MPEG Video Coding: MPEG-1, 2, 4, and 7

11

11.1 Overview

The Moving Picture Experts Group (MPEG) was established in 1988 to create a standard for delivery of digital video and audio. Membership grew rapidly from about 25 experts in 1988 to a community of hundreds of companies and organizations [1]. It is appropriately recognized that proprietary interests need to be maintained within the family of MPEG standards. This is accomplished by defining only a compressed bitstream that implicitly defines the decoder. The compression algorithms, and thus the encoders, are completely up to the manufacturers.

In this chapter, we will study some of the most important design issues of MPEG-1 and 2, followed by some basics of the later standards, MPEG-4 and 7, which have very different objectives.

With the emerging new video compression standards such as H.264 and H.265 (to be discussed in Chap. 12), one might view these MPEG standards as *old*, i.e., outdated. This is simply not a concern because: (a) The fundamental technology of hybrid coding and most important concepts that we will introduce here, such as motion compensation, DCT-based transform coding, and scalabilities, are used in all old and new standards. (b) Although the visual-object-based video representation and compression approach developed in MPEG-4 and 7 has not been commonly used in current popular standards, it has a great potential to be adopted in the future when the necessary Computer Vision technology for automatic object detection becomes more readily available.

11.2 MPEG-1

The MPEG-1 audio/video digital compression standard [2,3] was approved by the International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) MPEG group in November 1991 for *Coding of Moving Pictures*

Z.-N. Li et al., *Fundamentals of Multimedia*,
Texts in Computer Science, DOI: 10.1007/978-3-319-05290-8_11,

and Associated Audio for Digital Storage Media at up to about 1.5 Mbit/s [4]. Common digital storage media include compact discs (CDs) and video compact discs (VCDs). Out of the specified 1.5, 1.2 Mbps is intended for coded video, and 256 kbps(kilobits per second) can be used for stereo audio. This yields a picture quality comparable to VHS cassettes and a sound quality equal to CD audio.

In general, MPEG-1 adopts the CCIR601 digital TV format, also known as *Source Input Format* (SIF). MPEG-1 supports only noninterlaced video. Normally, its picture resolution is 352 × 240 for NTSC video at 30 fps, or 352 × 288 for PAL video at 25 fps. It uses 4:2:0 chroma subsampling.

The MPEG-1 standard, also referred to as ISO/IEC 11172 [4], has five parts: 11172-1 Systems, 11172-2 Video, 11172-3 Audio, 11172-4 Conformance, and 11172-5 Software. Briefly, Systems takes care of, among many things, dividing output into packets of bitstreams, multiplexing, and synchronization of the video and audio streams. Conformance (or compliance) specifies the design of tests for verifying whether a bitstream or decoder complies with the standard. Software includes a complete software implementation of the MPEG-1 standard decoder and a sample software implementation of an encoder.

As in H.261 and H.263, MPEG-1 employs the technology of *Hybrid Coding*, i.e., a combination of interpicture motion predictions and transform coding on residual errors. We will examine the main features of MPEG-1 video coding and leave discussions of MPEG audio coding to Chap. 14.

11.2.1 Motion Compensation in MPEG-1

As discussed in the last chapter, motion-compensation-based video encoding in H.261 works as follows: In motion estimation, each macroblock of the target P-frame is assigned a best matching macroblock from the previously coded I- or P-frame. This is called a *prediction*. The difference between the macroblock and its matching macroblock is the *prediction error*, which is sent to DCT and its subsequent encoding steps.

Since the prediction is from a previous frame, it is called *forward prediction*. Due to unexpected movements and occlusions in real scenes, the target macroblock may not have a good matching entity in the previous frame. Figure 11.1 illustrates that the macroblock containing part of a ball in the target frame cannot find a good matching macroblock in the previous frame, because half of the ball was occluded by another object. However, a match can readily be obtained from the next frame.

MPEG introduces a third frame type—*B-frames*—and their accompanying bidirectional motion compensation. Figure 11.2 illustrates the motion-compensation-based B-frame coding idea. In addition to the forward prediction, a backward prediction is also performed, in which the matching macroblock is obtained from a future I- or P-frame in the video sequence. Consequently, each macroblock from a B-frame will specify up to *two* motion vectors, one from the forward and one from the backward prediction.

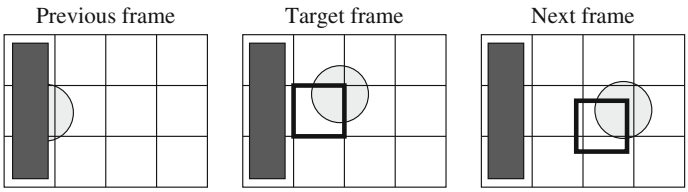

Fig. 11.1 The need for bidirectional search

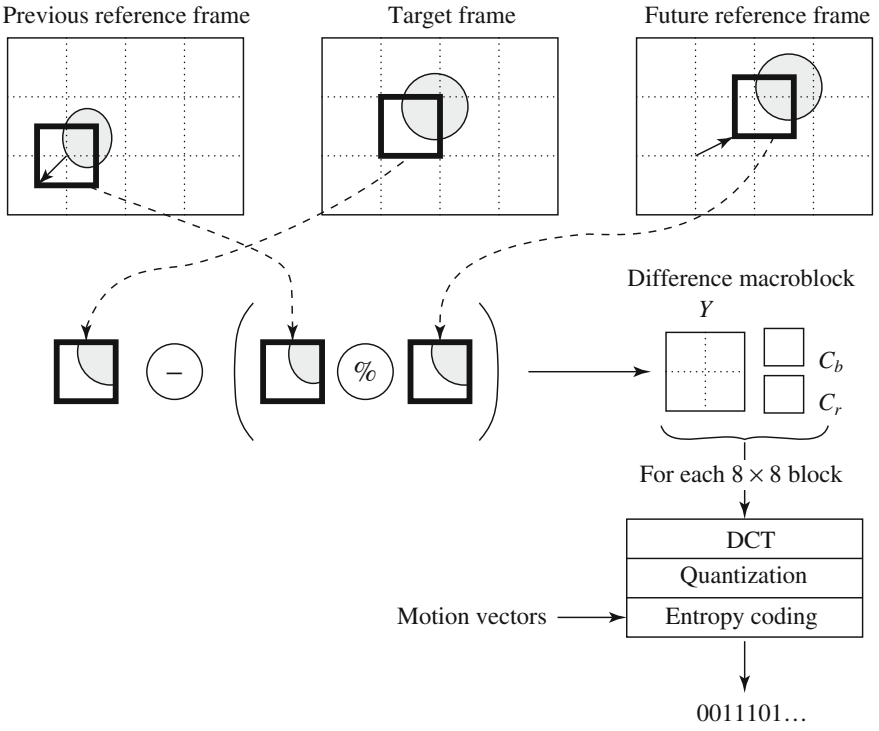

Fig. 11.2 B-frame coding based on bidirectional motion compensation

If matching in both directions is successful, two motion vectors will be sent, and the two corresponding matching macroblocks are averaged (indicated by "%" in the figure) before comparing to the target macroblock for generating the prediction error. If an acceptable match can be found in only one of the reference frames, only one motion vector and its corresponding macroblock will be used from either the forward or backward prediction.

Figure 11.3 illustrates a possible sequence of video frames. The actual frame pattern is determined at encoding time and is specified in the video's header. MPEG uses M to indicate the interval between a P-frame and its preceding I- or P-frame, and N to indicate the interval between two consecutive I-frames. In Fig. 11.3, $M = 3$, $N = 9$. A special case is $M = 1$, when no B-frame is used.

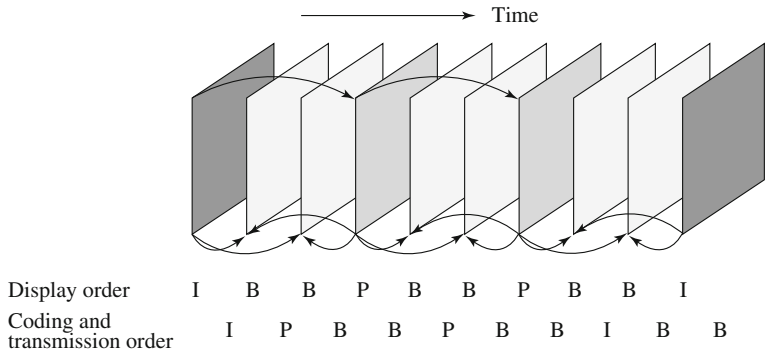

Fig. 11.3 MPEG frame sequence

Since the MPEG encoder and decoder cannot work for any macroblock from a B-frame without its succeeding P- or I-frame, the actual coding and transmission order (shown at the bottom of Fig. 11.3) is different from the display order of the video (shown above). The inevitable delay and need for buffering become an important issue in real-time network transmission, especially in streaming MPEG video.

11.2.2 Other Major Differences from H.261

Beside introducing bidirectional motion compensation (the B-frames), MPEG-1 also differs from H.261 in the following aspects:

- **Source formats** H.261 supports only CIF (352 × 288) and QCIF (176 × 144) source formats. MPEG-1 supports SIF (352 × 240 for NTSC, 352 × 288 for PAL). It also allows specification of other formats, as long as the *constrained parameter set* (*CPS*), shown in Table 11.1, is satisfied.
- **Slices** Instead of GOBs, as in H.261, an MPEG-1 picture can be divided into one or more *slices* (Fig. 11.4), which are more flexible than GOBs. They may contain variable numbers of macroblocks in a single picture and may also start and end anywhere, as long as they fill the whole picture. Each slice is coded independently. For example, the slices can have different scale factors in the quantizer. This provides additional flexibility in bitrate control.
  Moreover, the slice concept is important for error recovery, because each slice has a unique *slice_start_code*. A slice in MPEG is similar to the GOB in H.261 (and H.263): it is the lowest level in the MPEG layer hierarchy that can be fully recovered without decoding the entire set of variable-length codes in the bitstream.
- **Quantization** MPEG-1 quantization uses different quantization tables for its intra- and intercoding (Tables 11.2 and 11.3). The quantizer numbers for intracoding (Table 11.2) vary within a macroblock. This is different from H.261, where all quantizer numbers for AC coefficients are constant within a macroblock.

Table 11.1 The MPEG-1 constrained parameter set

Parameter	Value
Horizontal size of picture	≤768
Vertical size of picture	≤576
Number of macroblocks/picture	≤396
Number of macroblocks/second	≤9, 900
Frame rate	≤30 fps
Bitrate	≤1, 856 kbps

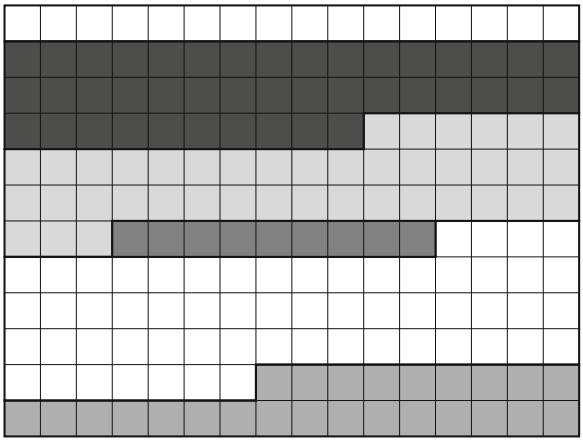

Fig. 11.4 Slices in an MPEG-1 picture

Table 11.2 Default quantization table (Q_1) for intracoding

8	16	19	22	26	27	29	34
16	16	22	24	27	29	34	37
19	22	26	27	29	34	34	38
22	22	26	27	29	34	37	40
22	26	27	29	32	35	40	48
26	27	29	32	35	40	48	58
26	27	29	34	38	46	56	69
27	29	35	38	46	56	69	83

Table 11.3 Default quantization table (Q_2) for intercoding

16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16

The $step\_size[i, j]$ value is now determined by the product of $Q[i, j]$ and $scale$, where Q_1 or Q_2 is one of the above quantization tables and $scale$ is an integer in the range [1, 31]. Using DCT and $QDCT$ to denote the DCT coefficients before and after quantization, for DCT coefficients in intramode,

$$\text{QDCT}[i, j] = \text{round}\left(\frac{8 \times DCT[i, j]}{step\_size[i, j]}\right) = \text{round}\left(\frac{8 \times DCT[i, j]}{Q_1[i, j] \times scale}\right), \tag{11.1}$$

and for DCT coefficients in intermode,

$$\text{QDCT}[i, j] = \left\lfloor \frac{8 \times DCT[i, j]}{step\_size[i, j]} \right\rfloor = \left\lfloor \frac{8 \times DCT[i, j]}{Q_2[i, j] \times scale} \right\rfloor, \tag{11.2}$$

where Q_1 and Q_2 refer to Tables 11.2 and 11.3, respectively.

Again, a `round` operator is typically used in Eq. (11.1) and hence leaves no dead zone, whereas a `floor` operator is used in Eq. (11.2), leaving a center dead zone in its quantization space.

- To increase precision of the motion-compensation-based predictions and hence reduce prediction errors, MPEG-1 allows motion vectors to be of subpixel precision (1/2 pixel). The technique of bilinear interpolation discussed in Sect. 10.5.1 for H.263 can be used to generate the needed values at half-pixel locations.
- MPEG-1 supports larger gaps between I- and P-frames and consequently a much larger motion-vector search range. Compared to the maximum range of ± 15 pixels for motion vectors in H.261, MPEG-1 supports a range of $[-512, 511.5]$ for half-pixel precision and $[-1{,}024, 1{,}023]$ for full-pixel precision motion vectors. However, due to the practical limitation in its picture resolution, such a large maximum range might never be used.
- The MPEG-1 bitstream allows random access. This is accomplished by the *Group of Pictures* (*GOP*) layer, in which each GOP is time-coded. In addition, the first frame in any GOP is an I-frame, which eliminates the need to reference other frames. Thus, the GOP layer allows the decoder to seek a particular position within the bitstream and start decoding from there.

Table 11.4 lists typical sizes (in kilobytes) for all types of MPEG-1 frames. It can be seen that the typical size of compressed P-frames is significantly smaller than that of I-frames, because interframe compression exploits temporal redundancy. Notably, B-frames are even smaller than P-frames, due partially to the advantage of bidirectional prediction. It is also because B-frames are often given the lowest priority in terms of preservation of quality; hence, a higher compression ratio can be assigned.

11.2.3 MPEG-1 Video Bitstream

Figure 11.5 depicts the six hierarchical layers for the bitstream of an MPEG-1 video.

1. **Sequence layer** A video sequence consists of one or more groups of pictures (GOPs). It always starts with a sequence header. The header contains information about the picture, such as *horizontal_size* and *vertical_size*, *pixel_aspect_ratio*, *frame_rate*, *bit_rate*, *buffer_size*, *quantization_matrix*, and so on. Optional sequence headers between GOPs can indicate parameter changes.

Table 11.4 Typical compression performance of MPEG-1 frames

Type	Size (kB)	Compression
I	18	7:1
P	6	20:1
B	2.5	50:1
Average	4.8	27:1

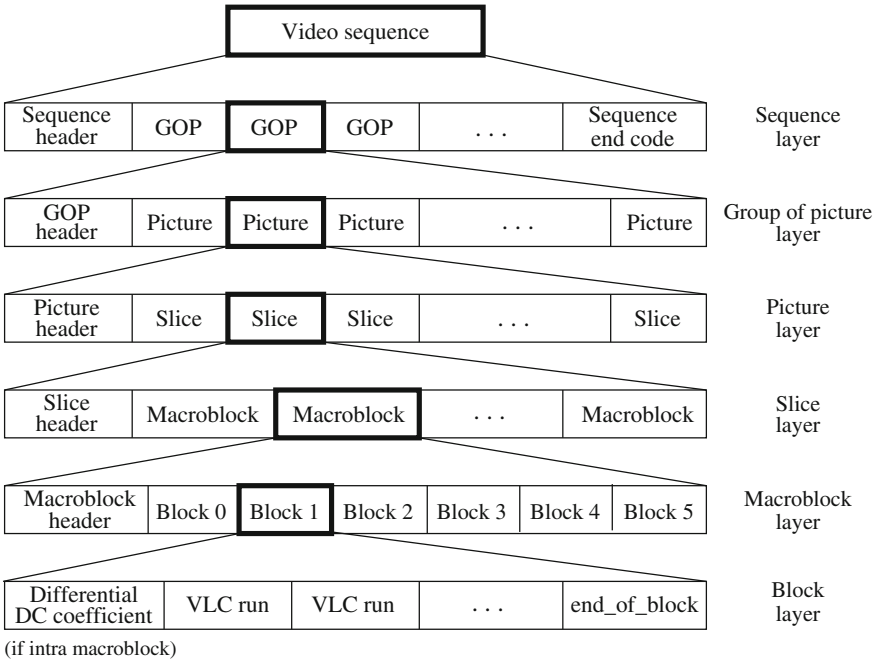

Fig. 11.5 Layers of MPEG-1 video bitstream

2. **GOPs layer** A GOP contains one or more pictures, one of which must be an I-picture. The GOP header contains information such as *time_code* to indicate hour-minute-second-frame from the start of the sequence.
3. **Picture layer** The three common MPEG-1 picture types are *I-picture* (intra-coding), *P-picture* (predictive coding), and *B-picture* (Bidirectional predictive coding), as discussed above. There is also an uncommon type, *D-picture* (DC coded), in which only DC coefficients are retained. MPEG-1 does not allow mixing D-pictures with other types, which makes D-pictures impractical.
4. **Slice layer** As mentioned earlier, MPEG-1 introduced the slice notion for bitrate control and for recovery and synchronization after lost or corrupted bits. Slices

may have variable numbers of macroblocks in a single picture. The length and position of each slice are specified in the header.

