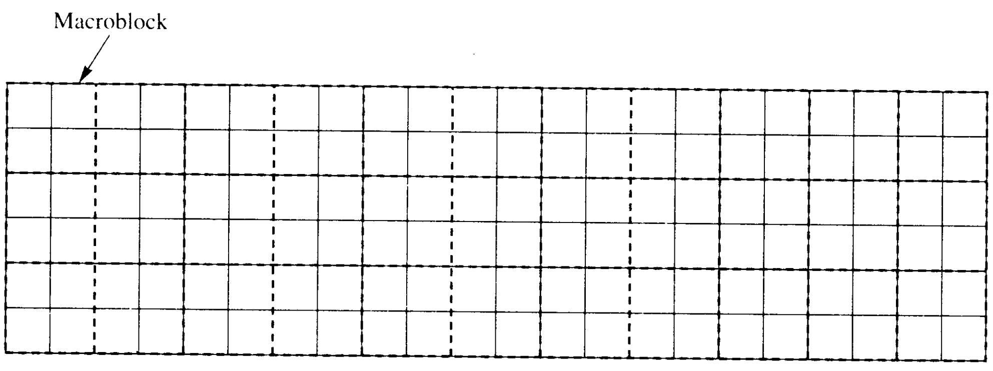

FIGURE 18. 12 **A GOB consisting of 33 macroblocks.**

buffer. Once a quantizer is selected, the receiver has to be informed about the selection. In H.261, this is done in one of two ways. Each macroblock is preceded by a header. The quantizer being used can be identified as part of this header. When the amount of activity or motion in the sequence is relatively constant, it is reasonable to expect that the same quantizer will be used for a large number of macroblocks. In this case, it would be wasteful to identify the quantizer being used with each macroblock. The macroblocks are organized into *groups of blocks* (GOBs), each of which consist of three rows of 11 macroblocks. This hierarchical arrangement is shown in Figure 18.12. Only the luminance blocks are shown. The header preceding each GOB contains a 5-bit field for identifying the quantizer. Once a quantizer has been identified in the GOB header, the receiver assumes that quantizer is being used, unless this choice is overridden using the macroblock header.

The quantization labels are encoded in a manner similar to, but not exactly the same as, JPEG. The labels are scanned in a zigzag fashion like JPEG. The nonzero labels are coded along with the number, or run, of coefficients quantized to zero. The 20 most commonly occurring combinations of (run, label) are coded with a single variable-length codeword. All other combinations of (run, label) are coded with a 20-bit word, made up of a 6-bit escape sequence, a 6-bit code denoting the run, and an 8-bit code for the label.

In order to avoid transmitting blocks that have no nonzero quantized coefficient, the header preceding each macroblock can contain a variable-length code called the *coded block pattern* (CBP) that indicates which of the six blocks contain nonzero labels. The CBP can take on one of 64 different pattern numbers, which are then encoded by a variable-length code. The pattern number is given by

$$\text{CBP} = 32P_1 + 16P_2 + 8P_3 + 4P_4 + 2P_5 + P_6$$

where P_1 through P_6 correspond to the six different blocks in the macroblock, and is one if the corresponding block has a nonzero quantized coefficient and zero otherwise.

18.5.5 Rate Control

The binary codewords generated by the transform coder form the input to a transmission buffer. The function of the transmission buffer is to keep the output rate of the encoder fixed. If the buffer starts filling up faster than the transmission rate, it sends a message back to the transform coder to reduce the output from the quantization. If the buffer is in danger of becoming emptied because the transform coder is providing bits at a rate lower than the transmission rate, the transmission buffer can request a higher rate from the transform coder. This operation is called *rate control.*

The change in rate can be affected in two different ways. First, the quantizer being used will affect the rate. If a quantizer with a large step size is used, a larger number of coefficients will be quantized to zero. Also, there is a higher probability that those not quantized to zero will be one of the those values that have a shorter variable-length codeword. Therefore, if a higher rate is required, the transform coder selects a quantizer with a smaller step size, and if a lower rate is required, the transform coder selects a quantizer with a larger step size. The quantizer step size is set at the beginning of each GOB, but can be changed at the beginning of any macroblock. If the rate cannot be lowered enough and there is a danger of buffer overflow, the more drastic option of dropping frames from transmission is used.

The ITU-T H.261 algorithm was primarily designed for videophone and videoconferencing applications. Therefore, the algorithm had to operate with minimal coding delay (less than 150 milliseconds). Furthermore, for videophone applications, the algorithm had to operate at very low bit rates. In fact, the title for the recommendation is "Video Codec for Audiovisual Services at $p \times 64\,\text{kbit/s}$," where p takes on values from 1 to 30. A p value of 2 corresponds to a total transmission rate of 128 kbps, which is the same as two voice-band telephone channels. These are very low rates for video, and the ITU-T H.261 recommendations perform relatively well at these rates.

18.6 Model-Based Coding

In speech coding, a major decrease in rate is realized when we go from coding waveforms to an analysis/synthesis approach. An attempt at doing the same for video coding is described in the next section. A technique that has not yet reached maturity but shows great promise for use in videophone applications is an analysis/synthesis technique. The analysis/synthesis approach requires that the transmitter and receiver agree on a model for the information to be transmitted. The transmitter then analyzes the information to be transmitted and extracts the model parameters, which are transmitted to the receiver. The receiver uses these parameters to synthesize the source information. While this approach has been successfully used for speech compression for a long time (see Chapter 15), the same has not been true for images. In a delightful book, *Signals, Systems, and Noise—The Nature and Process of Communications*, published in 1961, J.R. Pierce [14] described his "dream" of an analysis/synthesis scheme for what we would now call a videoconferencing system:

> Imagine that we had at the receiver a sort of rubbery model of the human face. Or we might have a description of such a model stored in the memory of a huge electronic computer.... Then, as the person before the transmitter talked, the

transmitter would have to follow the movements of his eyes, lips, and jaws, and other muscular movements and transmit these so that the model at the receiver could do likewise.

