. From Eq. (10.1), each pixel comparison requires three operations (subtraction, absolute value, addition). Thus the cost for obtaining a motion vector for a single macroblock is $(2p + 1) \cdot (2p + 1) \cdot N^2 \cdot 3 \Rightarrow O(p^2N^2)$.

As an example, let's assume the video has a resolution of 720×480 and a frame rate of 30 fps; also, assume $p = 15$ and $N = 16$. The number of operations needed for each motion vector search is thus

$$(2p + 1)^2 \cdot N^2 \cdot 3 = 31^2 \times 16^2 \times 3.$$

Considering that a single image frame has $\frac{720 \times 480}{N \cdot N}$ macroblocks, and 30 frames each second, the total operations needed per second is

$$\begin{aligned} \text{OPS\_per\_second} &= (2p + 1)^2 \cdot N^2 \cdot 3 \cdot \frac{720 \times 480}{N \cdot N} \cdot 30 \\ &= 31^2 \times 16^2 \times 3 \times \frac{720 \times 480}{16 \times 16} \times 30 \approx 29.89 \times 10^9. \end{aligned}$$

This would certainly make real-time encoding of this video difficult.

10.3.2 2D Logarithmic Search

A more efficient version, suboptimal but still usually effective, is called *Logarithmic Search*. The procedure for a 2D Logarithmic Search of motion vectors takes several iterations and is akin to a binary search. As Fig. 10.2 illustrates, only nine locations in the search window, marked "1," are initially used as seeds for a MAD-based search.

After the one that yields the minimum MAD is located, the center of the new search region is moved to it, and the stepsize (*offset*) is reduced to half. In the next iteration, the nine new locations are marked "2," and so on.[1] For the macroblock centered at (x_0, y_0) in the Target frame, the procedure is as follows:

Procedure 10.2 (Motion vector: 2D-Logarithmic search)

```
BEGIN
    offset = $\lceil \frac{p}{2} \rceil$;
    Specify nine macroblocks within the search window in the Reference frame,
    they are centered at $(x_0, y_0)$ and separated by offset horizontally and/or vertically;
    WHILE last $\neq$ TRUE
        {
            Find one of the nine specified macroblocks that yields the minimum $MAD$;
            if offset = 1 then last = TRUE;
            offset = $\lceil \text{offset}/2 \rceil$;
            Form a search region with the new offset and new center found;
        }
END
```

Instead of sequentially comparing with $(2p+1)^2$ macroblocks from the Reference frame, the 2-D Logarithmic Search will compare with only $9 \cdot (\lceil \log_2 p \rceil + 1)$ macroblocks. In fact, it would be $8 \cdot (\lceil \log_2 p \rceil + 1) + 1$, since the comparison that yielded the least MAD from the last iteration can be reused. Therefore, the complexity is dropped to $O(\log p \cdot N^2)$. Since p is usually of the same order of magnitude as N, the saving is substantial compared to $O(p^2 N^2)$.

Using the same example as in the previous subsection, the total operations per second drop to

$$\begin{aligned} \text{OPS\_per\_second} &= \big(8 \cdot (\lceil \log_2 p \rceil + 1) + 1\big) \cdot N^2 \cdot 3 \cdot \frac{720 \times 480}{N \cdot N} \cdot 30 \\ &= \big(8 \cdot \lceil \log_2 15 \rceil + 9\big) \times 16^2 \times 3 \times \frac{720 \times 480}{16 \times 16} \times 30 \\ &\approx 1.25 \times 10^9. \end{aligned}$$

10.3.3 Hierarchical Search

The search for motion vectors can benefit from a hierarchical (multiresolution) approach in which initial estimation of the motion vector can be obtained from images

[1] The procedure is heuristic. It assumes a general continuity (monotonicity) of image contents—that they do not change randomly within the search window. Otherwise, the procedure might not find the best match.

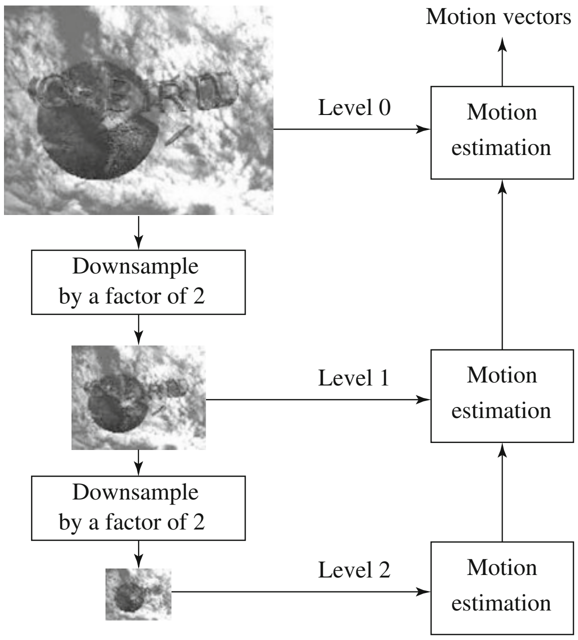


Fig. 10.3 A three-level hierarchical search for motion vectors

with a significantly reduced resolution. Figure 10.3 depicts a three-level hierarchical search in which the original image is at level zreo, images at levels one and two are obtained by downsampling from the previous levels by a factor of two, and the initial search is conducted at level two. Since the size of the macroblock is smaller and p can also be proportionally reduced at this level, the number of operations required is greatly reduced (by a factor of 16 at this level).

The initial estimation of the motion vector is coarse because of the lack of image detail and resolution. It is then refined level by level toward level zero. Given the estimated motion vector (u^k, v^k) at level k, a 3×3 neighborhood centered at $(2 \cdot u^k, 2 \cdot v^k)$ at level $k - 1$ is searched for the refined motion vector. In other words, the refinement is such that at level $k - 1$, the motion vector (u^{k-1}, v^{k-1}) satisfies

$$(2u^k - 1 \le u^{k-1} \le 2u^k + 1, \quad 2v^k - 1 \le v^{k-1} \le 2v^k + 1),$$

and yields minimum MAD for the macroblock under examination.