5. **Macroblock layer** Each macroblock consists of four Y blocks, one C_b block, and one C_r block. All blocks are 8×8.
6. **Block layer** If the blocks are intracoded, the differential DC coefficient (DPCM of DCs, as in JPEG) is sent first, followed by variable-length codes (VLC), for AC coefficients. Otherwise, DC and AC coefficients are both coded using the variable-length codes.

Mitchell et al. [5] provide detailed information on the headers in various MPEG-1 layers.

11.3 MPEG-2

Development of the MPEG-2 standard started in 1990. Unlike MPEG-1, which is basically a standard for storing and playing video on the CD of a single computer at a low bitrate (1.5 Mbps), MPEG-2 [6] is for higher quality video at a bitrate of more than 4 Mbps. It was initially developed as a standard for digital broadcast TV.

In the late 1980s, *Advanced TV* (*ATV*) was envisioned, to broadcast HDTV via terrestrial networks. During the development of MPEG-2, digital ATV finally took precedence over various early attempts at analog solutions to HDTV. MPEG-2 has managed to meet the compression and bitrate requirements of digital TV/HDTV and in fact supersedes a separate standard, MPEG-3, initially thought necessary for HDTV.

The MPEG-2 audio/video compression standard, also referred to as ISO/IEC 13818 [7], was approved by the ISO/IEC Moving Picture Experts Group in November 1994. Similar to MPEG-1, it has Parts for Systems, Video, Audio, Conformance, and Software, plus other aspects. Part 2, the video compression part of the standard, ISO/IEC 13818-2 is also known as H.262 in ITU-T (International Telecommunication Union-Telecommunications). MPEG-2 has gained wide acceptance beyond broadcasting digital TV over terrestrial, satellite, or cable networks. Among various applications such as Interactive TV, it was also adopted for *digital video disc*s or *digital versatile disc*s (DVDs).

MPEG-2 defined seven *profiles* aimed at different applications (e.g., low-delay videoconferencing, scalable video, HDTV). The profiles are *Simple, Main, SNR scalable, Spatially scalable, High, 4:2:2*, and *Multiview* (where two views would refer to stereoscopic video). Within each profile, up to four *levels* are defined. As Table 11.5 shows, not all profiles have four levels. For example, the Simple profile has only the Main level; whereas the High profile does not have the Low level.

Table 11.6 lists the four levels in the Main profile, with the maximum amount of data and targeted applications. For example, the High level supports a high picture resolution of $1{,}920 \times 1{,}152$, a maximum frame rate of 60 fps, maximum pixel rate of 62.7×10^6 per second, and a maximum data rate after coding of 80 Mbps. The

Table 11.5 Profiles and Levels in MPEG-2

Level	Simple profile	Main profile	SNR scalable profile	Spatially scalable profile	High profile	4:2:2 profile	Multiview profile
High		*			*	*	
High 1440		*		*	*	*	
Main	*	*	*		*	*	*
Low	*	*	*				

Table 11.6 Four levels in the main profile of MPEG-2

Level	Maximum resolution	Maximum fps	Maximum pixels/sec	Maximum coded data rate (Mbps)	Application
High	$1,920 \times 1,152$	60	62.7×10^6	80	Film production
High 1440	$1,440 \times 1,152$	60	47.0×10^6	60	Consumer HDTV
Main	720×576	30	10.4×10^6	15	Studio TV
Low	352×288	30	3.0×10^6	4	Consumer tape equivalent

Low level is targeted at SIF video; hence, it provides backward compatibility with MPEG-1. The Main level is for CCIR601 video, whereas High 1440 and High levels are aimed at European HDTV and North American HDTV, respectively.

The DVD video specification allows only four display resolutions: for example, at 29.97 fps, interlaced, 720×480, 704×480, 352×480, and 352×240. Hence, the DVD video standard uses only a restricted form of the MPEG-2 Main profile at the Main and Low levels.

11.3.1 Supporting Interlaced Video

MPEG-1 supports only noninterlaced (progressive) video. Since MPEG-2 is adopted by digital broadcast TV, it must also support interlaced video, because this is one of the options for digital broadcast TV and HDTV.

As mentioned earlier, in interlaced video each frame consists of two fields, referred to as the *top-field* and the *bottom-field*. In a *frame-picture*, all scanlines from both fields are interleaved to form a single frame. This is then divided into 16×16 macroblocks and coded using motion compensation. On the other hand, if each field is treated as a separate picture, then it is called *field picture*. As Fig. 11.6a shows, each frame-picture can be split into two field pictures. The figure shows 16 scanlines from a frame-picture on the left, as opposed to 8 scanlines in each of the two field portions of a field picture on the right.

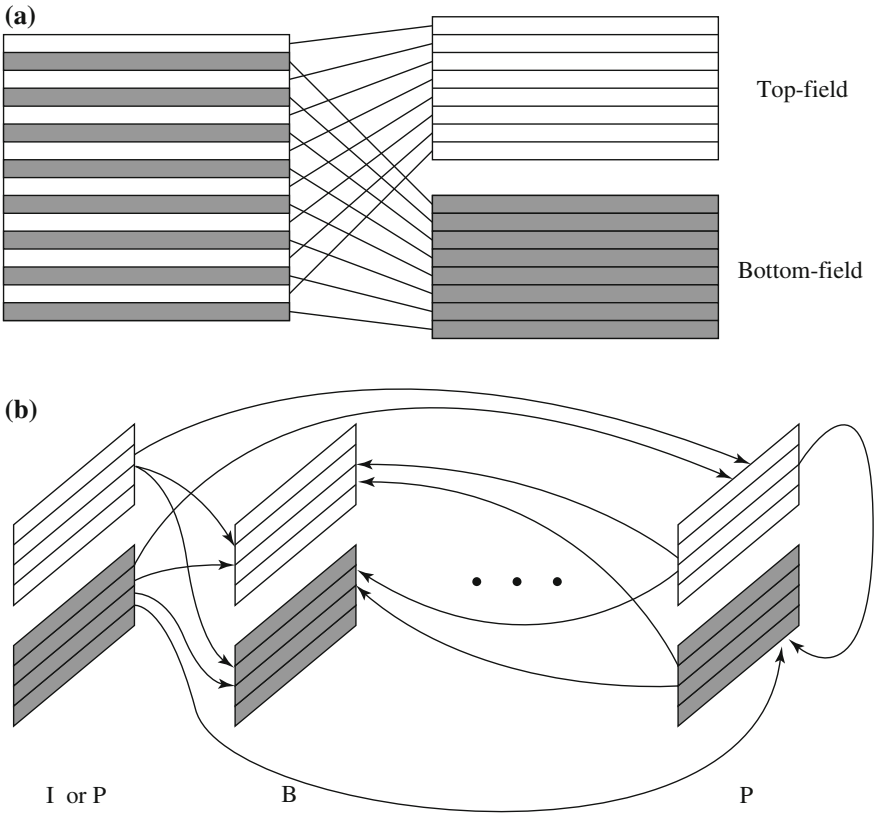

Fig. 11.6 Field pictures and field prediction for field pictures in MPEG-2: **a** frame-picture versus field pictures; **b** field prediction for field pictures

We see that, in terms of display area on the monitor/TV, each 16-column × 16-row macroblock in the field picture corresponds to a 16 × 32 block area in the frame picture, whereas each 16 × 16 macroblock in the frame picture corresponds to a 16 × 8 block area in the field picture. As shown below, this observation will become an important factor in developing different modes of predictions for motion-compensation-based video coding.

Five Modes of Predictions

MPEG-2 defines *frame prediction* and *field prediction* as well as five different prediction modes, suitable for a wide range of applications where the requirement for the accuracy and speed of motion compensation vary.

1. **Frame prediction for frame-pictures.** This is identical to MPEG-1 motion-compensation-based prediction methods in both P-frames and B-frames. Frame

prediction works well for videos containing only slow and moderate object and camera motions.

2. **Field prediction for field pictures.** (See Fig. 11.6b) This mode uses a macroblock size of 16×16 from field pictures. For P-field pictures (the rightmost ones shown in the figure), predictions are made from the two most recently encoded fields. Macroblocks in the top-field picture are forward-predicted from the top-field or bottom-field pictures of the preceding I- or P-frame. Macroblocks in the bottom-field picture are predicted from the top-field picture of the same frame or the bottom-field picture of the preceding I- or P-frame.
   For B-field pictures, both forward and backward predictions are made from field pictures of preceding and succeeding I- or P-frames. No regulation requires that field "parity" be maintained—that is, the top-field and bottom-field pictures can be predicted from either the top or bottom fields of the reference pictures.
3. **Field prediction for frame-pictures.** This mode treats the top-field and bottom-field of a frame-picture separately. Accordingly, each 16×16 macroblock from the target frame-picture is split into two 16×8 parts, each coming from one field. Field prediction is carried out for these 16×8 parts in a manner similar to that shown in Fig. 11.6b. Besides the smaller block size, the only difference is that the bottom-field will not be predicted from the top-field of the same frame, since we are dealing with frame-pictures now.
   For example, for P-frame-pictures, the bottom 16×8 part will instead be predicted from either field from the preceding I- or P-frame. Two motion vectors are thus generated for each 16×16 macroblock in the P-frame-picture. Similarly, up to four motion vectors can be generated for each macroblock in the B-frame-picture.
4. **16×8 MC for field pictures.** Each 16×16 macroblock from the target field picture is now split into top and bottom 16×8 halves—that is, the first eight rows and the next eight rows. Field prediction is performed on each half. As a result, two motion vectors will be generated for each 16×16 macroblock in the P-field picture and up to four motion vectors for each macroblock in the B-field picture. This mode is good for finer motion compensation when motion is rapid and irregular.
5. **Dual-prime for P-pictures.** This is the only mode that can be used for either frame-pictures or field pictures. At first, field prediction from each previous field with the same parity (top or bottom) is made. Each motion vector **MV** is then used to derive a calculated motion vector **CV** in the field with the opposite parity, taking into account the temporal scaling and vertical shift between lines in the top and bottom fields. In this way, the pair **MV** and **CV** yields two preliminary predictions for each macroblock. Their prediction errors are averaged and used as the final prediction error. This mode is aimed at mimicking B-picture prediction for P-pictures without adopting backward prediction (and hence less encoding delay).

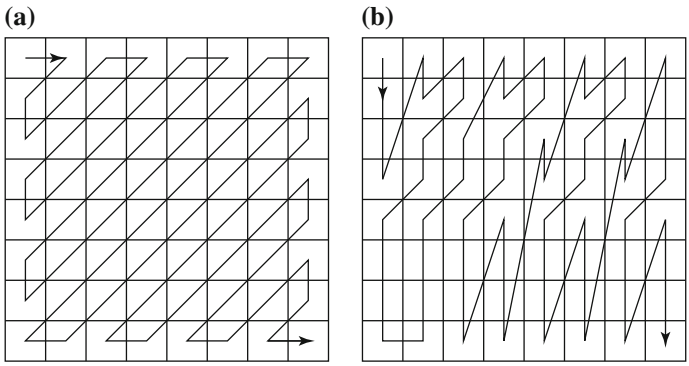

Fig. 11.7 **a** Zigzag (*progressive*) and **b** alternate (*interlaced*) scans of DCT coefficients for videos in MPEG-2

Alternate Scan and Field_DCT

Alternate Scan and *Field_DCT* are techniques aimed at improving the effectiveness of DCT on prediction errors. They are applicable only to frame-pictures in interlaced videos.

After frame prediction in frame-pictures, the prediction error is sent to DCT, where each block is of size 8×8. Due to the nature of interlaced video, the consecutive rows in these blocks are from different fields; hence, there is less correlation between them than between the alternate rows. This suggests that the DCT coefficients at low vertical spatial frequencies tend to have reduced magnitudes, compared to the ones in noninterlaced video.

Based on the above analysis, an alternate scan is introduced. It may be applied on a picture-by-picture basis in MPEG-2 as an alternative to a zigzag scan. As Fig. 11.7a indicates, zigzag scan assumes that in noninterlaced video, the DCT coefficients at the upper left corner of the block often have larger magnitudes. Alternate scan (Fig. 11.7b) recognizes that in interlaced video, the vertically higher spatial frequency components may have larger magnitudes and thus allows them to be scanned earlier in the sequence. Experiments have shown [6] that alternate scan can improve the PSNR by up to 0.3 dB over zigzag scan and is most effective for videos with fast motion.

In MPEG-2, Field_DCT can address the same issue. Before applying DCT, rows in the macroblock of frame-pictures can be reordered, so that the first eight rows are from the top-field and the last eight are from the bottom-field. This restores the higher spatial redundancy (and correlation) between consecutive rows. The reordering will be reversed after the IDCT. Field_DCT is not applicable to chrominance images, where each macroblock has only 8×8 pixels.

11.3.2 MPEG-2 Scalabilities

As in JPEG2000, *scalability* is also an important issue for MPEG-2. Since MPEG-2 is designed for a variety of applications, including digital TV and HDTV, the video will often be transmitted over networks with very different characteristics. Therefore it is necessary to have a single-coded bitstream that is *scalable* to various bitrates.

MPEG-2 *scalable coding* is also known as *layered coding*, in which a base layer and one or more enhancement layers can be defined. The base layer can be independently encoded, transmitted, and decoded, to obtain basic video quality. The encoding and decoding of the enhancement layer, however, depends on the base layer or the previous enhancement layer. Often, only one enhancement layer is employed, which is called two-layer scalable coding.

Scalable coding is suitable for MPEG-2 video transmitted over networks with following characteristics.

- **Very different bitrates** If the link speed is slow, only the bitstream from the base layer will be sent. Otherwise, bitstreams from one or more enhancement layers will also be sent, to achieve improved video quality.
- **Variable-bitrate (VBR) channels** When the bitrate of the channel deteriorates, bitstreams from fewer or no enhancement layers will be transmitted, and vice versa.
- **Noisy connections** The base layer can be better protected or sent via channels known to be less noisy.

Moreover, scalable coding is ideal for progressive transmission: bitstreams from the base layer are sent first, to give users a fast and basic view of the video, followed by gradually increased data and improved quality. This can be useful for delivering compatible digital TV (ATV) and HDTV.

MPEG-2 supports the following scalabilities:

- **SNR scalability** The enhancement layer provides higher SNR.
- **Spatial scalability** The enhancement layer provides higher spatial resolution.
- **Temporal scalability** The enhancement layer facilitates higher frame rate.
- **Hybrid scalability** This combines any two of the above three scalabilities.
- **Data partitioning** Quantized DCT coefficients are split into partitions.

SNR Scalability

Figure 11.8 illustrates how SNR scalability works in the MPEG-2 encoder and decoder.

The MPEG-2 SNR scalable encoder generates output bitstreams `Bits_base` and `Bits_enhance` at two layers. At the base layer, a coarse quantization of the DCT coefficients is employed, which results in fewer bits and a relatively low-quality video. After variable-length coding, the bitstream is called `Bits_base`. The coarsely quantized DCT coefficients are then inversely quantized (Q^{-1}) and fed to the enhancement layer, to be compared with the original DCT coefficient.

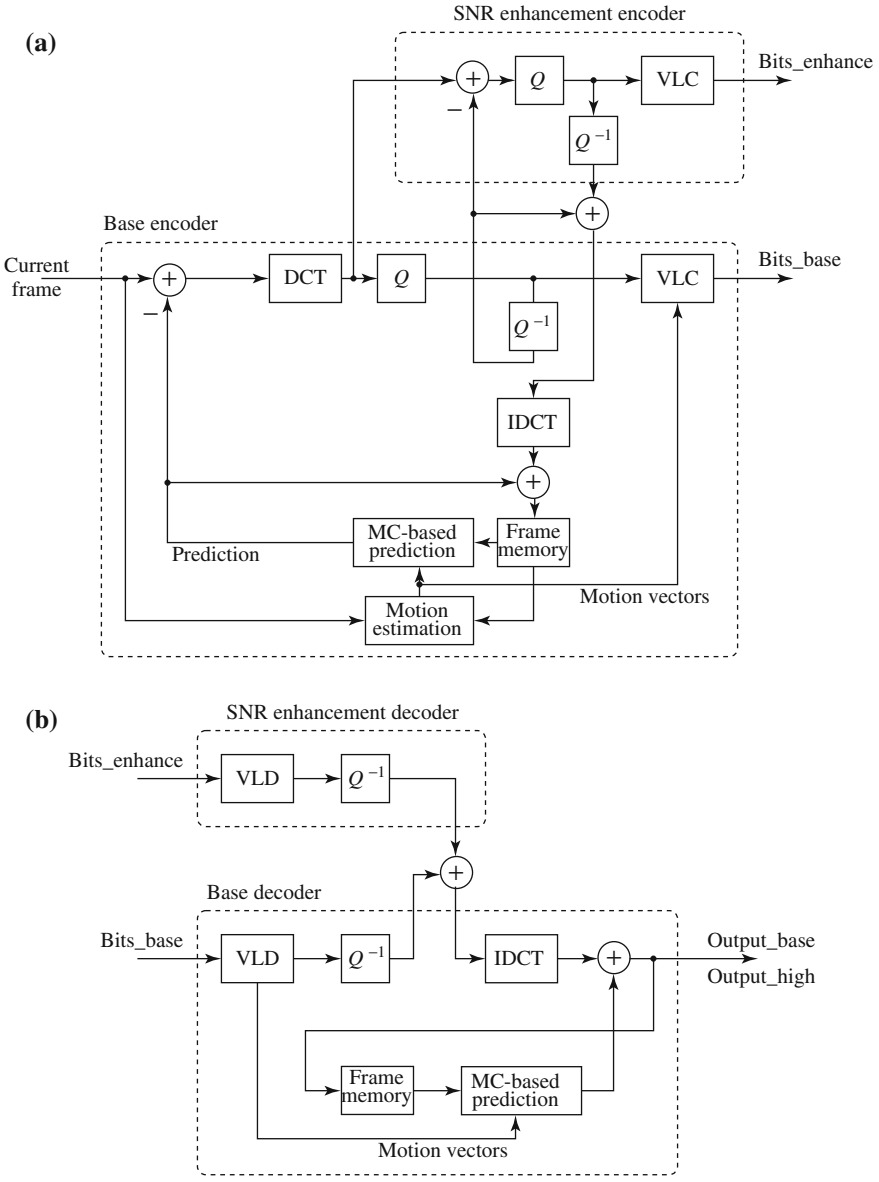

Fig. 11.8 MPEG-2 SNR scalability: **a** encoder; **b** decoder

Their difference is finely quantized to generate a *DCT coefficient refinement*, which, after variable-length coding, becomes the bitstream called `Bits_enhance`. The inversely quantized coarse and refined DCT coefficients are added back, and after inverse DCT (IDCT), they are used for motion-compensated prediction for the next

frame. Since the enhancement/refinement over the base layer improves the signal-to-noise-ratio, this type of scalability is called *SNR scalability*.