Pierce's dream is a reasonably accurate description of a three-dimensional model-based approach to the compression of facial image sequences. In this approach, a generic wireframe model, such as the one shown in Figure 18.13, is constructed using triangles. When encoding the movements of a specific human face, the model is adjusted to the face by matching features and the outer contour of the face. The image textures are then mapped onto this wireframe model to synthesize the face. Once this model is available to both transmitter and receiver, only changes in the face are transmitted to the receiver. These changes can be

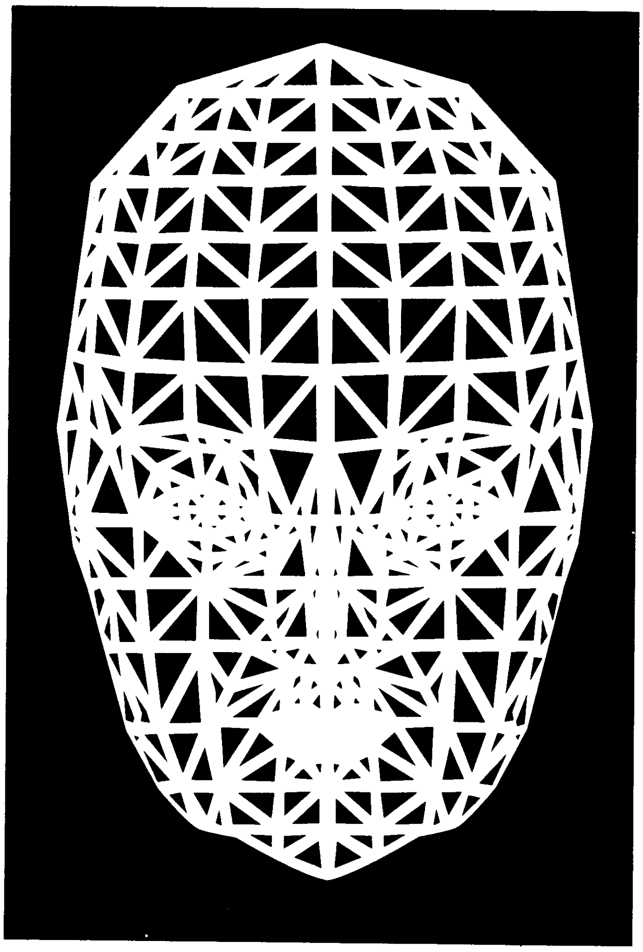

FIGURE 18.13 **Generic wireframe model.**

classified as *global motion* or *local motion* [246]. Global motion involves movement of the head, while local motion involves changes in the features—in other words, changes in facial expressions. The global motion can be modeled in terms of movements of rigid bodies. The facial expressions can be represented in terms of relative movements of the vertices of the triangles in the wireframe model. In practice, separating a movement into global and local components can be difficult because most points on the face will be affected by both the changing position of the head and the movement due to changes in facial expression. Different approaches have been proposed to separate these effects [247, 246, 248].

The global movements can be described in terms of rotations and translations. The local motions, or facial expressions, can be described as a sum of *action units* (AU), which are a set of 44 descriptions of basic facial expressions [249]. For example, AU1 corresponds to the raising of the inner brow and AU2 corresponds to the raising of the outer brow; therefore, AU1 + AU2 would mean raising the brow.

Although the synthesis portion of this algorithm is relatively straightforward, the analysis portion is far from simple. Detecting changes in features, which tend to be rather subtle, is a very difficult task. There is a substantial amount of research in this area, and if this problem is resolved, this approach promises rates comparable to the rates of the analysis/synthesis voice coding schemes. A good starting point for exploring this fascinating area is [250].

18.7 Asymmetric Applications

There are a number of applications in which it is cost effective to shift more of the computational burden to the encoder. For example, in multimedia applications where a video sequence is stored on a CD-ROM, the decompression will be performed many times and has to be performed in real time. However, the compression is performed only once, and there is no need for it to be in real time. Thus, the encoding algorithms can be significantly more complex. A similar situation arises in broadcast applications, where for each transmitter there might be thousands of receivers. In this section we will look at the standards developed for such asymmetric applications. These standards have been developed by a joint committee of the International Standards Organization (ISO) and the International Electrotechnical Society (IEC), which is best known as MPEG (Moving Picture Experts Group). MPEG was initially set up in 1988 to develop a set of standard algorithms, at different rates, for applications that required storage of video and audio on digital storage media. Originally, the committee had three work items, nicknamed MPEG-1, MPEG-2, and MPEG-3, targeted at rates of 1.5, 10, and 40 Mbits per second, respectively. Later, it became clear that the algorithms developed for MPEG-2 would accommodate the MPEG-3 rates, and the third work item was dropped [251]. The MPEG-1 work item resulted in a set of standards, ISO/IEC IS 11172, "Information Technology—Coding of Moving Pictures and Associated Audio for Digital Storage Media Up to about 1.5 Mbit/s" [252]. During the development of the standard, the committee felt that the restriction to digital storage media was not necessary, and the set of standards developed under the second work item, ISO/IEC 13818 or MPEG-2, has been issued under the title "Information Technology—Generic Coding of Moving Pictures and Associated Audio Information" [253]. In July of 1993 the MPEG committee began working on MPEG-4, the third and most ambitious of its standards. The goal of MPEG-4 was to provide an object-oriented framework for the encoding of multimedia. It took two years for the committee to arrive at a satisfactory definition of the scope of MPEG-4, and the call for proposals was finally issued in 1996. The standard ISO/IEC 14496 was finalized in 1998 and approved as an international standard in 1999. We have examined the audio standard in Chapter 16. In this section we briefly look at the video standards.