Let (x_0^k, y_0^k) denote the center of the macroblock at level k in the Target frame. The procedure for hierarchical motion vector search for the macroblock centered at (x_0^0, y_0^0) in the Target frame can be outlined as follows:

Procedure 10.3 (Motion vector: Hierarchical search)

```
BEGIN
  // Get macroblock center position at the lowest resolution level $k$, e.g., level 2.
  $x_0^k = x_0^0/2^k; \quad y_0^k = y_0^0/2^k;$
  Use Sequential (or 2D Logarithmic) search method to get initial estimated
  $\mathbf{MV}(u^k, v^k)$ at level $k$;
  WHILE last $\neq$ TRUE
     {
        Find one of the nine macroblocks that yields minimum $MAD$
         at level $k-1$ centered at
         $(2(x_0^k+u^k)-1 \le x \le 2(x_0^k+u^k)+1,\ 2(y_0^k+v^k)-1 \le y \le 2(y_0^k+v^k)+1)$;
         if $k = 1$ then last = TRUE;
         $k = k - 1$;
         Assign $(x_0^k, y_0^k)$ and $(u^k, v^k)$ with the new center location and motion
         vector;
     }
END
```

We will use the same example as in the previous sections to estimate the total operations needed each second for a three-level hierarchical search. For simplicity, the overhead for initially generating multiresolution target and reference frames will not be included, and it will be assumed that Sequential search is used at each level.

The total number of macroblocks processed each second is still $\frac{720\times480}{N\cdot N} \times 30$. However, the operations needed for each macroblock are reduced to

$$\left[\left(2\left\lceil\frac{p}{4}\right\rceil+1\right)^2\left(\frac{N}{4}\right)^2+9\left(\frac{N}{2}\right)^2+9N^2\right]\times 3.$$

Hence,

$$\begin{aligned}\text{OPS\_per\_second} &= \left[\left(2\left\lceil\frac{p}{4}\right\rceil+1\right)^2\left(\frac{N}{4}\right)^2+9\left(\frac{N}{2}\right)^2+9N^2\right]\\ &\quad \times 3 \times \frac{720\times480}{N\cdot N}\times 30\\ &= \left[\left(\frac{9}{4}\right)^2+\frac{9}{4}+9\right]\times 16^2\times 3\times\frac{720\times480}{16\times16}\times 30\\ &\approx 0.51\times10^9.\end{aligned}$$

Table 10.1 summarizes the comparison of the three motion vector search methods for a 720×480, 30 fps video when $p = 15$ and 7, respectively.

Table 10.1 Comparison of computational cost of motion vector search methods according to the examples

Search method	OPS_per_second for 720 × 480 at 30 fps	
	$p = 15$	$p = 7$
Sequential search	29.89×10^9	7.00×10^9
2D logarithmic search	1.25×10^9	0.78×10^9
Three-level hierarchical search	0.51×10^9	0.40×10^9

10.4 H.261

H.261 is an earlier digital video compression standard. Because its principle of motion compensation–based compression is very much retained in all later video compression standards, we will start with a detailed discussion of H.261.

The International Telegraph and Telephone Consultative Committee (CCITT) initiated the development of H.261 in 1988. The final recommendation was adopted by the ITU (International Telecommunication Union)—Telecommunication standardization sector (ITU-T), formerly CCITT, in 1990 [6].

The standard was designed for videophone, videoconferencing, and other audiovisual services over ISDN telephone lines (see Chap. 15.2.3). Initially, it was intended to support multiples (from 1 to 5) of 384 kbps (kilobits per second) channels. In the end, however, the video codec supports bitrates of $p \times 64$ kbps, where p ranges from 1 to 30. Hence the standard was once known as $p * 64$, pronounced "p star 64". The standard requires video encoders delay to be less than 150 ms, so that the video can be used for real-time, bidirectional video conferencing.

H.261 belongs to the following set of ITU recommendations for visual telephony systems:

- **H.221.** Frame structure for an audiovisual channel supporting 64 to 1,920 kbps
- **H.230.** Frame control signals for audiovisual systems
- **H.242.** Audiovisual communication protocols
- **H.261.** Video encoder/decoder for audiovisual services at $p \times 64$ kbps
- **H.320.** Narrowband audiovisual terminal equipment for $p \times 64$ kbps transmission.

Table 10.2 lists the video formats supported by H.261. Chroma subsampling in H.261 is 4:2:0. Considering the relatively low bitrate in network communications at the time, support for CCIR 601 QCIF is specified as required, whereas support for CIF is optional.

Figure 10.4 illustrates a typical H.261 frame sequence. Two types of image frames are defined: intra-frames (*I-frames*) and inter-frames (*P-frames*).

I-frames are treated as independent images. Basically, a transform coding method similar to JPEG is applied within each I-frame, hence the name "intra".

P-frames are not independent. They are coded by a forward predictive coding method in which current macroblocks are predicted from similar macroblocks in

Table 10.2 Video formats supported by H.261

Video format	Luminance image resolution	Chrominance image resolution	Bitrate (Mbps) (if 30 fps and uncompressed)	H.261 support
QCIF	176 × 144	88 × 72	9.1	Required
CIF	352 × 288	176 × 144	36.5	Optional

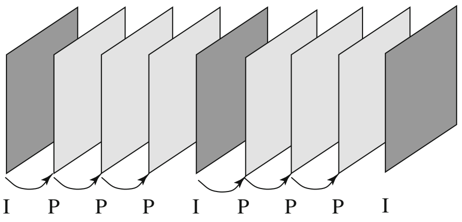


Fig. 10.4 H.261 Frame sequence

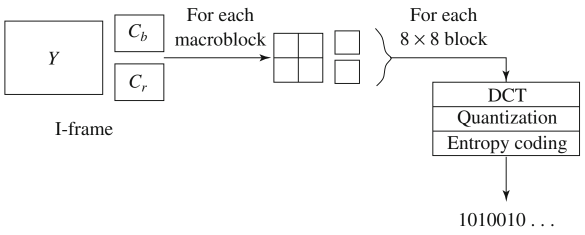


Fig. 10.5 I-frame coding

the preceding I- or P-frame, and *differences* between the macroblocks are coded. *Temporal redundancy removal* is hence included in P-frame coding, whereas I-frame coding performs only *spatial redundancy removal*. It is important to remember that prediction from a previous P-frame is allowed (not just from a previous I-frame).