If, for some reason (e.g., the breakdown of some network channel), `Bits_enhance` from the enhancement layer cannot be obtained, the above scalable scheme can still work using `Bits_base` only. In that case, the input from the inverse quantizer (Q^{-1}) of the enhancement layer simply has to be treated as zero.

The decoder (Fig. 11.8b) operates in reverse order to the encoder. Both `Bits_base` and `Bits_enhance` are variable-length decoded (VLD) and inversely quantized (Q^{-1}) before they are added together to restore the DCT coefficients. The remaining steps are the same as in any motion-compensation-based video decoder. If both bitstreams (`Bits_base` and `Bits_enhance`) are used, the output video is `Output_high` with enhanced quality. If only `Bits_base` is used, the output video `Output_base` is of basic quality.

Spatial Scalability

The base and enhancement layers for MPEG-2 spatial scalability are not as tightly coupled as in SNR scalability; hence, this type of scalability is somewhat less complicated. We will not show the details of both encoder and decoder, as we did above, but will explain only the encoding process, using high-level diagrams.

The base layer is designed to generate a bitstream of reduced-resolution pictures. Combining them with the enhancement layer produces pictures at the original resolution. As Fig. 11.9a shows, the original video data is spatially decimated by a factor of 2 and sent to the base layer encoder. After the normal coding steps of motion compensation, DCT on prediction errors, quantization, and entropy coding, the output bitstream is `Bits_base`.

As Fig. 11.9b indicates, the predicted macroblock from the base layer is now spatially interpolated to get to resolution 16×16. This is then combined with the normal, temporally predicted macroblock from the enhancement layer itself, to form the prediction macroblock for the purpose of motion compensation in this layered coding. The spatial interpolation here adopts *bilinear interpolation*, as discussed before.

The combination of macroblocks uses a simple weight table, where the value of the weight w is in the range of [0, 1.0]. If $w = 0$, no consideration is given to the predicted macroblock from the base layer. If $w = 1$, the prediction is entirely from the base layer. Normally, both predicted macroblocks are linearly combined, using the weights w and $1 - w$, respectively. To achieve minimum prediction errors, MPEG-2 encoders have an analyzer to choose different w values from the weight table on a macroblock basis.

Temporal Scalability

Temporally scalable coding has both the base and enhancement layers of video at a reduced temporal rate (frame rate). The reduced frame rates for the layers are often

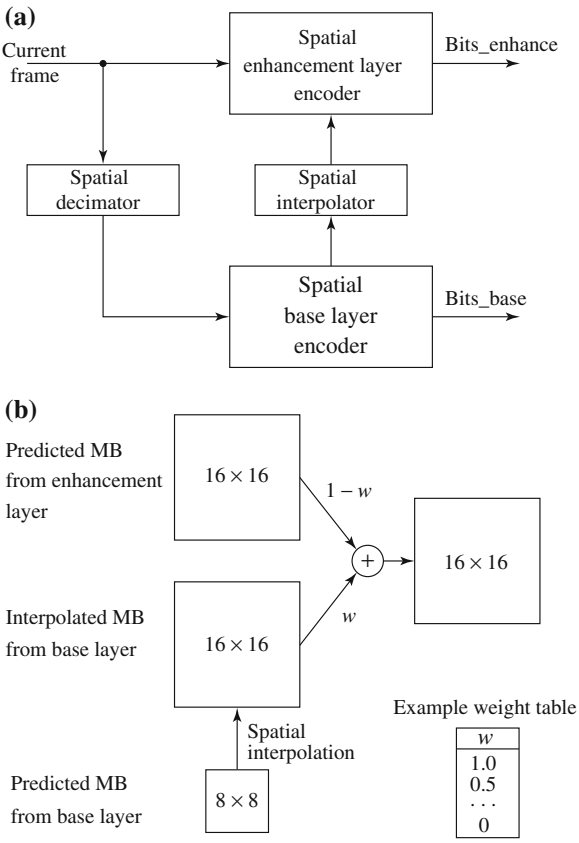

Fig. 11.9 Encoder for MPEG-2 Spatial scalability: **a** block diagram; **b** combining temporal and spatial predictions for encoding at enhancement layer

the same; however, they could also be different. Pictures from the base layer and enhancement layer(s) have the same spatial resolution as in the input video. When combined, they restore the video to its original temporal rate.

Figure 11.10 illustrates the MPEG-2 implementation of temporal scalability. The input video is temporally demultiplexed into two pieces, each carrying half the original frame rate. As before, the base layer encoder carries out the normal single-layer coding procedures for its own input video and yields the output bitstream `Bits_base`.

The prediction of matching macroblocks at the enhancement layer can be obtained in two ways [6]: *Interlayer motion-compensated prediction* or *combined motion-compensated prediction and interlayer motion-compensated prediction*.

- **Interlayer motion-compensated prediction** (Fig. 11.10b) The macroblocks of B-frames for motion compensation at the enhancement layer are predicted from the preceding and succeeding frames (either I-, P-, or B-) at the base layer, so as to exploit the possible interlayer redundancy in motion compensation.

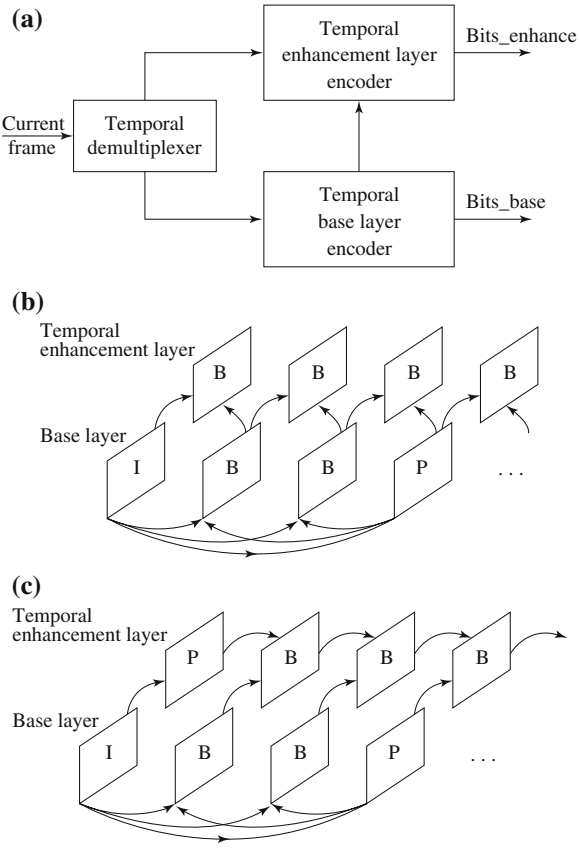

Fig. 11.10 Encoder for MPEG-2 temporal scalability: **a** block diagram; **b** interlayer motion-compensated prediction; **c** combined motion-compensated prediction and interlayer motion-compensated prediction

- **Combined motion-compensation prediction and interlayer motion-compensation prediction** (Fig. 11.10c) This further combines the advantages of the ordinary forward prediction and the above interlayer prediction. Macroblocks of B-frames at the enhancement layer are forward-predicted from the preceding frame at its own layer and "backward"-predicted from the preceding (or, alternatively, succeeding) frame at the base layer. At the first frame, the P-frame at the enhancement layer adopts only forward prediction from the I-frame at the base layer.

Hybrid Scalability

Any two of the above three scalabilities can be combined to form hybrid scalability. These combinations are:

- Spatial and temporal hybrid scalability
- SNR and spatial hybrid scalability

- SNR and temporal hybrid scalability

  Usually, a three-layer hybrid coder will be adopted, consisting of base layer, enhancement layer 1, and enhancement layer 2.

  For example, for Spatial and temporal hybrid scalability, the base layer and enhancement layer 1 will provide spatial scalability, and enhancement layers 1 and 2 will provide temporal scalability, in which enhancement layer 1 is effectively serving as a base layer.

  For the encoder, the incoming video data is first temporally demultiplexed into two streams: one to enhancement layer 2; the other to enhancement layer 1 and the base layer (after further spatial decimation for the base layer).

  The encoder generates three output bitstreams: (a) `Bits_base` from the base layer, (b) spatially enhanced `Bits_enhance1` from enhancement layer 1, and (c) spatially and temporally enhanced `Bits_enhance2` from enhancement layer 2.

  The implementations of the other two hybrid scalabilities are similar and are left as exercises.

Data Partitioning

The compressed video stream is divided into two partitions. The base partition contains lower-frequency DCT coefficients, and the enhancement partition contains high-frequency DCT coefficients. Although the partitions are sometimes also referred to as layers (base layer and enhancement layer), strictly speaking, data partitioning does not conduct the same type of layered coding, since a single stream of video data is simply divided up and does not depend further on the base partition in generating the enhancement partition. Nevertheless, data partitioning can be useful for transmission over noisy channels and for progressive transmission.

11.3.3 Other Major Differences from MPEG-1

- **Better resilience to bit errors.** Since MPEG-2 video will often be transmitted on various networks, some of them noisy and unreliable, bit errors are inevitable. To cope with this, MPEG-2 systems have two types of streams: *Program* and *Transport*. The Program stream is similar to the Systems stream in MPEG-1; hence, it also facilitates backward compatibility with MPEG-1.

  The Transport stream aims at providing error resilience and the ability to include multiple programs with independent time bases in a single stream, for asynchronous multiplexing and network transmission. Instead of using long, variable-length packets, as in MPEG-1 and in the MPEG-2 Program stream, it uses fixed-length (188-byte) packets. It also has a new header syntax, for better error checking and correction.
- **Support of 4:2:2 and 4:4:4 chroma subsampling.** In addition to 4:2:0 chroma subsampling, as in H.261 and MPEG-1, MPEG-2 also allows 4:2:2 and 4:4:4, to increase color quality. As discussed in Chap. 5, each chrominance picture in 4:2:2

Table 11.7 Possible nonlinear scale in MPEG-2

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
scale_i	1	2	3	4	5	6	7	8	10	12	14	16	18	20	22	24
i	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
scale_i	28	32	36	40	44	48	52	56	64	72	80	88	96	104	112	

is horizontally subsampled by a factor of 2, whereas 4:4:4 is a special case, where no chroma subsampling actually takes place.

- **Nonlinear quantization.** Quantization in MPEG-2 is similar to that in MPEG-1. Its *step_size* is also determined by the product of $Q[i, j]$ and $scale$, where Q is one of the default quantization tables for intra- or inter- coding. Two types of scales are allowed. For the first, *scale* is the same as in MPEG-1, in which it is an integer in the range of [1, 31] and $scale_i = i$. For the second type, however, a nonlinear relationship exists—that is, $scale_i \not\equiv i$. The ith scale value can be looked up in Table 11.7.
- **More restricted slice structure.** MPEG-1 allows slices to cross macroblock row boundaries. As a result, an entire picture can be a single slice. MPEG-2 slices must start and end in the same macroblock row. In other words, the left edge of a picture always starts a new slice, and the longest slice in MPEG-2 can have only one row of macroblocks.
- **More flexible video formats.** According to the standard, MPEG-2 picture sizes can be as large as 16 k × 16 k pixels. In reality, MPEG-2 is used mainly to support various picture resolutions as defined by DVD, ATV, and HDTV.

Similar to H.261, H.263, and MPEG-1, MPEG-2 specifies only its bitstream syntax and the decoder. This leaves much room for future improvement, especially on the encoder side. The MPEG-2 video-stream syntax is more complex than that of MPEG-1, and good references for it can be found in [6,7].

11.4 MPEG-4

11.4.1 Overview of MPEG-4

MPEG-1 and -2 employ *frame-based* coding techniques, in which each rectangular video frame is divided into macroblocks and then blocks for compression. This is also known as *block-based* coding. Their main concern is high compression ratio and satisfactory quality of video under such compression techniques. MPEG-4 had a very different emphasis [8]. Besides compression, it pays great attention to user interactivity. This allows a larger number of users to create and communicate their multimedia presentations and applications on new infrastructures, such as the Internet, the World Wide Web (WWW), and mobile/wireless networks. MPEG-4 departs from its pre-

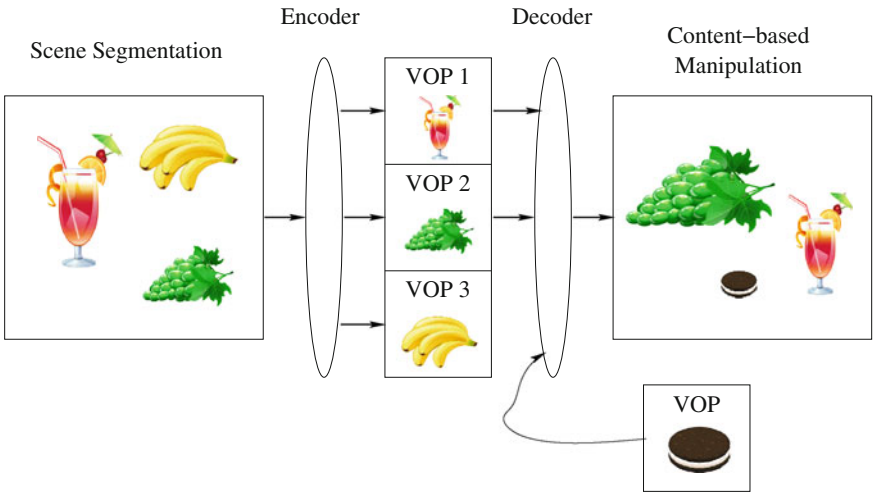

Fig. 11.11 Composition and manipulation of MPEG-4 videos (VOP = Video object plane)

decessors in adopting a new *object-based coding* approach—*media objects* are now entities for MPEG-4 coding. Media objects (also known as *audio and visual objects*) can be either *natural* or *synthetic*; that is to say, they may be captured by a video camera or created by computer programs.

Object-based coding not only has the potential of offering higher compression ratio but is also beneficial for digital video composition, manipulation, indexing, and retrieval. Figure 11.11 illustrates how MPEG-4 videos can be composed and manipulated by simple operations such as insertion/deletion, translation/rotation, scaling, and so on, on the visual objects.

MPEG-4 (version 1) was finalized in October 1998 and became an international standard in early 1999, referred to as ISO/IEC 14496 [9]. An improved version (version 2) was finalized in December 1999 and acquired International Standard status in 2000. Similar to the previous MPEG standards, its first five parts are Systems, Video, Audio, Conformance, and Software. This chapter will discuss the video compression issues in MPEG-4 Part 2 (formally ISO/IEC 14496-2).[1]

Originally targeted at low-bitrate communication (4.8–64 kbps for mobile applications and up to 2 Mbps for other applications), the bitrate for MPEG-4 video now covers a large range, between 5 kbps and 10 Mbps.

As the *Reference Models* in Fig. 11.12a shows, an MPEG-1 system simply delivers audio and video data from its storage and does not allow any user interactivity. MPEG-2 added an Interaction component (indicated by dashed lines in Fig. 11.12a) and thus permits limited user interactions in applications such as networked video and Interactive TV. MPEG-4 (Fig. 11.12b) is an entirely new standard for (a) composing

[1] MPEG-4 Part 10 which is identical to ITU-T H.264 AVC will be discussed in Chap. 12.

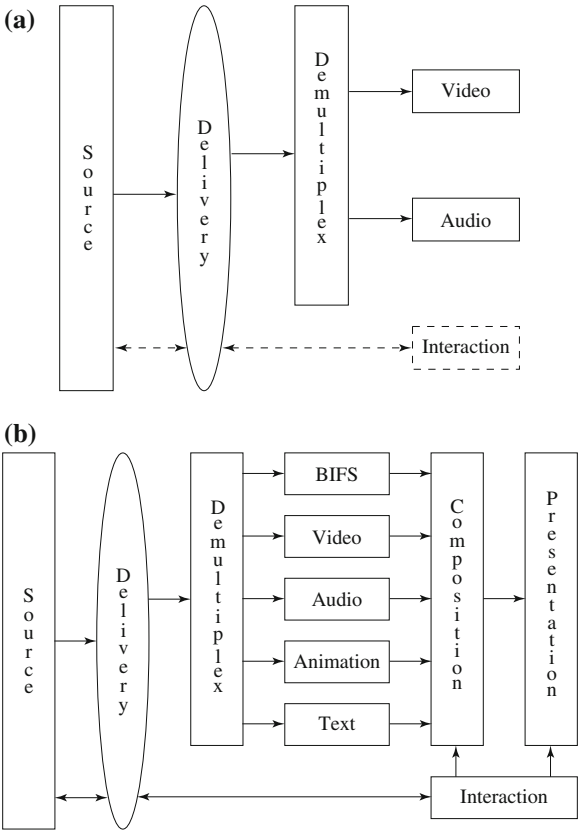

Fig. 11.12 Comparison of interactivity in MPEG standards: **a** reference models in MPEG-1 and 2 (interaction in *dashed lines* supported only by MPEG-2); **b** MPEG-4 reference model

media objects to create desirable audiovisual scenes, (b) multiplexing and synchronizing the bitstreams for these media data entities so that they can be transmitted with guaranteed Quality of Service (QoS), and (c) interacting with the audiovisual scene at the receiving end. MPEG-4 provides a toolbox of advanced coding modules and algorithms for audio and video compression.