18.8 The MPEG-1 Video Standard

The basic structure of the compression algorithm proposed by MPEG is very similar to that of ITU-T H.261. Blocks (8×8 in size) of either an original frame or the difference between a frame and the motion-compensated prediction are transformed using the DCT. The blocks are organized in macroblocks, which are defined in the same manner as in the H.261 algorithm, and the motion compensation is performed at the macroblock level. The transform coefficients are quantized and transmitted to the receiver. A buffer is used to smooth delivery of bits from the encoder and also for rate control.

The basic structure of the MPEG-1 compression scheme may be viewed as very similar to that of the ITU-T H.261 video compression scheme; however, there are significant differences in the details of this structure. The H.261 standard has videophone and videoconferencing as the main application areas; the MPEG standard at least initially had applications that require digital storage and retrieval as a major focus. This does not mean that use of either algorithm is precluded in applications outside its focus, but simply that the features of the algorithm may be better understood if we keep in mind the target application areas. In videoconferencing a call is set up, conducted, and then terminated. This set of events always occurs together and in sequence. When accessing video from a storage medium, we do not always want to access the video sequence starting from the first frame. We want the ability to view the video sequence starting at, or close to, some arbitrary point in the sequence. A similar situation exists in broadcast situations. Viewers do not necessarily tune into a program at the beginning. They may do so at any random point in time. In H.261 each frame, after the first frame, may contain blocks that are coded using prediction from the previous frame. Therefore, to decode a particular frame in the sequence, it is possible that we may have to decode the sequence starting at the first frame. One of the major contributions of MPEG-1 was the provision of a random access capability. This capability is provided rather simply by requiring that there be frames periodically that are coded without any reference to past frames. These frames are referred to as **I** frames.

In order to avoid a long delay between the time a viewer switches on the TV to the time a reasonable picture appears on the screen, or between the frame that a user is looking for and the frame at which decoding starts, the **I** frames should occur quite frequently. However, because the **I** frames do not use temporal correlation, the compression rate is quite low compared to the frames that make use of the temporal correlations for prediction. Thus, the number of frames between two consecutive **I** frames is a trade-off between compression efficiency and convenience.

In order to improve compression efficiency, the MPEG-1 algorithm contains two other kinds of frames, the *predictive coded* (**P**) frames and the *bidirectionally predictive coded* (**B**) frames. The **P** frames are coded using motion-compensated prediction from the last **I** or **P** frame, whichever happens to be closest. Generally, the compression efficiency of **P** frames is substantially higher than **I** frames. The **I** and **P** frames are sometimes called *anchor* frames, for reasons that will become obvious.

To compensate for the reduction in the amount of compression due to the frequent use of **I** frames, the MPEG standard introduced **B** frames. The **B** frames achieve a high level of compression by using motion-compensated prediction from the most recent anchor frame and the closest future anchor frame. By using both past and future frames for prediction, generally we can get better compression than if we only used prediction based on the past. For example, consider a video sequence in which there is a sudden change between one frame and the next. This is a common occurrence in TV advertisements. In this situation, prediction based on the past frames may be useless. However, predictions based on future frames would have a high probability of being accurate. Note that a **B** frame can only be generated after the future anchor frame has been generated. Furthermore, the **B** frame is not used for predicting any other frame. This means that **B** frames can tolerate more error because this error will not be propagated by the prediction process.

The different frames are organized together in a *group of pictures* (GOP). A GOP is the smallest random access unit in the video sequence. The GOP structure is set up as a trade-off between the high compression efficiency of motion-compensated coding and the fast picture acquisition capability of periodic intra-only processing. As might be expected, a GOP has to contain at least one **I** frame. Furthermore, the first **I** frame in a GOP is either the first frame of the GOP, or is preceded by **B** frames that use motion-compensated prediction only from this **I** frame. A possible GOP is shown in Figure 18.14.

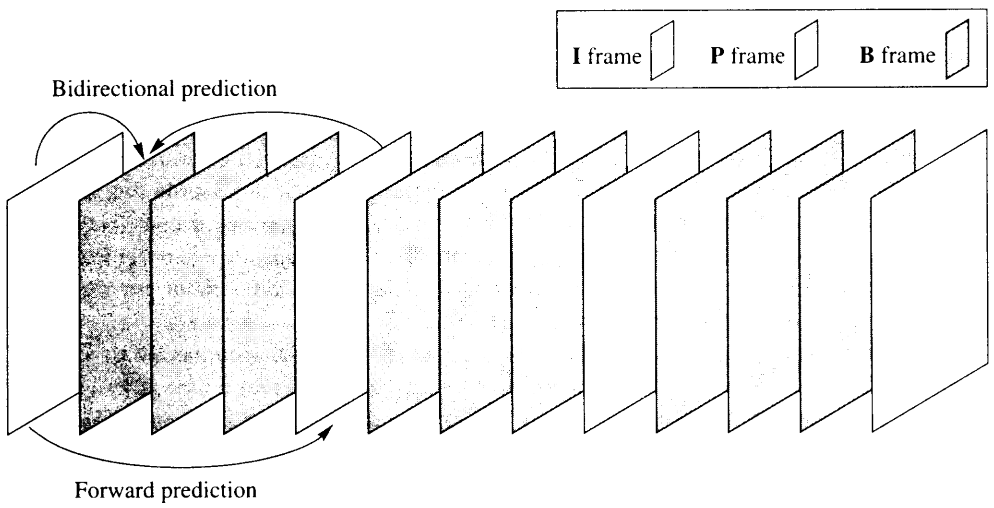

FIGURE 18. 14 A possible arrangement for a group of pictures.

TABLE 18.5 A typical sequence of frames in display order.