The interval between pairs of I-frames is a variable and is determined by the encoder. Usually, an ordinary digital video has a couple of I-frames per second. Motion vectors in H.261 are always measured in units of full pixels and have a limited range of ±15 pixels—that is, $p = 15$.

10.4.1 Intra-Frame (I-Frame) Coding

Macroblocks are of size 16 × 16 pixels for the Y frame of the original image. For Cb and Cr frames, they correspond to areas of 8 × 8, since 4:2:0 chroma subsampling

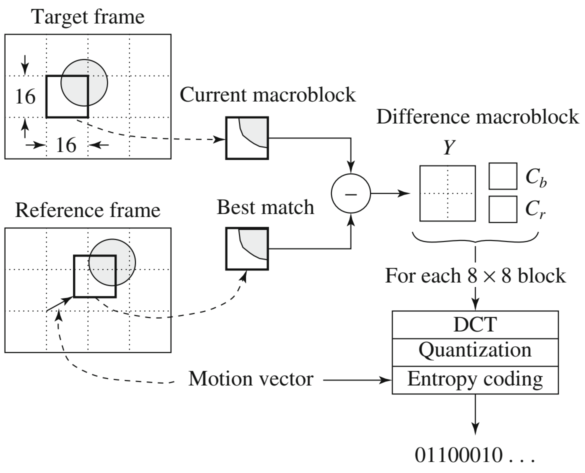


Fig. 10.6 H.261 P-frame coding based on motion compensation

is employed. Hence, a macroblock consists of four Y blocks, one Cb, and one Cr, 8 × 8 blocks.

For each 8 × 8 block, a DCT transform is applied. As in JPEG (discussed in detail in Chap. 9), the DCT coefficients go through a quantization stage. Afterwards, they are zigzag-scanned and eventually entropy-coded (as shown in Fig. 10.5).

10.4.2 Inter-Frame (P-Frame) Predictive Coding

Figure 10.6 shows the H.261 P-frame coding scheme based on motion compensation. For each macroblock in the Target frame, a motion vector is allocated by one of the search methods discussed earlier. After the prediction, a *difference macroblock* is derived to measure the *prediction error*. It is also carried in the form of four Y blocks, one Cb, and one Cr block. Each of these 8 × 8 blocks goes through DCT, quantization, zigzag scan, and entropy coding. The motion vector is also coded.

Sometimes, a good match cannot be found—the prediction error exceeds a certain acceptable level. The macroblock itself is then encoded (treated as an intra macroblock) and in this case is termed a *nonmotion compensated macroblock*.

P-frame coding encodes the difference macroblock (not the Target macroblock itself). Since the difference macroblock usually has a much smaller entropy than the Target macroblock, a large *compression ratio* is attainable.

In fact, even the motion vector is not directly coded. Instead, the difference, **MVD**, between the motion vectors of the preceding macroblock and current macroblock is sent for entropy coding:

$$\mathbf{MVD} = \mathbf{MV}_{\text{Preceding}} - \mathbf{MV}_{\text{Current}} \tag{10.3}$$

10.4.3 Quantization in H.261

The quantization in H.261 does not use 8×8 quantization matrices, as in JPEG and MPEG. Instead, it uses a constant, called *step_size*, for all DCT coefficients within a macroblock. According to the need (e.g., bitrate control of the video), *step_size* can take on any one of the 31 even values from 2 to 62. One exception, however, is made for the DC coefficient in intra mode, where a step_size of 8 is always used. If we use DCT and $QDCT$ to denote the DCT coefficients before and after quantization, then for DC coefficients in intra mode,

$$QDCT = \text{round}\left(\frac{DCT}{step\_size}\right) = \text{round}\left(\frac{DCT}{8}\right). \tag{10.4}$$

For all other coefficients:

$$QDCT = \left\lfloor \frac{DCT}{step\_size} \right\rfloor = \left\lfloor \frac{DCT}{2 \times scale} \right\rfloor, \tag{10.5}$$

where $scale$ is an integer in the range of [1, 31].

The midtread quantizer, discussed in Sect. 8.4.1 typically uses a `round` operator. Equation (10.4) uses this type of quantizer. However, Eq. (10.5) uses a `floor` operator and, as a result, leaves a center dead-zone (as Fig. 9.8 shows) in its quantization space, with a larger input range mapped to zero.

10.4.4 H.261 Encoder and Decoder

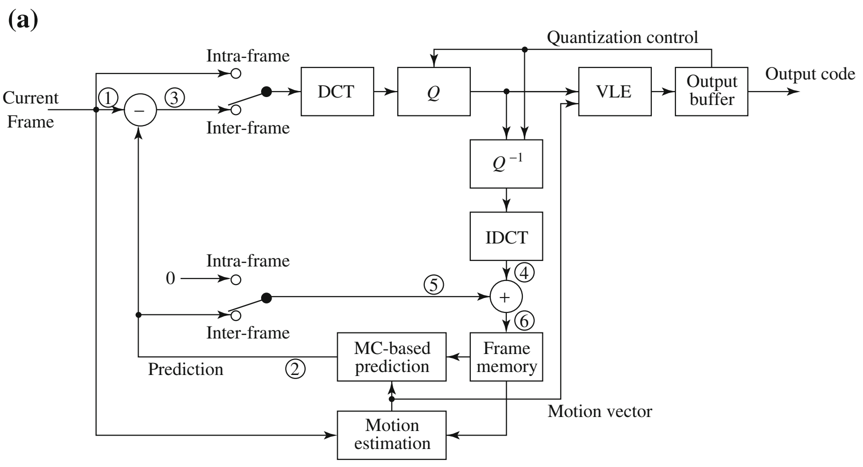
Figure 10.7 shows a relatively complete picture of how the H.261 encoder and decoder work. Here, Q and Q^{-1} stand for quantization and its inverse, respectively. Switching of the intra- and inter-frame modes can be readily implemented by a multiplexer. To avoid propagation of coding errors,

- An I-frame is usually sent a couple of times in each second of the video.
- As discussed earlier (see DPCM in Sect. 6.3.5), decoded frames (not the original frames) are used as reference frames in motion estimation.