MPEG-4 defines *BInary Format for Scenes* (BIFS) [10] that facilitates the composition of media objects into a scene. BIFS is often represented by a scene graph, in which the nodes describe audiovisual primitives and their attributes and the graph structure enables a description of spatial and temporal relationships of objects in the scene. BIFS is an enhancement of Virtual Reality Modeling Language (VRML). In particular, it emphasizes timing and synchronization of objects, which were lacking in the original VRML design. In addition to BIFS, MPEG-4 (version 2) provides a programming environment, *MPEG-J* [11], in which Java applications (called *MPEGlets*) can access Java packages and APIs so as to enhance end users' interactivity.

The hierarchical structure of MPEG-4 visual bitstreams is very different from that of MPEG-1 and 2 in that it is very much video-object-oriented. Figure 11.13 illustrates five levels of the hierarchical description of a scene in MPEG-4 visual

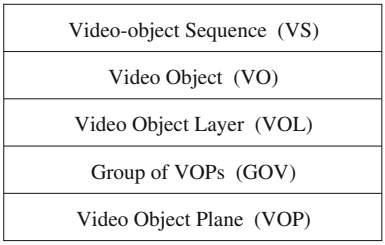

Fig. 11.13 Video-object-oriented hierarchical description of a scene in MPEG-4 visual bitstreams

bitstreams. In general, each *Video-object Sequence* (VS) will have one or more *Video Objects* (VOs), each VO will have one or more *Video Object Layers* (VOLs), and so on. Syntactically, all five levels have a unique start code in the bitstream, to enable random access.

1. **Video-object Sequence (VS).** VS delivers the complete MPEG-4 visual scene, which may contain 2D or 3D natural or synthetic objects.
2. **Video Object (VO).** VO is a particular object in the scene, which can be of arbitrary (nonrectangular) shape, corresponding to an object or background of the scene.
3. **Video Object Layer (VOL).** VOL facilitates a way to support (multilayered) scalable coding. A VO can have multiple VOLs under scalable coding or a single VOL under nonscalable coding. As a special case, MPEG-4 also supports a special type of VOL with a shorter header. This provides bitstream compatibility with the baseline H.263 [12].
4. **Group of Video Object Planes (GOV).** GOV groups video object planes. It is an optional level.
5. **Video Object Plane (VOP).** A VOP is a snapshot of a VO at a particular moment, reflecting the VO's shape, texture, and motion parameters at that instant. In general, a VOP is an image of arbitrary shape. A degenerate case in MPEG-4 video coding occurs when the entire rectangular video frame is treated as a VOP. In this case, it is equivalent to MPEG-1 and 2. MPEG-4 allows overlapped VOPs—that is, a VOP can partially occlude another VOP in a scene.

11.4.2 Video Object-Based Coding in MPEG-4

MPEG-4 encodes/decodes each VOP separately (instead of considering the whole frame). Hence, its video-object-based coding is also known as *VOP-based coding*. Our discussion will start with coding for natural objects (more details can be found in [13,14]). Section 11.4.3 describes synthetic object coding.

VOP-Based Coding Versus Frame-Based Coding

MPEG-1 and 2 do not support the VOP concept; hence, their coding method is referred to as *frame-based*. Since each frame is divided into many macroblocks from which motion-compensation-based coding is conducted, it is also known as *block-based coding*. Figure 11.14a shows three frames from a video sequence with a vehicle moving toward the left and a pedestrian walking in the opposite direction. Figure 11.14b shows the typical block-based coding in which the motion vector (**MV**) is obtained for one of the macroblocks.

MPEG-1 and 2 video coding are concerned only with *compression ratio* and do not consider the existence of visual objects. Therefore, the motion vectors generated may be inconsistent with object-level motion and would not be useful for object-based video analysis and indexing.

Figure 11.14c illustrates a possible example in which both potential matches yield small prediction errors. If Potential Match 2 yields a (slightly) smaller prediction error than Potential Match 1, **MV2** will be chosen as the motion vector for the macroblock in the block-based coding approach, although only **MV1** is consistent with the vehicle's direction of motion.

Object-based coding in MPEG-4 is aimed at solving this problem, in addition to improving compression. Figure 11.14d shows that each VOP is of arbitrary shape and will ideally obtain a unique motion vector consistent with the object's motion.

MPEG-4 VOP-based coding also employs the motion compensation technique. An Intraframe-coded VOP is called an *I-VOP*. Interframe-coded VOPs are called *P-VOP*s if only forward prediction is employed or *B-VOP*s if bidirectional predictions are employed. The new difficulty here is that the VOPs may have arbitrary shapes. Therefore, in addition to their texture, their shape information must now be coded.

It is worth noting that *texture* here actually refers to the visual content, that is, the graylevel and chroma values of the pixels in the VOP. MPEG-1 and 2 do not code shape information, since all frames are rectangular, but they do code the values of the pixels in the frame. In MPEG-1 and 2, this coding was not explicitly referred to as texture coding. The term "texture" comes from computer graphics and shows how this discipline has entered the video coding world with MPEG-4.

Below, we start with a discussion of motion-compensation-based coding for VOPs, followed by introductions to *texture coding*, *shape coding*, *static texture coding*, *sprite coding*, and *global motion compensation*.

Motion Compensation

This section addresses issues of VOP-based motion compensation in MPEG-4. Since I-VOP coding is relatively straightforward, our discussions will concentrate on coding for P-VOP and/or B-VOP unless I-VOP is explicitly mentioned.

As before, motion-compensation-based VOP coding in MPEG-4 again involves three steps: motion estimation, motion-compensation-based prediction, and coding of the prediction error. To facilitate motion compensation, each VOP is divided

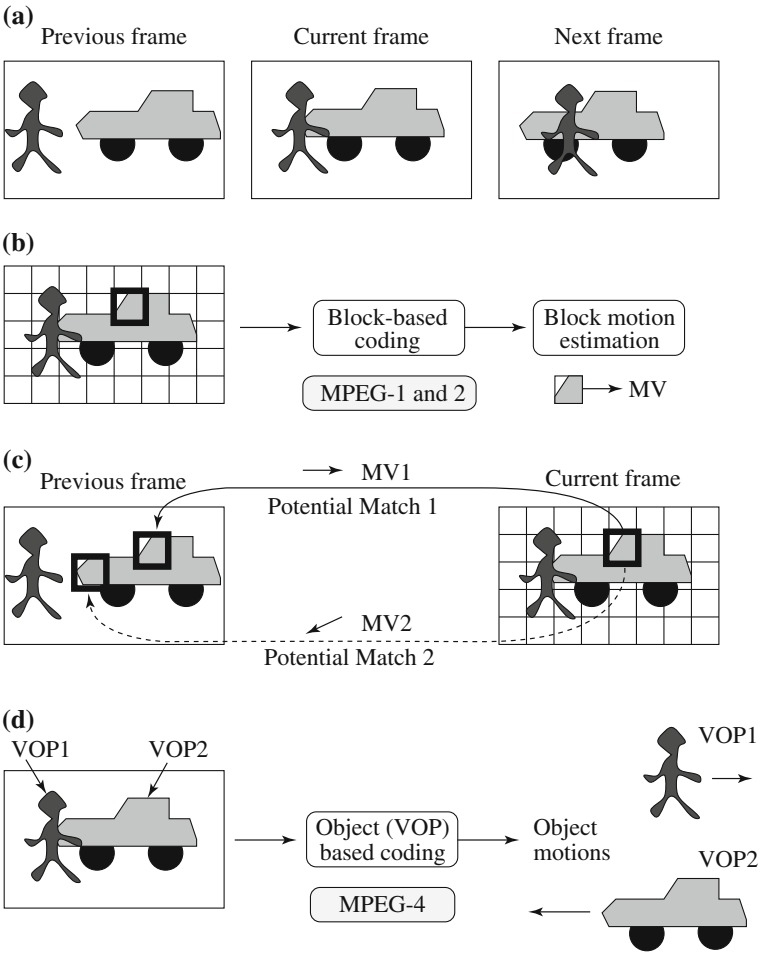

Fig. 11.14 Comparison between block-based coding and object-based coding: **a** a video sequence; **b** MPEG-1 and 2 block-based coding; **c** two potential matches in MPEG-1 and 2; **d** object-based coding in MPEG-4

into many macroblocks, as in previous frame-based methods. Macroblocks are by default 16×16 in luminance images and 8×8 in chrominance images and are treated specially when they straddle the boundary of an arbitrarily shaped VOP.

MPEG-4 defines a rectangular *bounding box* for each VOP. Its left and top bounds are the left and top bounds of the VOP, which in turn specify the shifted origin for the VOP from the original (0, 0) for the video frame in the absolute (frame) coordinate system (see Fig. 11.15). Both horizontal and vertical dimensions of the bounding box must be multiples of 16 in the luminance image. Therefore, the box is usually slightly larger than a conventional bounding box.

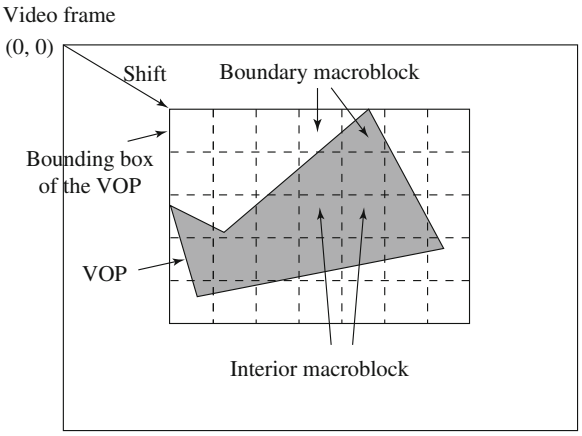

Fig. 11.15 Bounding box and boundary macroblocks of VOP

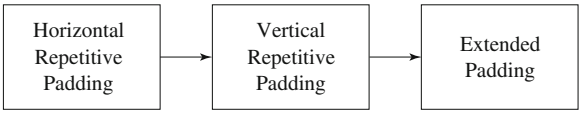

Fig. 11.16 A sequence of paddings for reference VOPs in MPEG-4

Macroblocks entirely within the VOP are referred to as *interior macroblocks*. As is apparent from Fig. 11.15, many of the macroblocks straddle the boundary of the VOP and are called *boundary macroblocks*.

Motion compensation for interior macroblocks is carried out in the same manner as in MPEG-1 and 2. However, boundary macroblocks could be difficult to match in motion estimation, since VOPs often have arbitrary (nonrectangular) shape, and their shape may change from one instant in the video to another. To help match every pixel in the target VOP and meet the mandatory requirement of rectangular blocks in transform coding (e.g., DCT), a preprocessing step of *padding* is applied to the reference VOPs prior to motion estimation.

Only pixels within the VOP of the current (target) VOP are considered for matching in motion compensation, and padding takes place only in the reference VOPs.

For quality, some better extrapolation method than padding could have been developed. Padding was adopted in MPEG-4 largely due to its simplicity and speed.

The first two steps of motion compensation are: padding, and motion vector coding.

Padding. For all boundary macroblocks in the reference VOP, *horizontal repetitive padding* is invoked first, followed by *vertical repetitive padding* (Fig. 11.16). Afterward, for all *exterior macroblocks* that are outside of the VOP but adjacent to one or more boundary macroblocks, *extended padding* is applied.

The horizontal repetitive padding algorithm examines each row in the boundary macroblocks in the reference VOP. Each boundary pixel is replicated to the left and/or

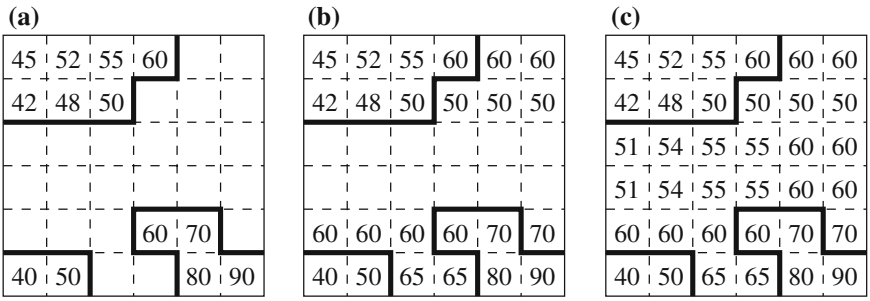

Fig. 11.17 An example of repetitive padding in a boundary macroblock of a reference VOP: **a** original pixels within the VOP; **b** after horizontal repetitive padding; **c** followed by vertical repetitive padding

right to fill in the values for the interval of pixels outside the VOP in the macroblock. If the interval is bounded by two boundary pixels, their average is adopted.

Algorithm 11.1 (Horizontal Repetitive Padding).

begin

for all rows in Boundary macroblocks in the Reference VOP
if ∃ (boundary pixel) in the row
for all $interval$ outside of VOP
if $interval$ is bounded by only one boundary pixel b
assign the value of b to all pixels in $interval$
else // $interval$ is bounded by two boundary pixels b_1 and b_2
assign the value of $(b_1 + b_2)/2$ to all pixels in $interval$

end

The subsequent vertical repetitive padding algorithm works similarly. It examines each column, and the newly padded pixels by the preceding horizontal padding process are treated as pixels inside the VOP for the purpose of this vertical padding.

Example 11.1

Figure 11.17 illustrates an example of repetitive padding in a boundary macroblock of a reference VOP. Figure 11.17a shows the luminance (or chrominance) intensity values of pixels in the VOP, with the VOP's boundary shown as darkened lines. For simplicity, the macroblock's resolution is reduced to 6×6 in this example, although its actual macroblock size is 16×16 in luminance images and 8×8 in chrominance images.

1. **Horizontal repetitive padding** (Fig. 11.17b)
   Row 0 The rightmost pixel of the VOP is the only boundary pixel. Its intensity value, 60, is used repetitively as the value of the pixels outside the VOP.
   Row 1 Similarly, the rightmost pixel of the VOP is the only boundary pixel. Its intensity value, 50, is used repetitively as the pixel value outside of the VOP.

Rows 2 and 3 No horizontal padding, since no boundary pixels exist.
Row 4 There exist two intervals outside the VOP, each bounded by a single boundary pixel. Their intensity values, 60 and 70, are used as the pixel values of the two intervals, respectively.
Row 5 A single interval outside the VOP is bounded by a pair of boundary pixels of the VOP. The average of their intensity values, $(50 + 80)/2 = 65$, is used repetitively as the value of the pixels between them.

2. **Vertical repetitive padding** (Fig. 11.17c)
Column 0 A single interval is bounded by a pair of boundary pixels of the VOP. One is 42 in the VOP; the other is 60, which just arose from horizontal padding. The average of their intensity values, $(42 + 60)/2 = 51$, is repetitively used as the value of the pixels between them.
Columns 1, 2, 3, 4 and 5 These columns are padded similarly to Column 0.

Extended Padding. Macroblocks entirely outside the VOP are *exterior* macroblocks. Exterior macroblocks immediately next to boundary macroblocks are filled by replicating the values of the border pixels of the boundary macroblock. We note that boundary macroblocks are by now fully padded, so all their horizontal and vertical border pixels have defined values. If an exterior macroblock has more than one boundary macroblock as its immediate neighbor, the boundary macroblock to use for extended padding follows a priority list: left, top, right, and bottom.

Later versions of MPEG-4 allow some average values of these macroblocks to be used. This extended padding process can be repeated to fill in all exterior macroblocks within the rectangular bounding box of the VOP.

Motion Vector Coding. Each macroblock from the target VOP will find a best matching macroblock from the reference VOP through the following motion estimation procedure:

Let $C(x+k, y+l)$ be pixels of the macroblock in the target VOP, and $R(x+i+k, y+j+l)$ be pixels of the macroblock in the reference VOP. Similar to MAD in Eq. (10.1), a *Sum of Absolute Difference* (SAD) for measuring the difference between the two macroblocks can be defined as

$$SAD(i,j) = \sum_{k=0}^{N-1}\sum_{l=0}^{N-1} |C(x+k, y+l) - R(x+i+k, y+j+l)| \cdot Map(x+k, y+l)$$

where N is the size of the macroblock. $Map(p,q) = 1$ when $C(p,q)$ is a pixel within the target VOP; otherwise, $Map(p,q) = 0$. The vector (i,j) that yields the minimum SAD is adopted as the motion vector $\mathbf{MV}(u,v)$:

$$(u,v) = \{\, (i,j) \mid SAD(i,j) \text{ is minimum}, \quad i \in [-p,p], \quad j \in [-p,p] \,\} \tag{11.3}$$

where p is the maximal allowable magnitude for u and v.

For motion compensation, the *motion vector* **MV** is coded. As in H.263 (see Fig. 11.11), the motion vector of the target macroblock is not simply taken as the **MV**. Instead, **MV** is predicted from three neighboring macroblocks. The prediction error for the motion vector is then variable-length coded.

Following are some of the advanced motion compensation techniques adopted similar to the ones in H.263 (see Sect. 10.5).

- Four motion vectors (each from an 8×8 block) can be generated for each macroblock in the luminance component of a VOP.
- Motion vectors can have subpixel precision. At half-pixel precision, the range of motion vectors is $[-2{,}048, 2{,}047]$. MPEG-4 also allows quarter-pixel precision in the luminance component of a VOP.
- Unrestricted motion vectors are allowed: **MV** can point beyond the boundaries of the reference VOP. When a pixel outside the VOP is referenced, its value is still defined, due to padding.