I	B	B	P	B	B	P	B	B	P	B	B	I
1	2	3	4	5	6	7	8	9	10	11	12	13

Because of the reliance of the **B** frame on future anchor frames, there are two different sequence orders. The *display order* is the sequence in which the video sequence is displayed to the user. A typical display order is shown in Table 18.5. Let us see how this sequence was generated. The first frame is an **I** frame, which is compressed without reference to any previous frame. The next frame to be compressed is the fourth frame. This frame is compressed using motion-compensated prediction from the first frame. Then we compress frame two, which is compressed using motion-compensated prediction from frame one and frame four. The third frame is also compressed using motion-compensated prediction from the first and fourth frames. The next frame to be compressed is frame seven, which uses motion-compensated prediction from frame four. This is followed by frames five and six, which are compressed using motion-compensated predictions from frames four and seven. Thus, there is a processing order that is quite different from the display order. The MPEG document calls this the *bitstream order*. The bitstream order for the sequence shown in Table 18.5 is given in Table 18.6. In terms of the bitstream order, the first frame in a GOP is always the **I** frame.

As we can see, unlike the ITU-T H.261 algorithm, the frame being predicted and the frame upon which the prediction is based are not necessarily adjacent. In fact, the number of frames between the frame being encoded and the frame upon which the prediction is based is variable. When searching for the best matching block in a neighboring frame, the region of search depends on assumptions about the amount of motion. More motion will lead to larger search areas than a small amount of motion. When the frame being predicted is always adjacent to the frame upon which the prediction is based, we can fix the search area based on our assumption about the amount of motion. When the number of frames between the frame being encoded and the prediction frame is variable, we make the search area a function of the distance between the two frames. While the MPEG standard does not specify the method used for motion compensation, it does recommend using a search area that grows with the distance between the frame being coded and the frame being used for prediction.

Once motion compensation has been performed, the block of prediction errors is transformed using the DCT and quantized, and the quantization labels are encoded. This procedure is the same as that recommended in the JPEG standard and is described in Chapter 12. The quantization tables used for the different frames are different and can be changed during the encoding process.

TABLE 18.6 A typical sequence of frames in bitstream order.

I	P	B	B	P	B	B	P	B	B	I	B	B
1	4	2	3	7	5	6	10	8	9	13	11	12

Rate control in the MPEG standard can be performed at the sequence level or at the level of individual frames. At the sequence level, any reduction in bit rate first occurs with the **B** frames because they are not essential for the encoding of other frames. At the level of the individual frames, rate control takes place in two steps. First, as in the case of the H.261 algorithm, the quantizer step sizes are increased. If this is not sufficient, then the higher-order frequency coefficients are dropped until the need for rate reduction is past.

The format for MPEG is very flexible. However, the MPEG committee has provided some suggested values for the various parameters. For MPEG-1 these suggested values are called the *constrained parameter bitstream* (CPB). The horizontal picture size is constrained to be less than or equal to 768 pixels, and the vertical size is constrained to be less than or equal to 576 pixels. More importantly, the pixel rate is constrained to be less than 396 macroblocks per frame if the frame rate is 25 frames per second or less, and 330 macroblocks per frame if the frame rate is 30 frames per second or less. The definition of a macroblock is the same as in the ITU-T H.261 recommendations. Therefore, this corresponds to a frame size of 352×288 pixels at the 25-frames-per-second rate, or a frame size of 352×240 pixels at the 30-frames-per-second rate. Keeping the frame at this size allows the algorithm to achieve bit rates of between 1 and 1.5 Mbits per second. When referring to MPEG-1 parameters, most people are actually referring to the CPB.

The MPEG-1 algorithm provides reconstructed images of VHS quality for moderate- to low-motion video sequences, and worse than VHS quality for high-motion sequences at rates of around 1.2 Mbits per second. As the algorithm was targeted to applications such as CD-ROM, there is no consideration of interlaced video. In order to expand the applicability of the basic MPEG algorithm to interlaced video, the MPEG committee provided some additional recommendations, the MPEG-2 recommendations.

18.9 The MPEG-2 Video Standard—H.262

While MPEG-1 was specifically proposed for digital storage media, the idea behind MPEG-2 was to provide a generic, application-independent standard. To this end, MPEG-2 takes a "tool kit" approach, providing a number of subsets, each containing different options from the set of all possible options contained in the standard. For a particular application, the user can select from a set of *profiles* and *levels*. The profiles define the algorithms to be used, while the levels define the constraints on the parameters. There are five profiles: *simple*, *main*, *snr-scalable* (where *snr* stands for signal-to-noise ratio), *spatially scalable*, and *high*. There is an ordering of the profiles; each higher profile is capable of decoding video encoded using all profiles up to and including that profile. For example, a decoder designed for profile *snr-scalable* could decode video that was encoded using profiles *simple*, *main*, and *snr-scalable*. The *simple* profile eschews the use of **B** frames. Recall that the **B** frames require the most computation to generate (forward and backward prediction), require memory to store the coded frames needed for prediction, and increase the coding delay because of the need to wait for "future" frames for both generation and reconstruction. Therefore, removal of the **B** frames makes the requirements simpler. The *main* profile is very much the algorithm we have discussed in the previous section. The *snr-scalable*, *spatially scalable*, and *high* profiles may use more than one bitstream to encode the video. The base bitstream is a lower-rate encoding of the video sequence. This bitstream could be decoded by itself to provide a reconstruction of the video sequence. The other bitstream is used to enhance the quality of the reconstruction. This layered approach is useful when transmitting video over a network, where some connections may only permit a lower rate. The base bitstream can be provided to these connections while providing the base and enhancement layers for a higher-quality reproduction over the links that can accommodate the higher bit rate. To understand the concept of layers, consider the following example.