To illustrate the operational detail of the encoder and decoder, let's use a scenario where frames I, P_1, and P_2 are encoded and then decoded. The data that goes through the observation points, indicated by the circled numbers in Fig. 10.7, is summarized in Tables 10.3 and 10.4. We will use I, P_1, P_2 for the original data, $\tilde{I}$, $\tilde{P}_1$, $\tilde{P}_2$ for the decoded data (usually a lossy version of the original), and P_1', P_2' for the predictions in the Inter-frame mode.

For the encoder, when the Current Frame is an Intra-frame, Point number 1 receives macroblocks from the I-frame, denoted I in Table 10.3. Each I undergoes DCT, Quantization, and Entropy Coding steps, and the result is sent to the Output Buffer, ready to be transmitted.

Meanwhile, the quantized DCT coefficients for I are also sent to Q^{-1} and IDCT and hence appear at Point 4 as $\tilde{I}$. Combined with a zero input from Point 5, the data at Point 6 remains as $\tilde{I}$ and this is stored in Frame Memory, waiting to be used for



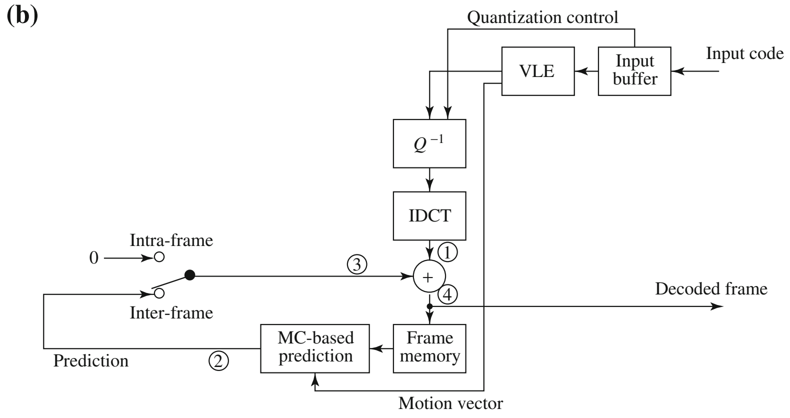


Fig. 10.7 H.261: **a** encoder; **b** decoder

Motion Estimation and Motion Compensation-based Prediction for the subsequent frame P_1.

Quantization Control serves as feedback—that is, when the Output Buffer is too full, the quantization step_size is increased, so as to reduce the size of the coded data. This is known as an *encoding rate control process*.

When the subsequent Current Frame P_1 arrives at Point 1, the Motion Estimation process is invoked to find the motion vector for the best matching macroblock in frame $\tilde{I}$ for each of the macroblocks in P_1. The estimated motion vector is sent to both Motion Compensation-based Prediction and Variable-Length Encoding (VLE). The MC-based Prediction yields the best matching macroblock in P_1. This is denoted as P_1' appearing at Point 2.

Table 10.3 Data flow at the observation points in H.261 encoder

Current frame	Observation point					
	1	2	3	4	5	6
I	I			$\tilde{I}$	0	$\tilde{I}$
P_1	P_1	P_1'	D_1	$\tilde{D}_1$	P_1'	$\tilde{P}_1$
P_2	P_2	P_2'	D_2	$\tilde{D}_2$	P_2'	$\tilde{P}_2$

Table 10.4 Data flow at the observation points in H.261 decoder

Current frame	Observation point			
	1	2	3	4
I	$\tilde{I}$		0	$\tilde{I}$
P_1	$\tilde{D}_1$	P_1'	P_1'	$\tilde{P}_1$
P_2	$\tilde{D}_2$	P_2'	P_2'	$\tilde{P}_2$

At Point 3, the "prediction error" is obtained, which is $D_1 = P_1 - P_1'$. Now D_1 undergoes DCT, Quantization, and Entropy Coding, and the result is sent to the Output Buffer. As before, the DCT coefficients for D_1 are also sent to Q^{-1} and IDCT and appear at Point 4 as $\tilde{D}_1$.

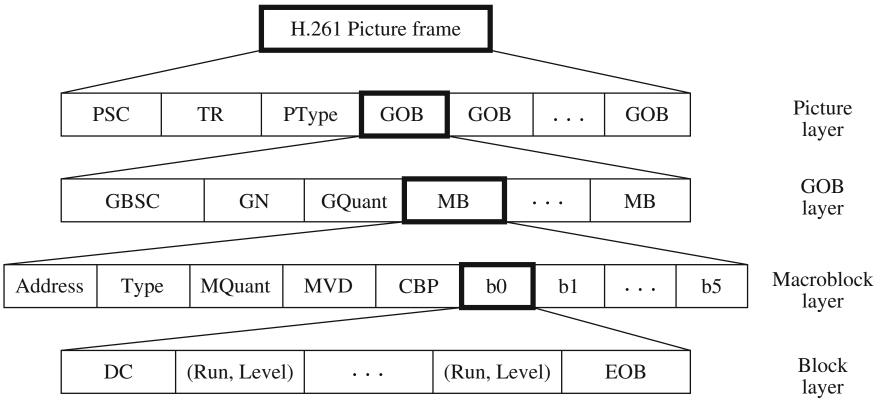
Added to P_1' at Point 5, we have $\tilde{P}_1 = P_1' + \tilde{D}_1$ at Point 6. This is stored in Frame Memory, waiting to be used for Motion Estimation and Motion Compensation-based Prediction for the subsequent frame P_2. The steps for encoding P_2 are similar to those for P_1, except that P_2 will be the Current Frame and P_1 becomes the Reference Frame.