Texture Coding

Texture refers to gray level (or chroma) variations and/or patterns in the VOP. Texture coding in MPEG-4 can be based either on DCT or *shape-Adaptive DCT* (SA-DCT).

Texture Coding Based on DCT. In I-VOP, the gray (or chroma) values of the pixels in each macroblock of the VOP are directly coded, using the DCT followed by VLC, which is similar to what is done in JPEG for still pictures. P-VOP and B-VOP use motion-compensation-based coding; hence, it is the prediction error that is sent to DCT and VLC. The following discussion will be focused on motion-compensation-based texture coding for P-VOP and B-VOP.

Coding for Interior macroblocks, each 16×16 in the luminance VOP and 8×8 in the chrominance VOP, is similar to the conventional motion-compensation-based coding in H.261, H.263, and MPEG-1 and 2. Prediction errors from the six 8×8 blocks of each macroblock are obtained after the conventional motion estimation step. These are sent to a DCT routine to obtain six 8×8 blocks of DCT coefficients.

For boundary macroblocks, areas outside the VOP in the reference VOP are padded using repetitive padding, as described above. After motion compensation, texture prediction errors within the target VOP are obtained. For portions of the boundary macroblocks in the target VOP outside the VOP, zeros are padded to the block sent to DCT, since ideally, prediction errors would be near zero inside the VOP. Whereas repetitive padding and extended padding were for better matching in motion compensation, this additional zero padding is for better DCT results in texture coding.

The quantization $step\_size$ for the DC component is 8. For the AC coefficients, one of the following two methods can be employed:

- The H.263 method, in which all coefficients receive the same quantizer controlled by a single parameter, and different macroblocks can have different quantizers.
- The MPEG-2 method, in which DCT coefficients in the same macroblock can have different quantizers and are further controlled by the $step\_size$ parameter.

Shape-Adaptive DCT (SA-DCT)-Based Coding for Boundary Macroblocks SA-DCT [15] is another texture coding method for boundary macroblocks. Due to its effectiveness, SA-DCT has been adopted for coding boundary macroblocks in MPEG-4 version 2.

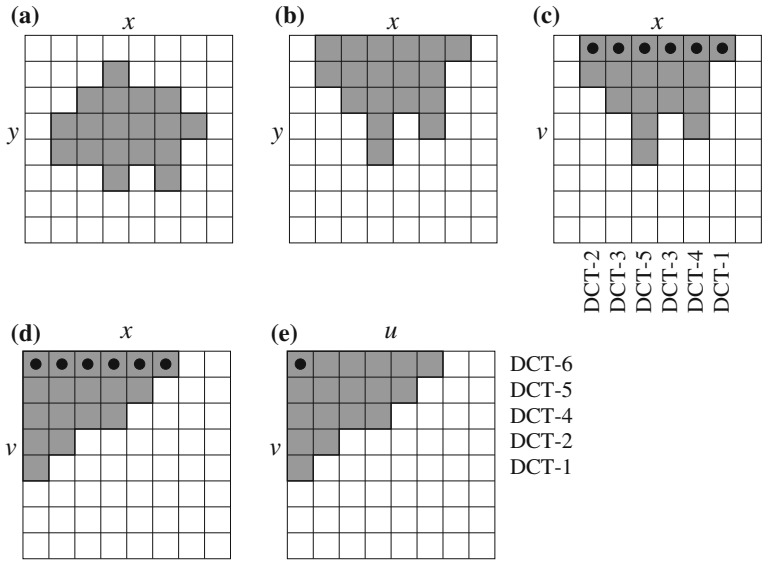

Fig. 11.18 Texture coding for boundary macroblocks using the shape-adaptive DCT (SA-DCT). **a** $f(x, y)$, **b** $f'(x, y)$, **c** $f'(x, v)$, **d** $f''(x, v)$, **e** $G(u, v)$

1D DCT-N is a variation of the 1D DCT described earlier [Eqs. (8.19) and (8.20)], in that N elements are used in the transform instead of a fixed $N = 8$ (For short, we will denote the 1D DCT-N transform by DCT-N in this section).

Equations (11.4) and (11.5) describe the DCT-N transform and its inverse, IDCT-N.

1D Discrete Cosine Transform-N (DCT-N)

$$F(u) = \sqrt{\frac{2}{N}}\, C(u) \sum_{i=0}^{N-1} \cos \frac{(2i+1)u\pi}{2N}\, f(i) \tag{11.4}$$

1D Inverse Discrete Cosine Transform-N (IDCT-N)

$$\tilde{f}(i) = \sum_{u=0}^{N-1} \sqrt{\frac{2}{N}}\, C(u) \cos \frac{(2i+1)u\pi}{2N}\, F(u) \tag{11.5}$$

where $i = 0, 1, \ldots, N-1$, $u = 0, 1, \ldots, N-1$, and

$$C(u) = \begin{cases} \frac{\sqrt{2}}{2} & \text{if } u = 0, \\ 1 & \text{otherwise} \end{cases}$$

SA-DCT is a 2D DCT and is computed as a separable 2D transform in two iterations of DCT-N. Figure 11.18 illustrates the process of texture coding for boundary macroblocks using SA-DCT. The transform is applied to each of the 8×8 blocks in the boundary macroblock.

Figure 11.18a shows one of the 8×8 blocks of a boundary macroblock, where pixels inside the macroblock, denoted $f(x, y)$, are shown in gray. The gray pixels are first shifted upward to obtain $f'(x, y)$, as Fig. 11.18b shows. In the first iteration, DCT-N is applied to each column of $f'(x, y)$, with N determined by the number of gray pixels in the column. Hence, we use DCT-2, DCT-3, DCT-5, and so on. The resulting DCT-N coefficients are denoted by $F'(x, v)$, as Fig. 11.18c shows, where the dark dots indicate the DC coefficients of the DCT-Ns. The elements of $F'(x, v)$ are then shifted to the left to obtain $F''(x, v)$ in Fig. 11.18d.

In the second iteration, DCT-N is applied to each row of $F''(x, v)$ to obtain $G(u, v)$ (Fig. 11.18e), in which the single dark dot indicates the DC coefficient $G(0, 0)$ of the 2D SA-DCT.

Some coding considerations:

- The total number of DCT coefficients in $G(u, v)$ is equal to the number of gray pixels inside the 8×8 block of the boundary macroblock, which is less than 8×8. Hence, the method is *shape adaptive* and is more efficient to compute.
- At decoding time, since the array elements must be shifted back properly after each iteration of IDCT-Ns, a binary mask of the original shape is required to decode the texture information coded by SA-DCT. The binary mask is the same as the *binary alpha map* described below.

Shape Coding

Unlike in MPEG-1 and 2, MPEG-4 must code the shape of the VOP, since shape is one of the intrinsic features of visual objects.

MPEG-4 supports two types of shape information: *binary* and *grayscale*. Binary shape information can be in the form of a binary map (also known as a *binary alpha map*) that is of the same size as the VOP's rectangular bounding box. A value of 1 (opaque) or 0 (transparent) in the bitmap indicates whether the pixel is inside or outside the VOP. Alternatively, the grayscale shape information actually refers to the shape's *transparency*, with gray values ranging from 0 (transparent) to 255 (opaque).

Binary Shape Coding. To encode the binary alpha map more efficiently, the map is divided into 16×16 blocks, also known as *Binary Alpha Blocks* (BAB). If a BAB is entirely opaque or transparent, it is easy to code, and no special technique of shape coding is necessary. It is the boundary BABs that contain the contour and hence the shape information for the VOP. They are the subject of binary shape coding.

Various contour-based and bitmap-based (or area-based) algorithms have been studied and compared for coding boundary BABs. Two of the finalists were both bitmap-based. One was the *Modified Modified READ* (MMR) *algorithm*, which was also an optional enhancement in the fax Group 3 (G3) standard [16] and the mandatory compression method in the Group 4 (G4) standard [17]. The other finalist was *Context-based Arithmetic Encoding* (CAE), which was initially developed for JBIG [18]. CAE was finally chosen as the binary shape-coding method for MPEG-4 because of its simplicity and compression efficiency.

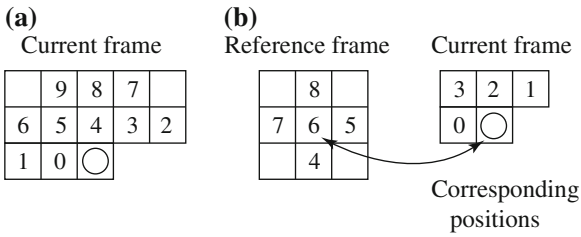

Fig. 11.19 Contexts in CAE for binary shape coding in MPEG-4. ◯ indicates the current pixel, and *digits* indicate the other pixels in the neighborhood: **a** intra-CAE; **b** inter-CAE

MMR is basically a series of simplifications of the *Relative Element Address Designate* (READ) algorithm. The basic idea behind the READ algorithm is to code the current line relative to the pixel locations in the previously coded line. The algorithm starts by identifying five pixel locations in the previous and current lines:

- a_0: the last pixel value known to both the encoder and decoder
- a_1: the transition pixel to the right of a_0
- a_2: the second transition pixel to the right of a_0
- b_1: the first transition pixel whose color is opposite to a_0 in the previously coded line
- b_2: the first transition pixel to the right of b_1 on the previously coded line

READ works by examining the relative positions of these pixels. At any time, both the encoder and decoder know the position of a_0, b_1, and b_2, while the positions a_1 and a_2 are known only in the encoder.

Three coding modes are used:

- If the run lengths on the previous and the current lines are similar, the distance between a_1 and b_1 should be much smaller than the distance between a_0 and a_1. Thus, the *vertical mode* encodes the current run length as $a_1 - b_1$.
- If the previous line has no similar run length, the current run length is coded using one-dimensional run-length coding. This is called the *horizontal mode*.
- If $a_0 \leq b_1 < b_2 < a_1$, we can simply transmit a codeword indicating it is in *pass mode* and advance a_0 to the position under b_2, and continue the coding process.

Some simplifications can be made to the READ algorithm for practical implementation. For example, if $\|a_1 - b_1\| < 3$, then it is enough to indicate that we can apply the vertical mode. Also, to prevent error propagation, a k-factor is defined, such that every k lines must contain at least one line coded using conventional run-length coding. These modifications constitute the *Modified READ* algorithm used in the G3 standard. The Modified Modified READ (MMR) algorithm simply removes the restrictions imposed by the k-factor.

For Context-based Arithmetic Encoding, Fig. 11.19 illustrates the "context" for a pixel in the boundary BAB. In intra-CAE mode, when only the target alpha map is involved (Fig. 11.19a), ten neighboring pixels (numbered from 0 to 9) in the same alpha map form the context. The ten binary numbers associated with these pixels can offer up to $2^{10} = 1{,}024$ possible contexts.

Now, it is apparent that certain contexts (e.g., all 1s or all 0s) appear more frequently than others. With some prior statistics, a probability table can be built to indicate the probability of occurrence for each of the 1,024 contexts.

Recall that Arithmetic Coding (Chap. 7) is capable of encoding a sequence of probabilistic symbols with a single number. Now, each pixel can look up the table to find a probability value for its context. CAE simply scans the 16×16 pixels in each BAB sequentially and applies Arithmetic coding to eventually derive a single floating-point number for the BAB.

Inter-CAE mode is a natural extension of intra-CAE: it involves both the target and reference alpha maps. For each boundary macroblock in the target frame, a process of motion estimation (in integer precision) and compensation is invoked first to locate the matching macroblock in the reference frame. This establishes the corresponding positions for each pixel in the boundary BAB.

Figure 11.19b shows the context of each pixel includes four neighboring pixels from the target alpha map and five pixels from the reference alpha map. According to its context, each pixel in the boundary BAB is assigned one of the $2^9 = 512$ probabilities. Afterward, the CAE algorithm is applied.

The 16×16 binary map originally contains 256 bits of information. Compressing it to a single floating number achieves a substantial saving.

The above CAE method is *lossless*! The MPEG-4 group also examined some simple lossy versions of the above shape-coding method. For example, the binary alpha map can be simply subsampled by a factor of 2 or 4 before arithmetic coding. The tradeoff is, of course, the deterioration of the shape.

Grayscale Shape Coding. The term *grayscale shape coding* in MPEG-4 could be misleading, because the true shape information is coded in the binary alpha map. Grayscale here is used to describe the *transparency* of the shape, not the texture!

In addition to the bitplanes for RGB frame buffers, raster graphics uses extra bitplanes for an *alpha map*, which can be used to describe the transparency of the graphical object. When the alpha map has more than one bitplane, multiple levels of transparency can be introduced—for example, 0 for transparent, 255 for opaque, and any number in between for various degrees of intermediate transparency. The term grayscale is used for transparency coding in MPEG-4 simply because the transparency number happens to be in the range of 0 to 255—the same as conventional 8-bit grayscale intensities.

Grayscale shape coding in MPEG-4 employs the same technique as in the texture coding described above. It uses the alpha map and block-based motion compensation and encodes prediction errors by DCT. The boundary macroblocks need padding, as before, since not all pixels are in the VOP.

Coding of the transparency information (grayscale shape coding) is lossy, as opposed to coding of the binary shape information, which is by default lossless.

Static Texture Coding

MPEG-4 uses wavelet coding for the texture of static objects. This is particularly applicable when the texture is used for mapping onto 3D surfaces.

As introduced in Chap. 8, wavelet coding can recursively decompose an image into *subbands* of multiple frequencies. The Embedded Zerotree Wavelet (EZW) algorithm [19] provides a compact representation by exploiting the potentially large number of insignificant coefficients in the subbands.

The coding of subbands in MPEG-4 static texture coding is conducted as follows:

- The subbands with the lowest frequency are coded using DPCM. Prediction of each coefficient is based on three neighbors.
- Coding of other subbands is based on a multiscale zerotree wavelet coding method.

The multiscale zerotree has a *parent–child relation* (PCR) *tree* for each coefficient in the lowest frequency subband. As a result, the location information of all coefficients is better tracked.

In addition to the original magnitude of the coefficients, the degree of quantization affects the data rate. If the magnitude of a coefficient is zero after quantization, it is considered insignificant. At first, a large quantizer is used; only the most significant coefficients are selected and subsequently coded using arithmetic coding. The difference between the quantized and the original coefficients is kept in residual subbands, which will be coded in the next iteration in which a smaller quantizer is employed. The process can continue for additional iterations; hence, it is very scalable.

Sprite Coding

Video photography often involves camera movements such as pan, tilt, zoom in/out, and so on. Often, the main objective is to track and examine foreground (moving) objects. Under these circumstances, the background can be treated as a static image. This creates a new VO type, the *sprite*—a graphic image that can freely move around within a larger graphic image or set of images.

To separate the foreground object from the background, we introduce the notion of a *sprite panorama*—a still image that describes the static background over a sequence of video frames. It can be generated using image "stitching" and warping techniques [20]. The large sprite panoramic image can be encoded and sent to the decoder only once, at the beginning of the video sequence. When the decoder receives separately coded foreground objects and parameters describing the camera movements thus far, it can efficiently reconstruct the scene.

Figure 11.20a shows a sprite that is a panoramic image stitched from a sequence of video frames. By combining the sprite background with the piper in the bluescreen image (Fig. 11.20b), the new video scene (Fig. 11.20c) can readily be decoded with the aid of the sprite code and the additional pan/tilt and zoom parameters. Clearly, foreground objects can either be from the original video scene or newly created to realize flexible object-based composition of MPEG-4 videos.

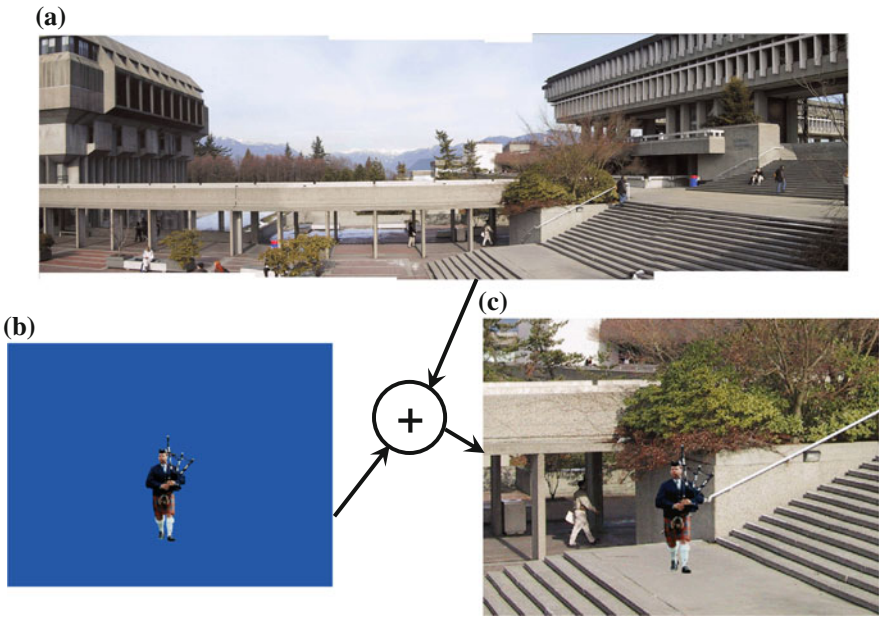

Fig. 11.20 Sprite coding: **a** the sprite panoramic image of the background; **b** the foreground object (in this case, a piper) in a *bluescreen image*; **c** the composed video scene. *Piper image courtesy of Simon Fraser University Pipe Band*

Global Motion Compensation

Common camera motions, such as pan, tilt, rotation, and zoom (so-called *global* motions, since they apply to every block), often cause rapid content change between successive video frames. Traditional block-based motion compensation would result in a large number of significant motion vectors. Also, these types of camera motions cannot all be described using the translational motion model employed by block-based motion compensation. *Global motion compensation (GMC)* is designed to solve this problem. There are four major components:

- **Global motion estimation.** Global motion estimation computes the motion of the current image with respect to the sprite. By "global" is meant overall change due to camera change—zooming in, panning to the side, and so on. It is computed by minimizing the sum of square differences between the sprite S and the global motion-compensated image I'.

$$E = \sum_{i=1}^{N} (S(x_i, y_i) - I'(x_i', y_i'))^2. \tag{11.6}$$

  The idea here is that if the background (possibly stitched) image is a sprite $S(x_i, y_i)$, we expect the new frame to consist mainly of the same background, altered by these

global camera motions. To further constrain the global motion estimation problem, the motion over the whole image is parameterized by a perspective motion model using eight parameters, defined as

$$x_i' = \frac{a_0 + a_1 x_i + a_2 y_i}{a_6 x_i + a_7 y_i + 1},$$
$$y_i' = \frac{a_3 + a_4 x_i + a_5 y_i}{a_6 x_i + a_7 y_i + 1}. \tag{11.7}$$

This resulting constrained minimization problem can be solved using a gradient-descent based method [21].