Example 18.9.1:

Suppose after the transform we obtain a set of coefficients, the first eight of which are

$$29.75 \quad 6.1 \quad -6.03 \quad 1.93 \quad -2.01 \quad 1.23 \quad -0.95 \quad 2.11$$

Let us suppose we quantize this set of coefficients using a step size of 4. For simplicity we will use the same step size for all coefficients. Recall that the quantizer label is given by

$$l_{ij} = \left\lfloor \frac{\theta_{ij}}{Q'_{ij}} + 0.5 \right\rfloor \tag{18.8}$$

and the reconstructed value is given by

$$\hat{\theta}_{ij} = l_{ij} \times Q'_{ij}. \tag{18.9}$$

Using these equations and the fact that $Q'_{ij} = 4$, the reconstructed values of the coefficients are

$$28 \quad 8 \quad -8 \quad 0 \quad -4 \quad 0 \quad -0 \quad 4$$

The error in the reconstruction is

$$1.75 \quad -1.9 \quad 1.97 \quad 1.93 \quad 1.99 \quad 1.23 \quad -0.95 \quad -1.89$$

Now suppose we have some additional bandwidth made available to us. We can quantize the difference and send that to enhance the reconstruction. Suppose we used a step size of 2 to quantize the difference. The reconstructed values for this enhancement sequence would be

$$2 \quad -2 \quad 2 \quad 2 \quad 2 \quad 2 \quad 0 \quad -2$$

Adding this to the previous base-level reconstruction, we get an enhanced reconstruction of

$$30 \quad 6 \quad -6 \quad 2 \quad -2 \quad 2 \quad 0 \quad 2$$

which results in an error of

$$-0.25 \quad 0.1 \quad -0.03 \quad -0.07 \quad -0.01 \quad -0.77 \quad -0.95 \quad 0.11$$

The layered approach allows us to increase the accuracy of the reconstruction when bandwidth is available, while at the same time permitting a lower-quality reconstruction when there is not sufficient bandwidth for the enhancement. In other words, the quality is *scalable*. In this particular case, the error between the original and reconstruction decreases because of the enhancement. Because the signal-to-noise ratio is a measure of error, this can be called *snr-scalable*. If the enhancement layer contained a coded bitstream corresponding to frames that would occur between frames of the base layer, the system could be called *temporally scalable*. If the enhancement allowed an upsampling of the base layer, the system is *spatially scalable*. ♦

The levels are *low, main, high 1440*, and *high*. The *low* level corresponds to a frame size of 352 × 240, the *main* level corresponds to a frame size of 720 × 480, the *high 1440* level corresponds to a frame size of 1440 × 1152, and the *high* level corresponds to a frame size of 1920 × 1080. All levels are defined for a frame rate of 30 frames per second. There are many possible combinations of profiles and levels, not all of which are allowed in the MPEG-2 standard. Table 18.7 shows the allowable combinations [251]. A particular profile-level combination is denoted by *XX@YY* where *XX* is the two-letter abbreviation for the profile and *YY* is the two-letter abbreviation for the level. There are a large number of issues, such as bounds on parameters and decodability between different profile-level combinations, that we have not addressed here because they do not pertain to our main focus, compression (see the international standard [253] for these details).

Because MPEG-2 has been designed to handle interlaced video, there are field, based alternatives to the **I**, **P** and **B** frames. The **P** and **B** frames can be replaced by two **P** fields or two **B** fields. The **I** frame can be replaced by two **I** fields or an **I** field and a **P** field where the **P** field is obtained by predicting the bottom field by the top field. Because an 8 × 8 field block actually covers twice the spatial distance in the vertical direction as an 8 frame block, the zigzag scanning is adjusted to adapt to this imbalance. The scanning pattern for an 8 × 8 field block is shown in Figure 18.15

The most important addition from the point of view of compression in MPEG-2 is the addition of several new motion-compensated prediction modes: the field prediction and the dual prime prediction modes. MPEG-1 did not allow interlaced video. Therefore, there was no need for motion compensation algorithms based on fields. In the **P** frames, field predictions are obtained using one of the two most recently decoded fields. When the first field in a frame is being encoded, the prediction is based on the two fields from the

TABLE 18.7 Allowable profile-level combinations in MPEG-2.

	Simple Profile	Main Profile	SNR-Scalable Profile	Spatially Scalable Profile	High Profile
High Level		Allowed			Allowed
High 1440		Allowed		Allowed	Allowed
Main Level	Allowed	Allowed	Allowed		Allowed
Low Level		Allowed	Allowed		

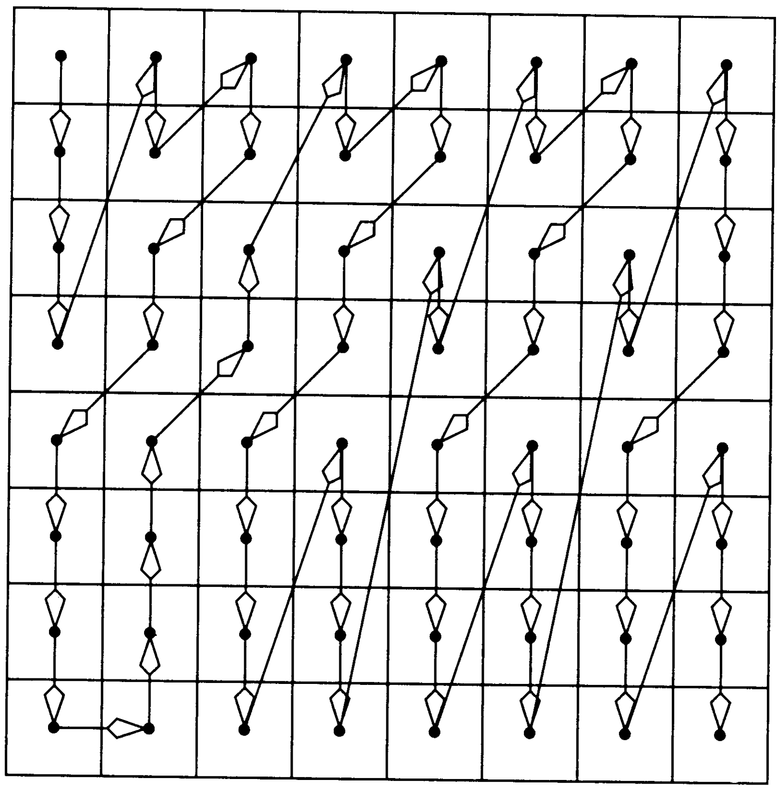

FIGURE 18. 15 **Scanning pattern for the DCT coefficients of a field block.**

previous frame. However, when the second field is being encoded, the prediction is based on the second field from the previous frame and the first field from the current frame. Information about which field is to be used for prediction is transmitted to the receiver. The field predictions are performed in a manner analogous to the motion-compensated prediction described earlier.