For the decoder, the input code for frames will be decoded first by Entropy Decoding, Q^{-1}, and IDCT. For Intra-frame mode, the first decoded frame appears at Point 1 and then Point 4 as $\tilde{I}$. It is sent as the first output and at the same time stored in the Frame Memory.

Subsequently, the input code for Inter-frame P_1 is decoded, and prediction error $\tilde{D}_1$ is received at Point 1. Since the motion vector for the current macroblock is also entropy-decoded and sent to Motion Compensation-based Prediction, the corresponding predicted macroblock P_1' can be located in frame $\tilde{I}$ and will appear at Points 2 and 3. Combined with $\tilde{D}_1$, we have $\tilde{P}_1 = P_1' + \tilde{D}_1$ at Point 4, and it is sent out as the decoded frame and also stored in the Frame Memory. Again, the steps for decoding P_2 are similar to those for P_1.

10.4.5 A Glance at the H.261 Video Bitstream Syntax

Let's take a brief look at the H.261 video bitstream syntax (see Fig. 10.8). This consists of a hierarchy of four layers: *Picture*, *Group of Blocks* (*GOB*), *Macroblock*, and *Block*.



PSC	Picture Start Code	TR	Temporal Reference
PType	Picture Type	GOB	Group of Blocks
GBSC	GOB Start Code	GN	Group Number
GQuant	GOB Quantizer	MB	Macroblock
MQuant	MB Quantizer	MVD	Motion Vector Data
CBP	Coded Block Pattern	EOB	End of Block

Fig. 10.8 Syntax of H.261 video bitstream

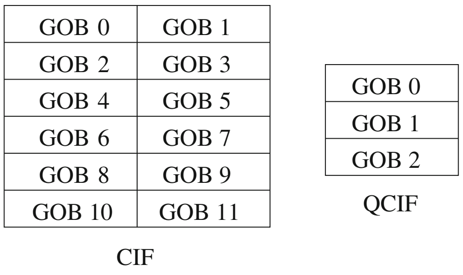


Fig. 10.9 Arrangement of GOBs in H.261 *luminance* images

1. **Picture layer**. *Picture Start Code* (*PSC*) delineates boundaries between pictures. *Temporal Reference* (*TR*) provides a timestamp for the picture. Since temporal subsampling can sometimes be invoked such that some pictures will not be transmitted, it is important to have TR, to maintain synchronization with audio. *Picture Type* (*PType*) specifies, for example, whether it is a CIF or QCIF picture.
2. **GOB layer.** H.261 pictures are divided into regions of 11×3 macroblocks (i.e., regions of 176×48 pixels in luminance images), each of which is called a *Group of Blocks* (*GOB*). Figure 10.9 depicts the arrangement of GOBs in a CIF or QCIF luminance image. For instance, the CIF image has 2×6 GOBs, corresponding to its image resolution of 352×288 pixels.

Each GOB has its *Start Code* (*GBSC*) and *Group number* (*GN*). The GBSC is unique and can be identified without decoding the entire variable-length code in the bitstream. In case a network error causes a bit error or the loss of some bits, H.261 video can be recovered and resynchronized at the next identifiable GOB, preventing the possible propagation of errors.

GQuant indicates the quantizer to be used in the GOB, unless it is overridden by any subsequent *Macroblock Quantizer* (*MQuant*). GQuant and MQuant are referred to as $scale$ in Eq. (10.5).

3. **Macroblock layer.** Each *macroblock* (*MB*) has its own *Address*, indicating its position within the GOB, quantizer (MQuant), and six 8×8 image blocks (4 Y, 1 Cb, 1 Cr). *Type* denotes whether it is an Intra- or Inter, motion compensated or nonmotion compensated macroblock. *Motion Vector Data* (*MVD*) is obtained by taking the difference between the motion vectors of the preceding and current macroblocks. Moreover, since some blocks in the macroblocks match well and some match poorly in Motion Estimation, a bitmask *Coded Block Pattern* (*CBP*) is used to indicate this information. Only well-matched blocks will have their coefficients transmitted.
4. **Block layer**. For each 8×8 block, the bitstream starts with *DC value*, followed by pairs of length of zero-run (*Run*) and the subsequent nonzero value (*Level*) for ACs, and finally the *End of Block* (*EOB*) code. The range of "Run" is [0, 63]. "Level" reflects quantized values—its range is $[-127, 127]$, and Level $\neq 0$.

10.5 H.263

H.263 is an improved video coding standard [7] for videoconferencing and other audiovisual services transmitted on Public Switched Telephone Networks (PSTN). It aims at low bitrate communications at bitrates of less than 64 kbps. It was adopted by the ITU-T Study Group 15 in 1995. Similar to H.261, it uses predictive coding for inter-frames, to reduce temporal redundancy, and transform coding for the remaining signal, to reduce spatial redundancy (for both intra-frames and difference macroblocks from inter-frame prediction) [7].

In addition to CIF and QCIF, H.263 supports sub-QCIF, 4CIF, and 16CIF. Table 10.5 summarizes video formats supported by H.263. If not compressed and assuming 30 fps, the bitrate for high-resolution videos (e.g., 16CIF) could be very high ($>$500 Mbps). For compressed video, the standard defines maximum bitrate per picture (BPPmaxKb), measured in units of 1,024 bits. In practice, a lower bit rate for compressed H.263 video can be achieved.

As in H.261, the H.263 standard also supports the notion of group of blocks. The difference is that GOBs in H.263 do not have a fixed size, and they always start and end at the left and right borders of the picture. As Fig. 10.10 shows, each QCIF luminance image consists of 9 GOBs and each GOB has 11×1 MBs (176×16 pixels), whereas each 4CIF luminance image consists of 18 GOBs and each GOB has 44×2 MBs (704×32 pixels).