- **Warping and blending.** Once the motion parameters are computed, the background images are warped to align with respect to the sprite. The coordinates of the warped image are computed using Eq. (11.7). Afterward, the warped image is blended into the current sprite to produce the new sprite. This can be done using simple averaging or some form of weighted averaging.
- **Motion trajectory coding.** Instead of directly transmitting the motion parameters, we encode only the displacements of reference points. This is called *trajectory coding* [21]. Points at the corners of the VOP bounding box are used as reference points, and their corresponding points in the sprite are calculated. The difference between these two entities is coded and transmitted as differential motion vectors.
- **Choice of local motion compensation (LMC) or GMC.** Finally, a decision has to be made whether to use GMC or LMC. For this purpose, we can apply GMC to the moving background and LMC to the foreground. Heuristically (and with much detail skipped), if $SAD_{GMC} < SAD_{LMC}$, then use GMC to generate the predicted reference VOP. Otherwise, use LMC as before.

11.4.3 Synthetic Object Coding in MPEG-4

The number of objects in videos that are created by computer graphics and animation software is increasing. These are denoted *synthetic objects* and can often be presented together with natural objects and scenes in games, TV ads and programs, and animation or feature films.

In this section, we briefly discuss *2D mesh-based* and *3D model-based* coding and animation methods for synthetic objects. Beek, Petajan, and Ostermann [22] provide a more detailed survey of this subject.

2D Mesh Object Coding

A *2D mesh* is a tessellation (or partition) of a 2D planar region using polygonal patches. The vertices of the polygons are referred to as *nodes* of the mesh. The most

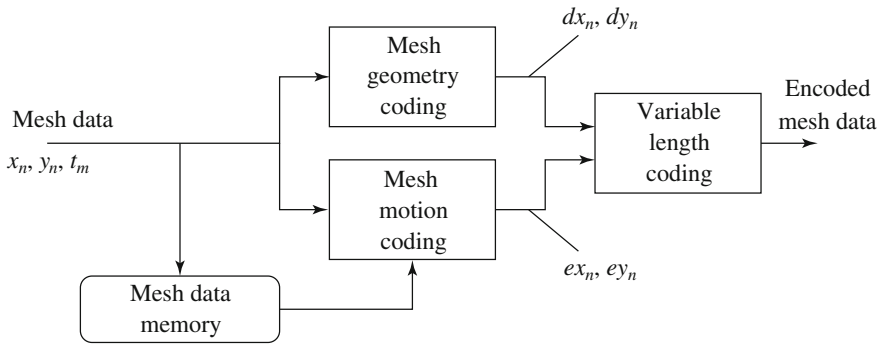

Fig. 11.21 2D Mesh object plane (MOP) encoding process

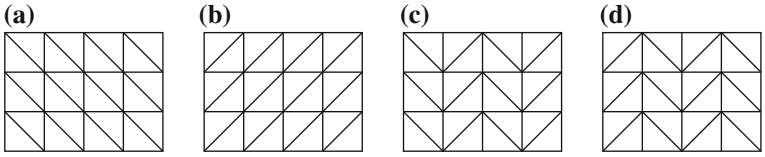

Fig. 11.22 Four types of uniform meshes: **a** type 0; **b** type 1; **c** type 2; **d** type3

popular meshes are *triangular meshes*, where all polygons are triangles. The MPEG-4 standard makes use of two types of 2D mesh: *uniform mesh* and *Delaunay mesh* [23]. Both are triangular meshes that can be used to model natural video objects as well as synthetic animated objects.

Since the triangulation structure (the edges between nodes) is known and can be readily regenerated by the decoder, it is not coded explicitly in the bitstream. Hence, 2D mesh object coding is compact. All coordinate values of the mesh are coded in half-pixel precision.

Each 2D mesh is treated as a *mesh object plane* (MOP). Figure 11.21 illustrates the encoding process for 2D MOPs. Coding can be divided into *geometry coding* and *motion coding*. As shown, the input data is the x and y coordinates of all the nodes and the triangles (t_m) in the mesh. The output data is the displacements (dx_n, dy_n) and the prediction errors of the motion (ex_n, ey_n), both of which are explained below.

2D Mesh Geometry Coding. MPEG-4 allows four types of uniform meshes with different triangulation structures. Figure 11.22 shows such meshes with 4×5 mesh nodes. Each uniform mesh can be specified by five parameters: the first two specify the number of nodes in each row and column, respectively; the next two specify the horizontal and vertical size of each rectangle (containing two triangles), respectively; and the last specifies the type of the uniform mesh.

Uniform meshes are simple and are especially good for representing 2D rectangular objects (e.g., the entire video frame). When used for objects of arbitrary shape,

they are applied to (overlaid on) the bounding boxes of the VOPs, which incurs some inefficiency.

A Delaunay mesh is a better object-based mesh representation for arbitrary-shaped 2D objects.

Definition 1: If $\mathcal{D}$ is a Delaunay triangulation, then any of its triangles $t_n = (P_i, P_j, P_k) \in \mathcal{D}$ satisfies the property that the circumcircle of t_n does not contain in its interior any other node point P_l.

A Delaunay mesh for a video object can be obtained in the following steps:

1. **Select boundary nodes of the mesh.** A polygon is used to approximate the boundary of the object. The polygon vertices are the *boundary nodes* of the Delaunay mesh. A possible heuristic is to select boundary points with high curvatures as boundary nodes.
2. **Choose interior nodes.** Feature points within the object's boundary such as edge points or corners, can be chosen as interior nodes for the mesh.
3. **Perform Delaunay triangulation.** A *constrained Delaunay triangulation* is performed on the boundary and interior nodes, with the polygonal boundary used as a constraint. The triangulation will use line segments connecting consecutive boundary nodes as edges and form triangles only within the boundary.

Constrained Delaunay Triangulation. Interior edges are first added to form new triangles. The algorithm will examine each interior edge to make sure it is *locally Delaunay*. Given two triangles (P_i, P_j, P_k) and (P_j, P_k, P_l) sharing an edge $\overline{jk}$, if (P_i, P_j, P_k) contains P_l or (P_j, P_k, P_l) contains P_i in the interior of its circumcircle, then $\overline{jk}$ is not locally Delaunay and will be replaced by a new edge $\overline{il}$.

If P_l falls exactly on the circumcircle of (P_i, P_j, P_k) (and accordingly, P_i also falls exactly on the circumcircle of (P_j, P_k, P_l)), then $\overline{jk}$ will be viewed as locally Delaunay only if P_i or P_l has the largest x coordinate among the four nodes.

Figure 11.23a, b shows the set of Delaunay mesh nodes and the result of the constrained Delaunay triangulation. If the total number of nodes is N, and $N = N_b + N_i$ where N_b and N_i denote the number of boundary nodes and interior nodes, respectively, then the total number of triangles in the Delaunay mesh is $N_b + 2N_i - 2$. In the above figure, this sum is $8 + 2 \times 6 - 2 = 18$.

Unlike a uniform mesh, the node locations in a Delaunay mesh are irregular; hence, they must be coded. By convention in MPEG-4, the location (x_0, y_0) of the top left boundary node[2] is coded first, followed by the other boundary points counterclockwise (see Fig. 11.23a) or clockwise. Afterwards, the locations of the interior nodes are coded in any order.

Except for the first location (x_0, y_0), all subsequent coordinates are coded differentially—that is, for $n \geq 1$,

$$dx_n = x_n - x_{n-1}, \qquad dy_n = y_n - y_{n-1}, \tag{11.8}$$

[2] The top left boundary node is defined as the one that has the minimum $x + y$ coordinate value. If more than one boundary node has the same $x + y$, the one with the minimum y is chosen.

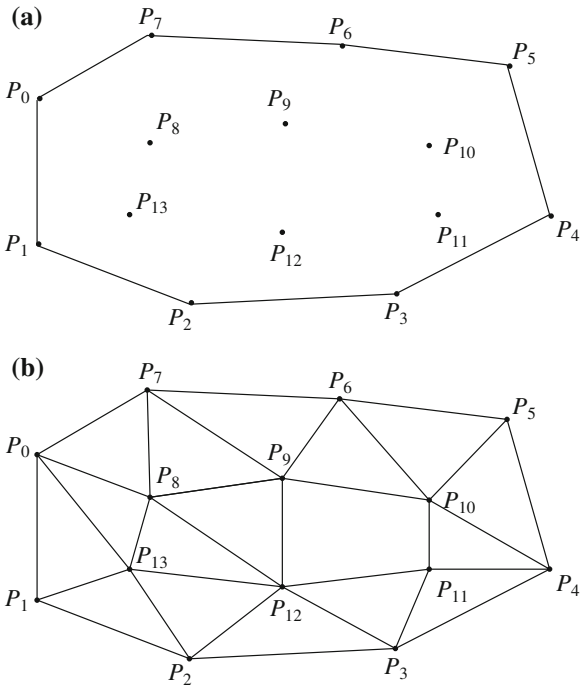

Fig. 11.23 Delaunay mesh: **a** boundary nodes (P_0 to P_7) and interior nodes (P_8 to P_{13}); **b** triangular mesh obtained by constrained Delaunay triangulation

and afterwards, dx_n, dy_n are variable-length coded.

2D Mesh Motion Coding. The motion of each MOP triangle in either a uniform or Delaunay mesh is described by the motion vectors of its three vertex nodes. A new mesh structure can be created only in the intraframe, and its triangular topology will not alter in the subsequent interframes. This enforces one-to-one mapping in 2D mesh motion estimation.

For any MOP triangle (P_i, P_j, P_k), if the motion vectors for P_i and P_j are known to be $\mathbf{MV_i}$ and $\mathbf{MV_j}$, then a prediction $\mathbf{Pred_k}$ will be made for the motion vector of P_k, rounded to a half-pixel precision:

$$\mathbf{Pred_k} = 0.5 \cdot (\mathbf{MV_i} + \mathbf{MV_j}). \tag{11.9}$$

The prediction error $\mathbf{e_k}$ is coded as

$$\mathbf{e_k} = \mathbf{MV_k} - \mathbf{Pred_k}. \tag{11.10}$$

Once the three motion vectors of the first MOP triangle t_0 are coded, at least one neighboring MOP triangle will share an edge with t_0, and the motion vector for its third vertex node can be coded, and so on.

The estimation of motion vectors will start at the *initial triangle* t_0, which is the triangle that contains the top left boundary node and the boundary node next to it, clockwise. Motion vectors for all other nodes in the MOP are coded differentially,

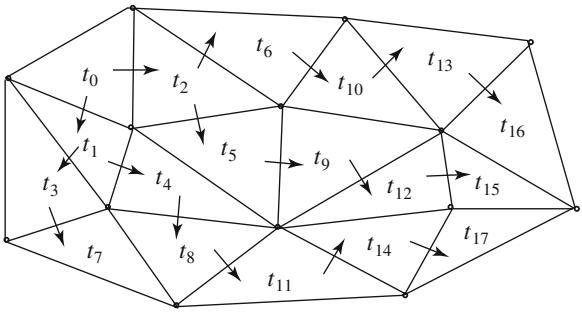

Fig. 11.24 A breadth-first order of MOP triangles for 2D mesh motion coding

according to Eq. (11.10). A breadth-first order is established for traversing the MOP triangles in the 2D mesh motion coding process.

Figure 11.24 shows how a spanning tree can be generated to obtain the breadth-first order of the triangles. As shown, the initial triangle t_0 has two neighboring triangles t_1 and t_2, which are not visited yet. They become child nodes of t_0 in the spanning tree.

Triangles t_1 and t_2, in turn, have their unvisited neighboring triangles (and hence child nodes) t_3, t_4, and t_5, t_6, respectively. The traverse order so far is t_0, t_1, t_2, t_3, t_4, t_5, in a breadth-first fashion. One level down the spanning tree, t_3 has only one child node t_7, since the other neighbor t_1 is already visited; t_4 has only one child node t_8; and so on.

2D Object Animation. The above mesh motion coding established a one-to-one mapping between the mesh triangles in the reference MOP and the target MOP. It generated motion vectors for all node points in the 2D mesh. Mesh-based texture mapping is now used to generate the texture for the new animated surface by warping [20] the texture of each triangle in the reference MOP onto the corresponding triangle in the target MOP. This facilitates the animation of 2D synthetic video objects.

For triangular meshes, a common mapping function for the warping is the *affine transform*, since it maps a line to a line and can guarantee that a triangle is mapped to a triangle. It will be shown below that given the six vertices of the two matching triangles, the parameters for the affine transform can be obtained, so that the transform can be applied to all points within the target triangle for texture mapping.

Given a point $\mathbf{P} = (x, y)$ on a 2D plane, a *linear transform* can be specified, such that

$$[x' \;\; y'] = [x \;\; y] \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \tag{11.11}$$

A transform T is linear if $T(\alpha\mathbf{X} + \beta\mathbf{Y}) = \alpha T(\mathbf{X}) + \beta T(\mathbf{Y})$, where α and β are scalars. The above linear transform is suitable for geometric operations such as rotation and scaling but not for translation, since addition of a constant vector is not possible.

Definition 2: A transform A is an *affine transform* if and only if there exists a vector $\mathbf{C}$ and a linear transform T such that $A(\mathbf{X}) = T(\mathbf{X}) + \mathbf{C}$.

If the point (x, y) is represented as $[x, y, 1]$ in the homogeneous coordinate system commonly used in graphics [24], then an *affine transform* that transforms $[x, y, 1]$ to $[x', y', 1]$ is defined as:

$$[x' \;\; y' \;\; 1] = [x \;\; y \;\; 1] \begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & 1 \end{bmatrix}. \tag{11.12}$$

It realizes the following mapping:

$$x' = a_{11}x + a_{21}y + a_{31} \tag{11.13}$$

$$y' = a_{12}x + a_{22}y + a_{32} \tag{11.14}$$

and has at most 6 degrees of freedom represented by the parameters a_{11}, a_{21}, a_{31}, a_{12}, a_{22}, and a_{32}.

The following 3×3 matrices are the affine transforms for translating by (T_x, T_y), rotating counterclockwise by θ, and scaling by factors S_x and S_y:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{bmatrix}, \quad \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The following are the affine transforms for a shear along the x-axis and y-axis, respectively:

$$\begin{bmatrix} 1 & 0 & 0 \\ H_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \begin{bmatrix} 1 & H_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where H_x and H_y are constants determining the degree of shear.

The above simple affine transforms can be combined (by matrix multiplications) to yield composite affine transforms—for example, for a translation followed by a rotation, or a shear followed by other transforms.

It can be proven (see Exercise 11.6) that any composite transform thus generated will have exactly the same matrix form and will have at most 6 degrees of freedom, specified by $a_{11}, a_{21}, a_{31}, a_{12}, a_{22}, a_{32}$.

If the triangle in the target MOP is

$$(\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2) = ((x_0, y_0), (x_1, y_1), (x_2, y_2))$$

and the matching triangle in the reference MOP is

$$(\mathbf{P}'_0, \mathbf{P}'_1, \mathbf{P}'_2) = ((x'_0, y'_0), (x'_1, y'_1), (x'_2, y'_2)),$$

then the mapping between the two triangles can be uniquely defined by the following:

$$\begin{bmatrix} x'_0 & y'_0 & 1 \\ x'_1 & y'_1 & 1 \\ x'_2 & y'_2 & 1 \end{bmatrix} = \begin{bmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & 1 \end{bmatrix} \tag{11.15}$$

Equation (11.15) contains six linear equations (three for x's and three for y's) required to resolve the six unknown coefficients $a_{11}, a_{21}, a_{31}, a_{12}, a_{22}, a_{32}$. Let Eq. (11.15) be stated as $\mathbf{X}' = \mathbf{X}A$. Then it is known that $A = \mathbf{X}^{-1}\mathbf{X}'$, with inverse

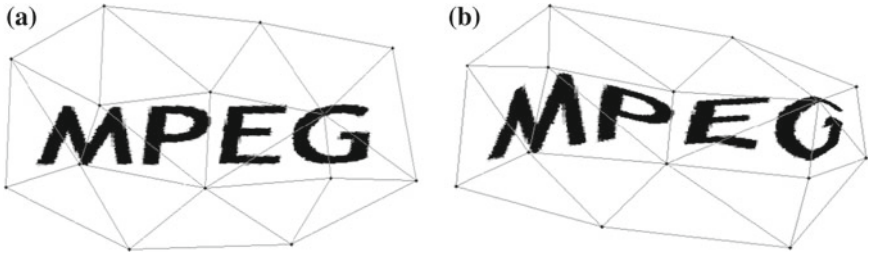

Fig. 11.25 Mesh-based texture mapping for 2D object animation. **a** the original mesh; **b** the mesh after an affine transform

matrix given by $\mathbf{X^{-1}} = adj(\mathbf{X})/det(\mathbf{X})$, where $adj(\mathbf{X})$ is the adjoint of $\mathbf{X}$ and $det(\mathbf{X})$ is the determinant. Therefore,

$$\begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & 1 \end{bmatrix} = \begin{bmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{bmatrix}^{-1} \begin{bmatrix} x'_0 & y'_0 & 1 \\ x'_1 & y'_1 & 1 \\ x'_2 & y'_2 & 1 \end{bmatrix}$$

$$= \frac{1}{det(\mathbf{X})} \begin{bmatrix} y_1 - y_2 & y_2 - y_0 & y_0 - y_1 \\ x_2 - x_1 & x_0 - x_2 & x_1 - x_0 \\ x_1 y_2 - x_2 y_1 & x_2 y_0 - x_0 y_2 & x_0 y_1 - x_1 y_0 \end{bmatrix} \begin{bmatrix} x'_0 & y'_0 & 1 \\ x'_1 & y'_1 & 1 \\ x'_2 & y'_2 & 1 \end{bmatrix} \quad (11.16)$$

where $det(\mathbf{X}) = x_0(y_1 - y_2) - y_0(x_1 - x_2) + (x_1 y_2 - x_2 y_1)$.