In addition to the regular frame and field prediction, MPEG-2 also contains two additional modes of prediction. One is the 16 × 8 motion compensation. In this mode, two predictions are generated for each macroblock, one for the top half and one for the bottom half. The other is called the dual prime motion compensation. In this technique, two predictions are formed for each field from the two recent fields. These predictions are averaged to obtain the final prediction.

18.9.1 The Grand Alliance HDTV Proposal

When the Federal Communications Commission (FCC) requested proposals for the HDTV standard, they received four proposals for digital HDTV from four consortia. After the evaluation phase, the FCC declined to pick a winner among the four, and instead suggested that all these consortia join forces and produce a single proposal. The resulting partnership has the exalted title of the "Grand Alliance." Currently, the specifications for the digital HDTV system use MPEG-2 as the compression algorithm. The Grand Alliance system uses the *main* profile of the MPEG-2 standard implemented at the *high* level.

18.10 ITU-T Recommendation H.263

The H.263 standard was developed to update the H.261 video conferencing standard with the experience acquired in the development of the MPEG and H.262 algorithms. The initial algorithm provided incremental improvement over H.261. After the development of the core algorithm, several optional updates were proposed, which significantly improved the compression performance. The standard with these optional components is sometimes referred to as H.263+ (or H.263++).

In the following sections we first describe the core algorithm and then describe some of the options. The standard focuses on noninterlaced video. The different picture formats addressed by the standard are shown in Table 18.8. The picture is divided into *Groups of Blocks* (GOBs) or slices. A Group of Blocks is a strip of pixels across the picture with a height that is a multiple of 16 lines. The number of multiples depends on the size of the picture, and the bottom-most GOB may have less than 16 lines. Each GOB is divided into macroblocks, which are defined as in the H.261 recommendation.

A block diagram of the baseline video coder is shown in Figure 18.16. It is very similar to Figure 18.10, the block diagram for the H.261 encoder. The only major difference is the ability to work with both predicted or **P** frames and intra or **I** frames. As in the case of H.261, the motion-compensated prediction is performed on a macroblock basis. The vertical and horizontal components of the motion vector are restricted to the range [−16, 15.5]. The transform used for representing the prediction errors in the case of the **P** frame and the pixels in the case of the **I** frames is the discrete cosine transform. The transform coefficients are quantized using uniform midtread quantizers. The DC coefficient of the intra block is quantized using a uniform quantizer with a step size of 8. There are 31 quantizers available for the quantization of all other coefficients with stepsizes ranging from 2 to 62. Apart from the DC coefficient of the intra block, all coefficients in a macroblock are quantized using the same quantizer.

TABLE 18.8 The standardized H.263 formats [254].

Picture format	Number of luminance pixels (columns)	Number of luminance lines (rows)	Number of chrominance pixels (columns)	Number of chrominance lines(rows)
sub-QCIF	128	96	64	48
QCIF	176	144	88	72
CIF	352	288	176	144
4CIF	704	576	352	288
16CIF	1408	1152	704	576