Table 10.5 Video formats supported by H.263

Video format	Luminance image resolution	Chrominance image resolution	Bitrate (Mbps) (if 30 fps and uncompressed)	Bitrate (kbps) BPPmaxKb (compressed)
Sub-QCIF	128 × 96	64 × 48	4.4	64
QCIF	176 × 144	88 × 72	9.1	64
CIF	352 × 288	176 × 144	36.5	256
4CIF	704 × 576	352 × 288	146.0	512
16CIF	1408 × 1152	704 × 576	583.9	1024

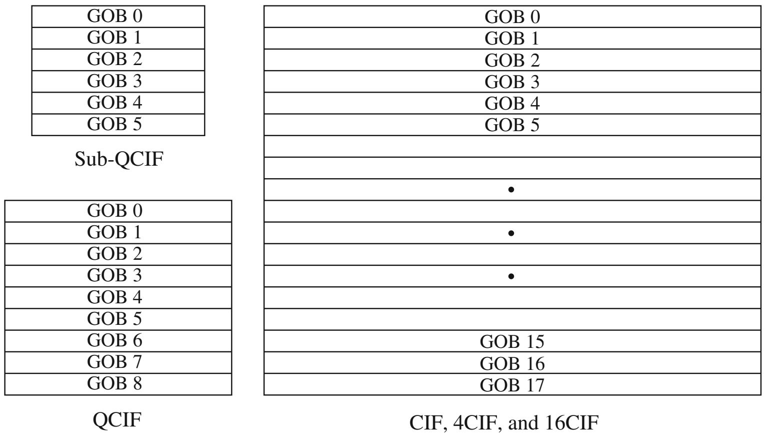


Fig. 10.10 Arrangement of GOBs in H.263 luminance images

10.5.1 Motion Compensation in H.263

The process of motion compensation in H.263 is similar to that of H.261. The motion vector (**MV**) is, however, not simply derived from the current macroblock. The horizontal and vertical components of the **MV** are predicted from the median values of the horizontal and vertical components, respectively, of **MV1**, **MV2**, **MV3** from the "previous", "above" and "above and right" macroblocks [see Fig. 10.11a]. Namely, for the macroblock with **MV**(u, v),

$$\begin{aligned} u_p &= median(u_1, u_2, u_3), \\ v_p &= median(v_1, v_2, v_3). \end{aligned} \tag{10.6}$$

Instead of coding the **MV**(u, v) itself, the error vector $(\delta u, \delta v)$ is coded, where $\delta u = u - u_p$ and $\delta v = v - v_p$. As shown in Fig. 10.11b, when the current MB is at the border of the picture or GOB, either (0, 0) or **MV1** is used as the motion vector for the out of-bound MB(s).

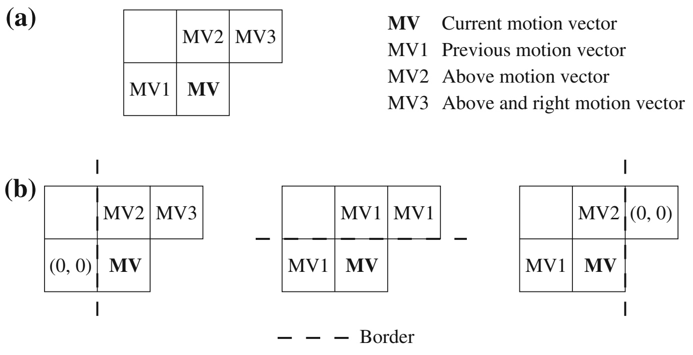


Fig. 10.11 Prediction of motion vector in H.263: **a** predicted **MV**of the current macroblock is the median of (**MV1**, **MV2**, **MV3**); **b** special treatment of MVs when the current macroblock is at border of picture or GOB

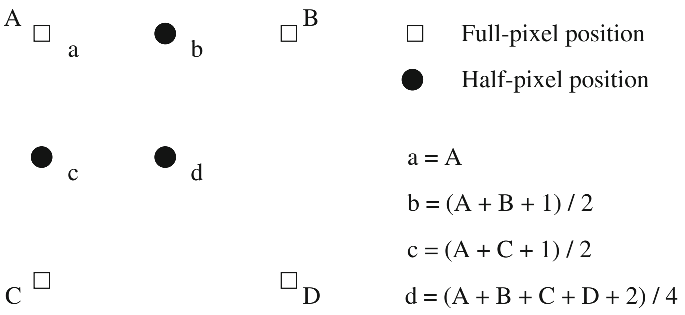


Fig. 10.12 Half-pixel prediction by bilinear interpolation in H.263

To improve the quality of motion compensation—that is, to reduce the prediction error—H.263 supports *half-pixel precision* as opposed to full-pixel precision only in H.261. The default range for both the horizontal and vertical components u and v of $\mathbf{MV}(u, v)$ is now $[-16, 15.5]$.

The pixel values needed at half-pixel positions are generated by a simple *bilinear interpolation* method, as shown in Fig. 10.12, where A, B, C, D and a, b, c, d are pixel values at full-pixel positions and half-pixel positions respectively, and " / " indicates division by truncation (also known as integer division).

10.5.2 Optional H.263 Coding Modes

Besides its core coding algorithm, H.263 specifies many negotiable coding options in its various Annexes. Four of the common options are as follows:

- **Unrestricted motion vector mode.** The pixels referenced are no longer restricted to within the boundary of the image. When the motion vector points outside the image boundary, the value of the boundary pixel geometrically closest to the referenced pixel is used. This is beneficial when image content is moving across the edge of the image, often caused by object and/or camera movements. This mode also allows an extension of the range of motion vectors. The maximum range of motion vectors is $[-31.5, 31.5]$, which enables efficient coding of fast-moving objects in videos.