Since the three vertices of the mesh triangle are never colinear points, it is ensured that $\mathbf{X}$ is not singular—that is, $det(\mathbf{X}) \neq 0$. Therefore Eq. (11.16) always has a unique solution.

The above affine transform is piecewise—that is, each triangle can have its own affine transform. It works well only when the object is mildly deformed during the animation sequence. Figure 11.25a shows a Delaunay mesh with a simple word mapped onto it. Figure 11.25b shows the warped word in a subsequent MOP in the animated sequence after an affine transform.

3D Model-Based Coding

Because of the frequent appearances of human faces and bodies in videos, MPEG-4 has defined special 3D models for *face objects* and *body objects*. Some of the potential applications for these new video objects include teleconferencing, human–computer interfaces, games, and e-commerce. In the past, 3D wireframe models and their animations have been studied for 3D object animation [25]. MPEG-4 goes beyond wireframes, so that the surfaces of the face or body objects can be shaded or texture-mapped.

Face Object Coding and Animation. Face models for individual faces could either be created manually or generated automatically through computer vision and

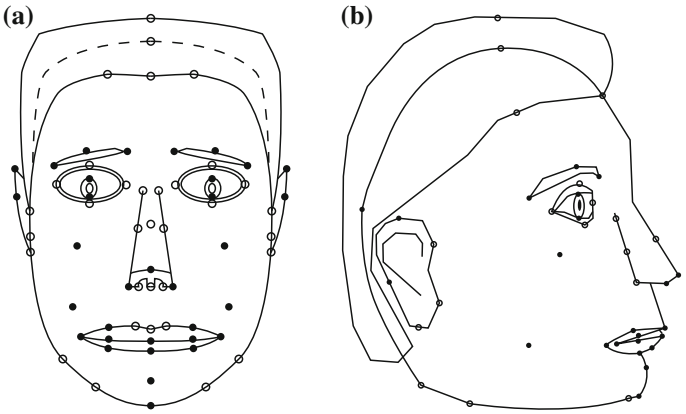

Fig. 11.26 Feature points for face definition parameters (FDPs) (Feature points for teeth and tongue are not shown) **a** front view; **b** side view

pattern recognition techniques. However, the former is cumbersome and nevertheless inadequate, and the latter has yet to be achieved reliably.

MPEG-4 has adopted a generic default face model, developed by the Virtual Reality Modeling Language (VRML) Consortium [26]. *Face Animation Parameters (FAPs)* can be specified to achieve desirable animations—deviations from the original "neutral" face. In addition, *Face Definition Parameters (FDPs)* can be specified to better describe individual faces. Figure 11.26 shows the feature points for FDPs. Feature points that can be affected by animation (FAPs) are shown as solid circles; and those that are not affected are shown as empty circles.

Sixty-eight FAPs are defined [22]: FAP 1 is for visemes and FAP 2 for facial expressions. Visemes code highly realistic lip motions by modeling the speaker's current mouth position. All other FAPs are for possible movements of head, jaw, lip, eyelid, eyeball, eyebrow, pupil, chin, cheek, tongue, nose, ear, and so on.

For example, expressions include *neutral, joy, sadness, anger, fear, disgust and surprise*. Each is expressed by a set of features—sadness for example, by slightly closed eyes, relaxed mouth, and upward-bent inner eyebrows. FAPs for movement include *head_pitch, head_yaw, head_roll, open_jaw, thrust_jaw, shift_jaw, push_bottom_lip, push_top_lip,* and so on.

For compression, the FAPs are coded using predictive coding. Predictions for FAPs in the target frame are made based on FAPs in the previous frame, and prediction errors are then coded using arithmetic coding. DCT can also be employed to improve the compression ratio, although it is considered more computationally expensive. FAPs are also quantized, with different quantization step sizes employed to explore the fact that certain FAPs (e.g., open_jaw) need less precision than others (e.g., push_top_lip).

Body Object Coding and Animation. MPEG-4 Version 2 introduced *body objects*, which are a natural extension to face objects.

Working with the Humanoid Animation (H-Anim) Group in the VRML Consortium, MPEG adopted a generic virtual human body with default posture. The default is standing, with feet pointing to the front, arms at the sides, with palms facing inward. There are 296 *Body Animation Parameters* (BAPs). When applied to any MPEG-4-compliant generic body, they will produce the same animation.

A large number of BAPs describe joint angles connecting different body parts, including the spine, shoulder, clavicle, elbow, wrist, finger, hip, knee, ankle, and toe. This yields 186 degrees of freedom to the body, 25 to each hand alone. Furthermore, some body movements can be specified in multiple levels of detail. For example, five different levels, supporting 9, 24, 42, 60, and 72 degrees of freedom can be used for the spine, depending on the complexity of the animation.

For specific bodies, *Body Definition Parameters* (BDPs) can be specified for body dimensions, body surface geometry, and, optionally, texture. Body surface geometry uses a *3D polygon mesh* representation, consisting of a set of polygonal planar surfaces in 3D space [24]. The 3D mesh representation is popular in computer graphics for surface modeling. Coupled with texture mapping, it can deliver good (photorealistic) renderings.

The coding of BAPs is similar to that of FAPs: quantization and predictive coding are used, and the prediction errors are further compressed by arithmetic coding.

11.4.4 MPEG-4 Parts, Profiles and Levels

So far, MPEG-4 has over 28 Parts [9], and more are still being developed. It not only specified Visual in Part 2 and Audio in Part 3, but also specialized subjects such as Graphics, Animation, Music, Scene description, Object descriptor, Delivery Multimedia Integration Framework (DMIF), Streaming, and Intellectual Property Management and Protection (IPMP) in various Parts. MPEG-4 Part 10 is about Advanced Video Coding (AVC) which is identical to ITU-T H.264 AVC.

MPEG-4 Part 2 defined more than 20 visual Profiles, e.g., Simple, Advanced Simple, Core, Main, Simple Studio, etc. The commonly used ones are: *Simple Profile (SP)* and *Advanced Simple Profile (ASP)*. The latter is adopted by several popular video coding software such as DivX, Nero Digital, and Quicktime 6. The Open-source software Xvid supports both SP and ASP.

To target for various applications, MPEG-4 Part 2 also defined multiple Levels in each profile, e.g., L0 to L3 for SP, and L0 to L5 for ASP. In general, the lower Levels in these profiles support low-bitrate video formats (CIF, QCIF) and applications such as videoconferencing on the web; whereas the higher Levels support higher quality videos.

11.5 MPEG-7

As more and more multimedia content becomes an integral part of various applications, effective and efficient retrieval becomes a primary concern. In October 1996, the MPEG group therefore took on the development of another major standard, MPEG-7, following on MPEG-1, 2, and 4.

One common ground between MPEG-4 and MPEG-7 is the focus on audiovisual *objects*. The main objective of MPEG-7 [27–29] is to serve the need of audiovisual content-based retrieval (or audiovisual object retrieval) in applications such as digital libraries. Nevertheless, it is certainly not limited to retrieval—it is applicable to any multimedia applications involving the generation (*content creation*) and usage (*content consumption*) of multimedia data. Unlike MPEG-1, -2 and -4, it is not "yet another" standard for video coding.

MPEG-7 became an international standard in September 2001. Its formal name is *Multimedia Content Description Interface*, documented in ISO/IEC 15938 [30]. The standard's first seven Parts are Systems, Description Definition Language, Visual, Audio, Multimedia Description Schemes, Reference Software, and Conformance and Testing. Since 2002, there are further developments in Parts 8 to 12, mostly focusing on various profiles and query format.

MPEG-7 supports a variety of multimedia applications. Its data may include still pictures, graphics, 3D models, audio, speech, video, and composition information (how to combine these elements). These MPEG-7 data elements can be represented in textual or binary format, or both. Part 1 (Systems) specifies the syntax of *Binary format for MPEG-7* (BiM) data. Part 2 (Description Definition Language) specifies the syntax of the textual format which adopts XML Schema as its language of choice. A bidirectional lossless mapping is defined between the textual and binary representations.

Figure 11.27 illustrates some possible applications that will benefit from MPEG-7. As shown, features are extracted and used to instantiate MPEG-7 *descriptions*. They are then coded by the MPEG-7 encoder and sent to the *storage and transmission media*. Various search and query engines issue search and browsing requests, which constitute the *pull* activities of the Internet, whereas the agents filter out numerous materials *pushed* onto the *terminal*—users and/or computer systems and applications that consume the data.

For multimedia content description, MPEG-7 has developed *Descriptors* (D), *Description Schemes* (DS), and a *Description Definition Language* (DDL). Following are some of the important terms:

- **Feature.** A characteristic of the data
- **Descriptor (D).** A definition (syntax and semantics) of the feature
- **Description Scheme (DS).** Specification of the structure and relationship between Ds and DSs (see [31])
- **Description.** A set of instantiated Ds and DSs that describes the structural and conceptual information of the content, storage and usage of the content, and so on
- **Description Definition Language (DDL).** Syntactic rules to express and combine DSs and Ds (see [32])

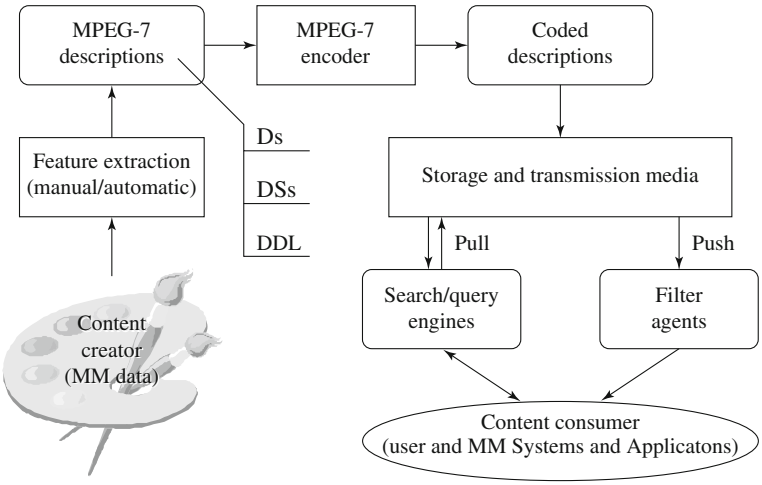

Fig. 11.27 Possible applications using MPEG-7

It is made clear [30] that the scope of MPEG-7 is to standardize the Ds, DSs and DDL for descriptions. The mechanism and process of producing and consuming the descriptions are beyond the scope of MPEG-7. These are left open for industry innovation and competition and, more importantly, for the arrival of ever-improving new technologies.

Similar to the Simulation Model (SM) in MPEG-1 video, the Test Model (TM) in MPEG-2 video, and the Verification Models (VMs) in MPEG-4 (video, audio, SNHC, and systems), MPEG-7 names its working model the *Experimentation Model (XM)*—an alphabetical pun! XM provides descriptions of various tools for evaluating the Ds, DSs and DDL, so that experiments and verifications can be conducted and compared by multiple independent parties all over the world. The first set of such experiments is called the *core experiments*.

11.5.1 Descriptor (D)

MPEG-7 descriptors are designed to describe both low-level features, such as color, texture, shape, and motion, and high-level features of semantic objects, such as events and abstract concepts. As mentioned above, methods and processes for automatic and even semiautomatic feature extraction are not part of the standard. Despite the efforts and progress in the fields of image and video processing, computer vision, and pattern recognition, automatic and reliable feature extraction is not expected in the near future, especially at the high level.

The descriptors are chosen based on a comparison of their performance, efficiency, and size. Low-level visual descriptors for basic visual features [33] include

- **Color**
  - **Color space.** (a) RGB, (b) YCbCr, (c) HSV (hue, saturation, value) [24], (d) HMMD (HueMaxMinDiff) [34], (e) 3D color space derivable by a 3×3 matrix from RGB, (f) monochrome
  - **Color quantization.** (a) Linear, (b) nonlinear, (c) lookup tables
  - **Dominant colors.** A small number of representative colors in each region or image. These are useful for image retrieval based on color similarity
  - **Scalable color.** A color histogram in HSV color space. It is encoded by a Haar transform and hence is scalable
  - **Color layout.** Spatial distribution of colors for color-layout-based retrieval
  - **Color structure.** The frequency of a *color structuring element* describes both the color content and its structure in the image. The color structure element is composed of several image samples in a local neighborhood that have the same color
  - **Group of Frames/ Group of Pictures (GoF/GoP) color.** Similar to the scalable color, except this is applied to a video segment or a group of still images. An aggregated color histogram is obtained by the application of *average, median*, or *intersection* operations to the respective bins of all color histograms in the GoF/GoP and is then sent to the Haar transform
- **Texture**
  - **Homogeneous texture.** Uses orientation and scale-tuned Gabor filters [35] that quantitatively represent regions of homogeneous texture. The advantage of Gabor filters is that they provide simultaneous optimal resolution in both space and spatial frequency domains [36]. Also, they are bandpass filters that conform to the human visual profile. A filter bank consisting of 30 Gabor filters, at five different scales and six different directions for each scale, is used to extract the texture descriptor
  - **Texture browsing.** Describes the *regularity*, *coarseness*, and *directionality* of edges used to represent and browse homogeneous textures. Again, Gabor filters are used
  - **Edge histogram.** Represents the spatial distribution of four directional (0°, 45°, 90°, 135°) edges and one nondirectional edge. Images are divided into small subimages, and an edge histogram with five bins is generated for each subimage
- **Shape**
  - **Region-based shape.** A set of *Angular Radial Transform* (ART) [28] coefficients is used to describe an object's shape. An object can consist of one or more regions, with possibly some holes in the object. ART transform is a 2D complex transform defined in terms of polar coordinates on a unit disc. ART basis functions are separable along the angular and radial dimensions. Thirty-six basic functions, 12 angular, and 3 radial, are used to extract the shape descriptor
  - **Contour-based shape.** Uses a *curvature scale space* (CSS) representation [37] that is invariant to scale and rotation, and robust to nonrigid motion and partial occlusion of the shape
  - **3D shape.** Describes 3D mesh models and *shape index* [38]. The histogram of the shape indices over the entire mesh is used as the descriptor

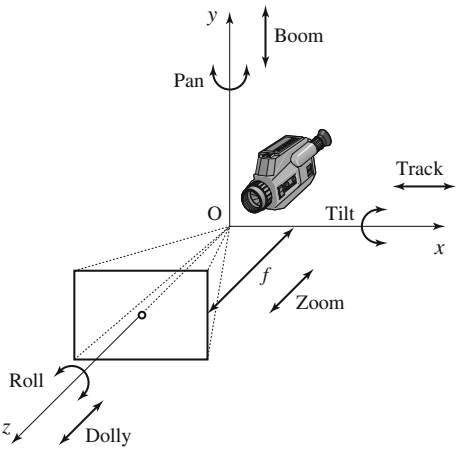

Fig. 11.28 Camera motions: pan, tilt, roll, dolly, track, and boom (Camera has an effective focal length of f. It is shown initially at the origin, pointing to the direction of z-axis)

- **Motion**
  - **Camera motion.** Fixed, pan, tilt, roll, dolly, track, boom (See Fig. 11.28 and [39])
  - **Object motion trajectory.** A list of keypoints (x, y, z, t). Optional interpolation functions are used to specify the acceleration along the path (See [39])
  - **Parametric object motion.** The basic model is the 2D affine model for translation, rotation, scaling, shear, and the combination of these. A planar perspective model and quadratic model can be used for perspective distortion and more complex movements
  - **Motion activity.** Provides descriptions such as the intensity, pace, mood, and so on, of the video—for example, "scoring in a hockey game" or "interviewing a person"
- **Localization**
  - **Region locator.** Specifies the localization of regions in images with a box or a polygon
  - **Spatiotemporal locator.** Describes spatiotemporal regions in video sequences. Uses one or more sets of descriptors of regions and their motions
- **Others**
  - **Face recognition.** A normalized face image is represented as a 1D vector, then projected onto a set of 49 basis vectors, representing all possible face vectors.

11.5.2 Description Scheme (DS)

This section provides a brief overview of MPEG-7 Description Schemes (DSs) in the areas of Basic elements, Content management, Content description, Navigation and access, Content organization, and User interaction.