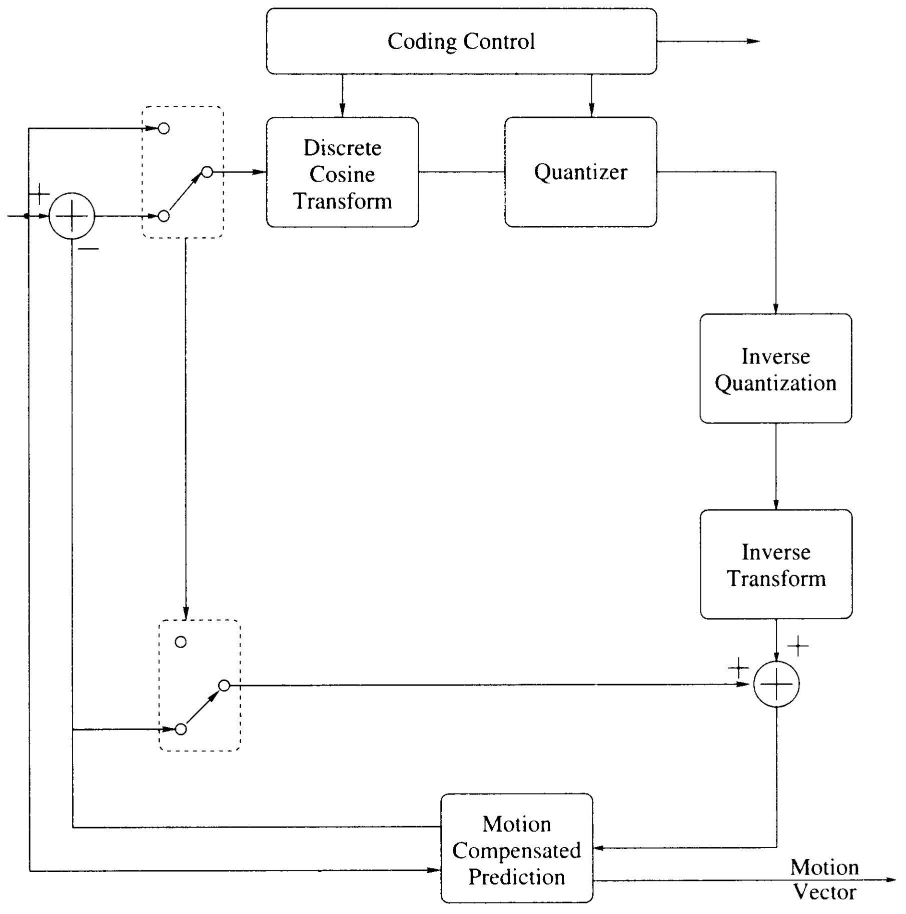

FIGURE 18.16 **A block diagram of the H.263 video compression algorithm.**

The motion vectors are differentially encoded. The prediction is the median of the motion vectors in the neighboring blocks. The H.263 recommendation allows half pixel motion compensation as opposed to only integer pixel compensation used in H.261. Notice that the sign of the component is encoded in the last bit of the variable length code, a "0" for positive values and a "1" for negative values. Two values that differ only in their sign differ only in the least significant bit.

The code for the quantized transform coefficients is indexed by three indicators. The first indicates whether the coefficient being encoded is the last nonzero coefficient in the zigzag scan. The second indicator is the number of zero coefficients preceding the coefficient being encoded, and the last indicates the absolute value of the quantized coefficient level. The sign bit is appended as the last bit of the variable length code.

Here we describe some of the optional modes of the H.263 recommendation. The first four options were part of the initial H.263 specification. The remaining options were added later and the resulting standard is sometimes referred to as the H.263+ standard.

18.10.1 Unrestricted Motion Vector Mode

In this mode the motion vector range is extended to $[-31.5, 31.5]$, which is particularly useful in improving the compression performance of the algorithm for larger picture sizes. The mode also allows motion vectors to point outside the picture. This is done by repeating the edge pixels to create the picture beyond its boundary.

18.10.2 Syntax-Based Arithmetic Coding Mode

In this mode the variable length codes are replaced with an arithmetic coder. The word length for the upper and lower limits is 16. The option specifies several different Cum Count tables that can be used for arithmetic coding. There are separate Cum Count tables for encoding motion vectors, intra DC component, and intra and inter coefficients.

18.10.3 Advanced Prediction Mode

In the baseline mode a single motion vector is sent for each macroblock. Recall that a macroblock consists of four 8×8 luminance blocks and two chrominance blocks. In the advanced prediction mode the encoder can send four motion vectors, one for each luminance block. The chrominance motion vectors are obtained by adding the four luminance motion vectors and dividing by 8. The resulting values are adjusted to the nearest half pixel position. This mode also allows for *Overlapped Block Motion Compensation (OBMC).* In this mode the motion vector is obtained by taking a weighted sum of the motion vector of the current block and two of the four vertical and horizontal neighboring blocks.

18.10.4 PB-frames and Improved PB-frames Mode

The PB frame consists of a **P** picture and a **B** picture in the same frame. The blocks for the **P** frame and the **B** frame are interleaved so that a macroblock consists of six blocks of a **P** picture followed by six blocks of a **B** picture. The motion vector for the **B** picture is derived from the motion vector for the **P** picture by taking into account the time difference between the **P** picture and the **B** picture. If the motion cannot be properly derived, a delta correction is included. The improved PB-frame mode updates the PB-frame mode to include forward, backward, and bidirectional prediction.

18.10.5 Advanced Intra Coding Mode

The coefficients for the **I** frames are obtained directly by transforming the pixels of the picture. As a result, there can be significant correlation between some of the coefficients of neighboring blocks. For example, the DC coefficient represents the average value of a block. It is very likely that the average value will not change significantly between blocks. The same may be true, albeit to a lesser degree, for the low-frequency horizontal and vertical coefficients. The advanced intra coding mode allows the use of this correlation by using coefficients from neighboring blocks for predicting the coefficients of the block being encoded. The prediction errors are then quantized and coded.

When this mode is used, the quantization approach and variable length codes have to be adjusted to adapt to the different statistical properties of the prediction errors. Furthermore, it might also become necessary to change the scan order. The recommendation provides alternate scanning patterns as well as alternate variable length codes and quantization strategies.

18.10.6 Deblocking Filter Mode

This mode is used to remove blocking effects from the 8×8 block edges. This smoothing of block boundaries allows for better prediction. This mode also permits the use of four motion vectors per macroblock and motion vectors that point beyond the edge of the picture.

18.10.7 Reference Picture Selection Mode

This mode is used to prevent error propagation by allowing the algorithm to use a picture other than the previous picture to perform prediction. The mode permits the use of a back-channel that the decoder uses to inform the encoder about the correct decoding of parts of the picture. If a part of the picture is not correctly decoded, it is not used for prediction. Instead, an alternate frame is selected as the reference frame. The information about which frame was selected as the reference frame is transmitted to the decoder. The number of possible reference frames is limited by the amount of frame memory available.

18.10.8 Temporal, SNR, and Spatial Scalability Mode

This is very similar to the scalability structures defined earlier for the MPEG-2 algorithm. Temporal scalability is achieved by using separate **B** frames, as opposed to the PB frames. SNR scalability is achieved using the kind of layered coding described earlier. Spatial scalability is achieved using upsampling.

18.10.9 Reference Picture Resampling

Reference picture resampling allows a reference picture to be "warped" in order to permit the generation of better prediction. It can be used to adaptively alter the resolution of pictures during encoding.

18.10.10 Reduced-Resolution Update Mode

This mode is used for encoding highly active scenes. The macroblock in this mode is assumed to cover an area twice the height and width of the regular macroblock. The motion vector is assumed to correspond to this larger area. Using this motion vector a predicted macroblock is created. The transform coefficients are decoded and then upsampled to create the expanded texture block. The predicted and texture blocks are then added to obtain the reconstruction.