- **Syntax-based arithmetic coding mode.** Like H.261, H.263 uses Variable-Length Coding (VLC) as a default coding method for the DCT coefficients. Variable-length coding implies that each symbol must be coded into a fixed, integral number of bits. By employing arithmetic coding, this restriction is removed, and a higher compression ratio can be achieved. Experiments show bitrate savings of 4 % for inter-frames and 10 % for intra-frames in this mode.
  As in H.261, the syntax of H.263 is structured as a hierarchy of four layers, each using a combination of fixed- and variable-length code. In the *syntax-based arithmetic coding* (*SAC*) mode, all variable-length coding operations are replaced with arithmetic coding operations. According to the syntax of each layer, the arithmetic encoder needs to code a different bitstream from various components. Since each of these bitstreams has a different distribution, H.263 specifies a model for each distribution, and the arithmetic coder switches the model on the fly, according to the syntax.
- **Advanced prediction mode.** In this mode, the macroblock size for motion compensation is reduced from 16 to 8. Four motion vectors (from each of the 8×8 blocks) are generated for each macroblock in the luminance image. Afterwards, each pixel in the 8×8 luminance prediction block takes a weighted sum of three predicted values based on the motion vector of the current luminance block and two out of the four motion vectors from the neighboring blocks—that is, one from the block at the left or right side of the current luminance block and one from the block above or below. Although sending four motion vectors incurs some additional overhead, the use of this mode generally yields better prediction and hence considerable gain in compression.
- **PB-frames mode.** As shown by MPEG (detailed discussions in Chap. 11), the introduction of a B-frame, which is predicted bidirectionally from both the previous frame and the future frame, can often improve the quality of prediction and hence the compression ratio without sacrificing picture quality. In H.263, a PB-frame consists of two pictures coded as one unit: one P-frame, predicted from the previous decoded I-frame or P-frame (or P-frame part of a PB-frame), and one B-frame, predicted from both the previous decoded I- or P-frame and the P-frame currently being decoded (Fig. 10.13).
  The use of the PB-frames mode is indicated in *PTYPE*. Since the P- and B-frames are closely coupled in the PB-frame, the bidirectional motion vectors for the B-frame need not be independently generated. Instead, they can be temporally scaled and further enhanced from the forward motion vector of the P-frame [8] so as to reduce the bitrate overhead for the B-frame. PB-frames mode yields satisfactory results for videos with moderate motion. Under large motions, PB-frames do not

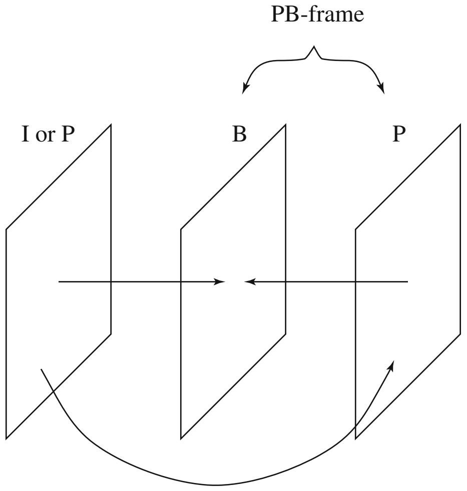


Fig. 10.13 A PB-frame in H.263

compress as well as B-frames. An improved mode has been developed in H.263 version 2.

10.5.3 H.263+ and H.263++

The second version of H.263, also known as H.263+ was approved in January 1998 by ITU-T Study Group 16. It is fully backward-compatible with the design of H.263 version 1.

The aim of H.263+ is to broaden the potential applications and offer additional flexibility in terms of custom source formats, different pixel aspect ratios, and clock frequencies. H.263+ includes numerous recommendations to improve code efficiency and error resilience [9]. It also provides 12 new negotiable modes, in addition to the four optional modes in H.263.

Since its development came after the standardization of MPEG-1 and 2, it is not surprising that it also adopts many aspects of the MPEG standards. Below, we mention only briefly some of these features and leave their detailed discussion to the next chapter, where we study the MPEG standards.

- The unrestricted motion vector mode is redefined under H.263+. It uses *Reversible Variable-Length Coding* (*RVLC*) to encode the difference motion vectors. The RVLC encoder is able to minimize the impact of transmission error by allowing the decoder to decode from both forward and reverse directions. The range of motion vectors is extended again to $[-256, 256]$. Refer to [10, 11] for more detailed discussions on the construction of RVLC.
- A *slice* structure is used to replace GOB for additional flexibility. A slice can contain a variable number of macroblocks. The transmission order can be either sequential or arbitrary, and the shape of a slice is not required to be rectangular.
- H.263+ implements *Temporal*, *SNR*, and *Spatial scalabilities*. Scalability refers to the ability to handle various constraints, such as display resolution, bandwidth, and

hardware capabilities. The enhancement layer for Temporal scalability increases perceptual quality by inserting B-frames between two P-frames.

SNR scalability is achieved by using various quantizers of smaller and smaller step_size to encode additional enhancement layers into the bitstream. Thus, the decoder can decide how many enhancement layers to decode according to computational or network constraints. The concept of Spatial scalability is similar to that of SNR scalability. In this case, the enhancement layers provide increased spatial resolution.

- H.263+ supports improved PB-frames mode, in which the two motion vectors of the B-frame do not have to be derived from the forward motion vector of the P-frame, as in version 1. Instead, they can be generated independently, as in MPEG-1 and 2.
- Deblocking filters in the coding loop reduce blocking effects. The filter is applied to the edge boundaries of the four luminance and two chrominance blocks. The coefficient weights depend on the quantizer step_size for the block. This technique results in better prediction as well as a reduction in blocking artifacts.

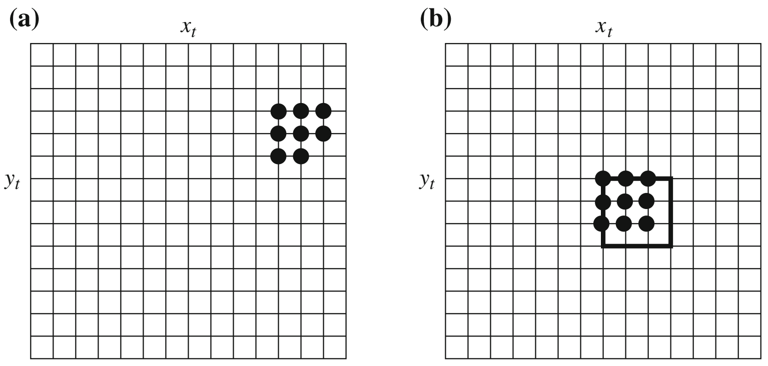
The development of H.263 continued beyond its second version. The third version, H.263 v3, also known as H.263++, was initially approved in the Year 2000. A further developed version was approved in 2005 [7]. H.263++ includes the baseline coding methods of H.263 and additional recommendations for *enhanced reference picture selection* (*ERPS*), *data partition slice* (*DPS*), and additional supplemental enhancement information.