- **Basic elements**
  - **Datatypes and mathematical structures.** Vectors, matrices, histograms, and so on
  - **Constructs.** Links media files and localizing segments, regions, and so on
  - **Schema tools.** Includes root elements (starting elements of MPEG-7 XML documents and descriptions), top-level elements (organizing DSs for specific content-oriented descriptions), and package tools (grouping related DS components of a description into packages)
- **Content Management**
  - **Media Description.** Involves a single DS, the *MediaInformation* DS, composed of a *MediaIdentification* D and one or more *Media Profile* Ds that contain information such as coding method, transcoding hints, storage and delivery formats, and so on
  - **Creation and Production Description.** Includes information about creation (title, creators, creation location, date, etc.), classification (genre, language, parental guidance, etc.), and related materials
  - **Content Usage Description.** Various DSs to provide information about usage rights, usage record, availability, and finance (cost of production, income from content use)
- **Content Description**
  - **Structural Description.** A *Segment* DS describes structural aspects of the content. A *segment* is a section of an audiovisual object. The relationship among segments is often represented as a *segment tree*. When the relationship is not purely hierarchical, a *segment graph* is used

    The *Segment* DS can be implemented as a class object. It has five subclasses: *Audiovisual segment DS*, *Audio segment DS*, *Still region DS*, *Moving region DS*, and *Video segment DS*. The subclass DSs can recursively have their own subclasses

    A Still region DS, for example, can be used to describe an image in terms of its creation (title, creator, date), usage (copyright), media (file format), textual annotation, color histogram, and possibly texture descriptors, and so on. The initial region (image, in this case) can be further decomposed into several regions, which can in turn have their own DSs.

    Figure 11.29 shows a Video segment for a marine rescue mission, in which a person was lowered onto a boat from a helicopter. Three moving regions are inside the Video segment. A segment graph can be constructed to include such structural descriptions as composition of the video frame (helicopter, person, boat) spatial relationship and motion (above, on, close-to, move-toward, etc.) of the regions
  - **Conceptual Description.** This involves higher level (nonstructural) description of the content, such as *Event* DS for basketball game or Lakers ballgame, *Object* DS for John or person, *State* DS for semantic properties at a given time or location, and *Concept* DS for abstract notations such as "freedom" or "mystery." As for Segment DSs, the concept DSs can also be organized in a tree or graph

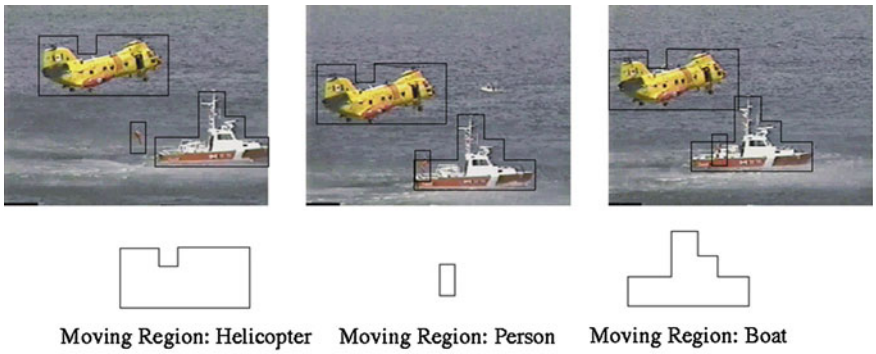

Fig. 11.29 MPEG-7 video segment

- **Navigation and access**
  - **Summaries.** These provide a video summary for quick browsing and navigation of the content, usually by presenting only the keyframes. The following DSs are supported: *Summarization* DS, *HierarchicalSummary* DS, *HighlightLevel* DS, *SequentialSummary* DS. Hierarchical summaries provide a keyframe hierarchy of multiple levels, whereas sequential summaries often provide a slide show or audiovisual skim, possibly with synchronized audio and text
    Figure 11.30 illustrates a summary for a video of a "dragon-boat" parade and race in a park. The summary is organized in a three-level hierarchy. Each video segment at each level is depicted by a keyframe of thumbnail size
  - **Partitions and Decompositions.** This refers to *view* partitions and decompositions. The *View partitions* (specified by *View* DSs) describe different space and frequency views of the audiovisual data, such as a spatial view (this could be a spatial segment of an image), temporal view (as in a temporal segment of a video), frequency view (as in a wavelet subband of an image), or resolution view (as in a thumbnail image), and so on. The *View decompositions* DSs specify different tree or graph decompositions for organizing the views of the audiovisual data, such as a SpaceTree DS (a quad-tree image decomposition)
  - **Variations of the Content.** A *Variation* DS specifies a variation from the original data in image resolution, frame rate, color reduction, compression, and so on. It can be used by servers to adapt audiovisual data delivery to network and terminal characteristics for a given Quality of Service (QoS)
- **Content Organization**
  - **Collections.** The *CollectionStructure* DS groups audiovisual contents into clusters. It specifies common properties of the cluster elements and relationships among the clusters
  - **Models.** *Model* DSs include a Probability model DS, Analytic model DS, and Classifier DS that extract the models and statistics of the attributes and features of the collections

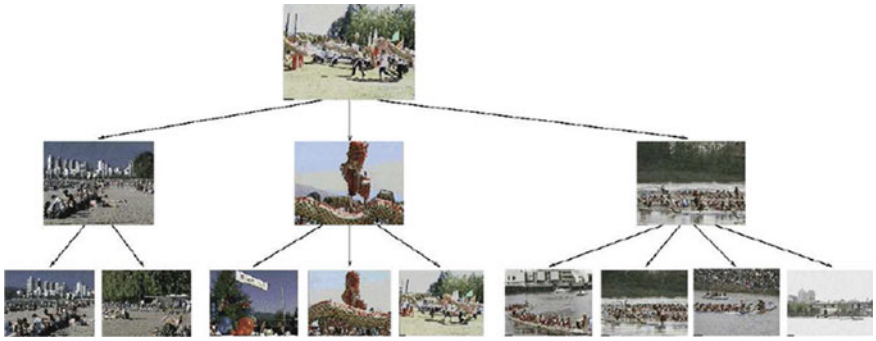

Fig. 11.30 A video summary

- **User Interaction**
  - **UserPreference.** DSs describe user preferences in the consumption of audiovisual contents, such as content types, browsing modes, privacy characteristics, and whether preferences can be altered by an agent that analyzes user behavior.

11.5.3 Description Definition Language (DDL)

MPEG-7 adopted the XML Schema Language initially developed by the WWW Consortium (W3C) as its Description Definition Language (DDL). Since XML Schema Language was not designed specifically for audiovisual contents, some extensions are made to it. Without the details, the MPEG-7 DDL has the following components:

- **XML Schema structure components**
  - The Schema—the wrapper around definitions and declarations
  - Primary structural components, such as simple and complex type definitions, and attribute and element declarations
  - Secondary structural components, such as attribute group definitions, identity-constraint definitions, group definitions, and notation declarations
  - "Helper" components, such as annotations, particles, and wildcards
- **XML Schema datatype components**
  - Primitive and derived data types
  - Mechanisms for the user to derive new data types
  - Type checking better than XML 1.0
- **MPEG-7 Extensions**
  - Array and matrix data types
  - Multiple media types, including audio, video, and audiovisual presentations
  - Enumerated data types for `MimeType`, `CountryCode`, `RegionCode`, `CurrencyCode`, and `CharacterSetCode`
  - *Intellectual Property Management and Protection* (IPMP) for Ds and DSs

11.6 Exercises

1. As we know, MPEG video compression uses I-, P-, and B-frames. However, the earlier H.261 standard does not use B-frames. Describe a situation in which video compression would not be as effective without B-frames (Your answer should be different from the one in Fig. 11.1).
2. The MPEG-1 standard introduced B-frames, and the motion-vector search range has accordingly been increased from $[-15, 15]$ in H.261 to $[-512, 511.5]$. Why was this necessary? Calculate the number of B-frames between consecutive P-frames that would justify this increase.
3. B-frames provide obvious coding advantages, such as increase in SNR at low bitrates and bandwidth savings. What are some of the disadvantages of B-frames?
4. Redraw Fig. 11.8 of the MPEG-2 two-layer SNR scalability encoder and decoder to include a second enhancement layer.
5. Draw block diagrams for an MPEG-2 encoder and decoder for (a) SNR and spatial hybrid scalability, (b) SNR and temporal hybrid scalability.
6. Why are not B-frames used as reference frames for motion compensation? Suppose there is a mode where any frame type can be specified as a reference frame. Discuss the tradeoffs of using reference B-frames instead of P-frames in a video sequence (i.e., eliminating P-frames completely).
7. Write a program to implement the SNR scalability in MPEG-2. Your program should be able to work on any macroblock using any quantization *step_sizes* and should output both `Bits_base` and `Bits_enhance` bitstreams. The variable-length coding step can be omitted.
8. MPEG-4 motion compensation is supposed to be VOP-based. At the end, the VOP is still divided into macroblocks (interior macroblock, boundary macroblock, etc.) for motion compensation.
   (a) What are the potential problems of the current implementation? How can they be improved?
   (b) Can there be true VOP-based motion compensation? How would it compare to the current implementation?
9. MPEG-1, 2, and 4 are all known as decoder standards. The compression algorithms, hence the details of the encoder, are left open for future improvement and development. For MPEG-4, the major issue of *video object segmentation*—how to obtain the VOPs—is left unspecified.
   (a) Propose some of your own approaches to video object segmentation.
   (b) What are the potential problems of your approach?
10. Motion vectors can have subpixel precision. In particular, MPEG-4 allows quarter-pixel precision in the luminance VOPs. Describe an algorithm that will realize this precision.
11. As a programming project, compute the SA-DCT for the following 8×8 block:

$$\begin{matrix} 0 & 0 & 0 & 0 & 16 & 0 & 0 & 0 \\ 4 & 0 & 8 & 16 & 32 & 16 & 8 & 0 \\ 4 & 0 & 16 & 32 & 64 & 32 & 16 & 0 \\ 0 & 0 & 32 & 64 & 128 & 64 & 32 & 0 \\ 4 & 0 & 0 & 32 & 64 & 32 & 0 & 0 \\ 0 & 16 & 0 & 0 & 32 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 16 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

12. What is the computational cost of SA-DCT, compared to ordinary DCT? Assume the video object is a 4×4 square in the middle of an 8×8 block.
13. Affine transforms can be combined to yield a composite affine transform. Prove that the composite transform will have exactly the same form of matrix (with $[0\ 0\ 1]^T$ as the last column) and at most 6 degrees of freedom, specified by the parameters $a_{11}, a_{21}, a_{31}, a_{12}, a_{22}, a_{32}$.
14. Mesh-based motion coding works relatively well for 2D animation and face animation. What are the main problems when it is applied to body animation?
15. What is the major motivation behind the development of MPEG-7? Give three examples of real-world applications that may benefit from MPEG-7.
16. Two of the main shape descriptors in MPEG-7 are "region-based" and "contour-based." There are, of course, numerous ways of describing the shape of regions and contours.
    (a) What would be your favorite shape descriptor?
    (b) How would it compare to ART and CSS in MPEG-7?

References

1. L. Chiariglione, The development of an integrated audiovisual coding standard: MPEG. Proc. IEEE **83**, 151–157 (1995)
2. D.J. Le Gall, MPEG: a video compression standard for multimedia applications. Commun. ACM **34**(4), 46–58 (1991)
3. R. Schafer, T. Sikora, Digital video coding standards and their role in video communications. Proc. IEEE **83**(6), 907–924 (1995)
4. Information technology—Coding of moving pictures and associted audio for digital storage media at up to about 1.5 Mbit/s. Int. Standard: ISO/IEC 11172, Parts 1–5 (1992)
5. J.L. Mitchell, W.B. Pennebaker, C.E. Fogg, D.J. LeGall, *MPEG Video Compression Standard*. (Chapman & Hall, New York, 1996)
6. B.G. Haskell, A. Puri, A. Netravali, *Digital Video: an Introduction to MPEG-2*. (Chapman & Hall, New York, 1996)
7. Information technology—Generic coding of moving pictures and associated audio information. Int. Standard: ISO/IEC 13818, Parts 1–11 (2004)
8. T. Sikora, The MPEG-4 video standard verification model. IEEE Trans. Circuits Syst. Video Technol. (Special issue on MPEG-4) **7**(1), 19–31 (1997)

9. Information technology—Generic coding of audio-visual objects. Int. Standard: ISO/IEC 14496, Parts 1–28 (2012)
10. A. Puri, T. Chen (eds.), *Multimedia Systems, Standards, and Networks* (Marcel Dekker, New York, 2000)
11. G. Fernando et al., Java in MPEG-4 (MPEG-J), in *Multimedia, Systems, Standards, and Networks*, ed. by A. Puri, T. Chen (Marcel Dekker, New York, 2000), pp. 449–460
12. Video Coding for Low Bit Rate Communication, ITU-T Recommendation H.263, Version 1, 1995, Version 2, 1998, Version 3, 2000, Revised 2005
13. A. Puri et al., MPEG-4 natural video coding—Part I, in *Multimedia, Systems, Standards, and Networks*, ed. by A. Puri, T. Chen (Marcel Dekker, New York, 2000), pp. 205–244
14. T. Ebrahimi, F. Dufaux, Y. Nakaya, MPEG-4 natural video coding - Part II, in *Multimedia, Systems, Standards, and Networks*, ed. by A. Puri, T. Chen (Marcel Dekker, New York, 2000), pp. 245–269
15. P. Kauff, et al. Functional coding of video using a shape-adaptive DCT algorithm and an object-based motion prediction toolbox. IEEE Trans. Circuits Syst. Video Technol. (Special issue on MPEG-4) **7**(1), 181–196 (1997)
16. Standardization of Group 3 facsimile apparatus for document transmission. ITU-T Recommendation T.4, 1980
17. Facsimile coding schemes and coding control functions for Group 4 facsimile apparatus. ITU-T Recommendation T.6, 1984
18. Information technology—Coded representation of picture and audio information—progressive bi-Level image compression. Int. Standard: ISO/IEC 11544, also ITU-T Recommendation T.82, 1992
19. J.M. Shapiro, Embedded image coding using zerotrees of wavelet coefficients. IEEE Trans. Signal Process. **41**(12), 3445–3462 (1993)
20. G. Wolberg, *Digital Image Warping* (Computer Society Press, Los Alamitos, CA, 1990)
21. M.C. Lee, et al. A layered video object coding system using sprite and affine motion model. IEEE Trans. Circuits Syst. Video Technol. **7**(1), 130–145 (1997)
22. P. van Beek, MPEG-4 synthetic video, in *Multimedia, Systems, Standards, and Networks*, ed. by A. Puri, T. Chen (Marcel Dekker, New York, 2000), pp. 299–330
23. A.M. Tekalp, P. van Beek, C. Toklu, B. Gunsel, 2D mesh-based visual object representation for interactive synthetic/natural digital video. Proc. IEEE **86**, 1029–1051 (1998)
24. John F. Hughes, Andries van Dam, Morgan McGuire, David F. Sklar, James D. Foley, Steven K. Feiner, K. Akeley, *Computer Graphics: Principles and Practice*, 3rd ed. (Addison-Wesley, Upper Saddle River, 2013)
25. A. Watt, M. Watt, *Advanced Animation and Rendering Techniques*. (Addison-Wesley, Upper Saddle River, 1992)
26. Information technology—The Virtual Reality Modeling Language—Part 1: Functional specification and UTF-8 encoding. Int. Standard: ISO/IEC 14772–1 (1997)
27. S.F. Chang, T. Sikora, A. Puri, Overview of the MPEG-7 standard. IEEE Trans. Circuits Syst. Video Technol. (Special issue on MPEG-7) **11**(6), 688–695 (2001)
28. B.S. Manjunath, P. Salembier, T. Sikora (eds.), *Introduction to MPEG-7: Multimedia Content Description Interface*. (Wiley, Chichester, 2002)
29. H.G. Kim, N. Moreau, T. Sikora, *MPEG-7 Audio and Beyond: Audio Content Indexing and Retrieval*. (Wiley, New York, 2005)
30. Information technology—Multimedia content description interface. Int. Standard: ISO/IEC 15938, Parts 1–12 (2008)
31. P. Salembier, J. R. Smith, MPEG-7 multimedia description schemes. IEEE Trans. Circuits Syst. Video Technol. **11**(6), 748–759 (2001)
32. J. Hunter, F. Nack, An overview of the MPEG-7 description definition language (DDL) proposals. Signal Process. Image Commun. **16**(1–2), 271–293 (2001)

33. T. Sikora, The MPEG-7 visual standard for content description—an overview. IEEE Trans. Circuits Syst. Video Technol. (Special issue on MPEG-7) **11**(6), 696–702 (2001)
34. B.S. Manjunath, J.-R. Ohm, V.V. Vasudevan, A. Yamada, Color and texture descriptors. IEEE Trans. Circuits Syst. Video Technol. **11**, 703–715 (2001)
35. B.S. Manjunath, G.M. Haley, W.Y. Ma, in *Multiband Techniques for Texture Classification and Segmentation*, ed. by A. Bovik, Handbook of Image and Video Processing (Academic Press, San Diego, 2000), pp. 367–381
36. T.P. Weldon, W.E. Higgins, D.F. Dunn, Efficient Gabor filter design for texture segmentation. Pattern Recogn **29**(12), 2005-2015 (1996)
37. F. Mokhtarian, A.K. Mackworth, A theory of multiscale, curvature-based shape representation for planar curves. IEEE Trans. Pattern Anal. Mach. Intell. **14**(8), 789–805 (1992)
38. J.J. Koenderink, A.J. van Doorn, Surface shape and curvature scales. Image Vision Comput. **10**, 557–565 (1992)
39. S. Jeannin et al., Motion descriptor for content-based video representation. Signal Process. Image Commun. **16**(1–2), 59–85 (2000)