ERPS mode operates by managing a multiframe buffer for stored frames, enhancing coding efficiency and error resilience. DPS mode provides additional enhancement to error resilience by separating header and motion vector data from DCT coefficient data in the bitstream and protects the motion vector data by using a reversible code. The additional supplemental enhancement information provides the ability to add backward-compatible enhancements to an H.263 bitstream.

Since we will describe in detail the newer standards H.264 and H.265 in Chap. 12, and many fundamental ideas are quite similar to the latest H.263, we will leave the rest of the discussions to that chapter.

10.6 Exercises

1. Describe how H.261 deals with *temporal* and *spatial* redundancies in video.
2. An H.261 video has the three color channels Y, C_r, C_b. Should **MV**s be computed for each channel and then transmitted? Justify your answer. If not, which channel should be used for motion compensation?
3. Thinking about my large collection of `JPEG` images (of my family taken in various locales), I decide to unify them and make them more accessible by simply combining them into a big H.261-compressed file. My reasoning is that I can simply use a viewer to step through the file, making a cohesive whole out of



● Pixel with intensity value 10

Other background (unmarked) pixels all have intensity value 100

Fig. 10.14 2D Logarithmic search for motion vectors: **a** Reference frame; **b** Target frame

my collection. Comment on the utility of this idea, in terms of the compression ratio achievable for the set of images.

4. In block-based video coding, what takes more effort: compression or decompression? Briefly explain why.
5. Work out the following problem of 2D Logarithmic Search for motion vectors in detail (see Fig. 10.14).
   The target (current) frame is a P-frame. The size of macroblocks is 4×4. The motion vector is $\mathbf{MV}(\Delta x, \Delta y)$, in which $\Delta x \in [-p, p]$, $\Delta y \in [-p, p]$. In this question, assume $p \equiv 5$.
   The macroblock in question (darkened) in the frame has its upper left corner at (x_t, y_t). It contains nine dark pixels, each with intensity value 10; the other seven pixels are part of the background, which has a uniform intensity value of 100. The reference (previous) frame has eight dark pixels.
   (a) What is the best Δx, Δy, and Mean Absolute Error (MAE) for this macroblock?
   (b) Show step by step how the 2D Logarithmic Search is performed, include the locations and passes of the search and all intermediate Δx, Δy, and MAEs.
6. The logarithmic **MV** search method is suboptimal, in that it relies on continuity in the residual frame.
   (a) Explain why that assumption is necessary, and offer a justification for it.
   (b) Give an example where this assumption fails.
   (c) Does the hierarchical search method suffer from suboptimality too?
7. A video sequence is to be encoded using H.263 in PB-mode, having a frame size of 4CIF, frame rate of 30 fps, and video length of 90 min. The following is known about the compression parameters: on average, two I-frames are encoded per second. The video at the required quality has an I-frame average compression

ratio of 10:1, an average P-frame compression ratio twice as good as I-frame, and an average B-frame compression ratio twice as good as P-frame. Assuming the compression parameters include all necessary headers, calculate the encoded video size.

8. Assuming a search window of size $2p + 1$, what is the complexity of motion estimation for a QCIF video in the advanced prediction mode of H.263, using
   (a) The brute-force (sequential search) method?
   (b) The 2D logarithmic method?
   (c) The hierarchical method?
9. Discuss how the advanced prediction mode in H.263 achieves better compression.
10. In H.263 motion estimation, the *median* of the motion vectors from three preceding macroblocks (see Fig. 10.11a) is used as a prediction for the current macroblock. It can be argued that the median may not necessarily reflect the best prediction. Describe some possible improvements on the current method.
11. H.263+ allows independent forward **MV**s for B-frames in a PB-frame. Compared to H.263 in PB-mode, what are the tradeoffs? What is the point in having PB joint coding if B-frames have independent motion vectors?

References

1. A.M. Tekalp, *Digital video processing* (Prentice Hall, Upper Saddle River, 1995)
2. C.A. Poynton, *Digital Video and HDTV Algorithms and Interfaces* (Morgan Kaufmann, San Francisco, 2002)
3. V. Bhaskaran, K. Konstantinides, *Image and Video Compression Standards: Algorithms and Architectures*, 2nd edn. (Kluwer Academic Publishers, Boston, 1997)
4. Y. Wang, J. Ostermann, Y.Q. Zhang, *Video Processing and Communications* (Prentice Hall, Upper Sadle River, 2002)
5. D. Marr, *Vision* (The MIT Press, San Francisco, 2010)
6. Video codec for audiovisual services at $p \times 64$ kbit/s. ITU-T Recommendation H.261, version 1 (1990) version 2, March (1993)
7. Video coding for low bit rate communication. ITU-T recommendation H.263, version 1 (1995), Version 2 (1998), version 3 (2000), Revised (2005)
8. B.G. Haskell, A. Puri, A. Netravali, *Digital Video: An Introduction to MPEG-2* (Chapman and Hall, New York, 1996)
9. G. Cote, B. Erol, M. Gallant, H.263+. IEEE Trans. Circuits Syst. Video Technol. **8**(7), 849–866 (1998)
10. Y. Takishima, M. Wada, H. Murakami, Reversible variable length codes. IEEE Trans. Commun. **43**(2–4), 158–162 (1995)
11. C.W. Tsai, J.L. Wu, On constructing the Huffman-code-based reversible variable-length codes. IEEE Trans. Commun. **49**(9), 1506–1509 (2001)