18.10.11 Alternative Inter VLC Mode

The variable length codes for inter and intra frames are designed with different assumptions. In the case of the inter frames it is assumed that the values of the coefficients will be small and there can be large numbers of zero values between nonzero coefficients. This is a result of prediction which, if successfully employed, would reduce the magnitude of the differences, and hence the coefficients, and would also lead to large numbers of zero-valued coefficients. Therefore, coefficients indexed with large runs and small coefficient values are assigned shorter codes. In the case of the intra frames, the opposite is generally true. There is no prediction, therefore there is a much smaller probability of runs of zero-valued coefficients. Also, large-valued coefficients are quite possible. Therefore, coefficients indexed by small run values and larger coefficient values are assigned shorter codes. During periods of increased temporal activity, prediction is generally not as good and therefore the assumptions under which the variable length codes for the inter frames were created are violated. In these situations it is likely that the variable length codes designed for the intra frames are a better match. The alternative inter VLC mode allows for the use of the intra codes in these sitations, improving the compression performance. Note that the codewords used in intra and inter frame coding are the same. What is different is the interpretation. To detect the proper interpretation, the decoder first decodes the block assuming an inter frame codebook. If the decoding results in more than 64 coefficients it switches its interpretation.

18.10.12 Modified Quantization Mode

In this mode, along with changes in the signalling of changes in quantization parameters, the quantization process is improved in several ways. In the baseline mode, both the luminanace and chrominance components in a block are quantized using the same quantizer. In the modified quantization mode, the quantizer used for the luminance coefficients is different from the quantizer used for the chrominance component. This allows the quantizers to be more closely matched to the statistics of the input. The modified quantization mode also allows for the quantization of a wider range of coefficient values, preventing significant overload. If the coefficient exceeds the range of the baseline quantizer, the encoder sends an escape symbol followed by an 11-bit representation of the coefficient. This relaxation of the structured representation of the quantizer outputs makes it more likely that bit errors will be accepted as valid quantizer outputs. To reduce the chances of this happening, the mode prohibits "unreasonable" coefficient values.

18.10.13 Enhanced Reference Picture Selection Mode

Motion-compensated prediction is accomplished by searching the previous picture for a block similar to the block being encoded. The enhanced reference picture selection mode allows the encoder to search more than one picture to find the best match and then use the best-suited picture to perform motion-compensated prediction. Reference picture selection can be accomplished on a macroblock level. The selection of the pictures to be used for motion compensation can be performed in one of two ways. A sliding window of M pictures can be used and the last M decoded, with reconstructed pictures stored in a multipicture buffer. A more complex adaptive memory (not specified by the standard) can also be used in place of the simple sliding window. This mode significantly enhances the prediction, resulting in a reduction in the rate for equivalent quality. However, it also increases the computational and memory requirements on the encoder. This memory burden can be mitigated to some extent by assigning an unused label to pictures or portions of pictures. These pictures, or portions of pictures, then do not need to be stored in the buffer. This unused label can also be used as part of the adaptive memory control to manage the pictures that are stored jn the buffer.

18.11 ITU-T Recommendation H.264, MPEG-4 Part 10, Advanced Video Coding

As described in the previous section, the H.263 recommendation started out as an incremental improvement over H.261 and ended up with a slew of optional features, which in fact make the improvement over H.261 more than incremental. In H.264 we have a standard that started out with a goal of significant improvement over the MPEG-1/2 standards and achieved those goals. The standard, while initiated by ITU-T's Video Coding Experts Group (VCEG), ended up being a collaboration of the VCEG and ISO/IEC's MPEG committees which joined to form the Joint Video Team (JVT) in December of 2001 [255]. The collaboration of various groups in the development of this standard has also resulted in the richness of names. It is variously known as ITU-T H.264, MPEG-4 Part 10, MPEG-4 Advanced Video Coding (AVC), as well as the name under which it started its life, H.26L. We will just refer to it as H.264.

The basic block diagram looks very similar to the previous schemes. There are intra and inter pictures. The inter pictures are obtained by subtracting a motion compensated prediction from the original picture. The residuals are transformed into the frequency domain. The transform coefficients are scanned, quantized, and encoded using variable length codes. A local decoder reconstructs the picture for use in future predictions. The intra picture is coded without reference to past pictures.

While the basic block diagram is very similar to the previous standards the details are quite different. We will look at these details in the following sections. We begin by looking at the basic structural elements, then look at the decorrelation of the inter frames. The decorrelation process includes motion-compensated prediction and transformation of the prediction error. We then look at the decorrelation of the intra frames. This includes intra prediction modes and transforms used in this mode. We finally look at the different binary coding options.

The macroblock structure is the same as used in the other standards. Each macroblock consists of four 8×8 luminance blocks and two chrominance blocks. An integer number of sequential macroblocks can be put together to form a slice. In the previous standards the smallest subdivision of the macroblock was into its 8×8 component blocks. The H.264 standard allows 8×8 macroblock partitions to be further divided into sub-macroblocks of size 8×4, 4×8, and 4×4. These smaller blocks can be used for motion-compensated prediction, allowing for tracking of much finer details than is possible with the other standards. Along with the 8×8 partition, the macroblock can also be partitioned into two 8×16 or 16×8 blocks. In field mode the H.264 standard groups 16×8 blocks from each field to form a 16×16 macroblock.

18.11.1 Motion-Compensated Prediction

The H.264 standard uses its macroblock partitions to develop a tree-structured motion compensation algorithm. One of the problems with motion-compensated prediction has always been the selection of the size and shape of the block used for prediction. Different parts of a video scene will move at different rates in different directions or stay put. A smaller-size block allows tracking of diverse movement in the video frame, leading to better prediction and hence lower bit rates. However, more motion vectors need to be encoded and transmitted, using up valuable bit resources. In fact, in some video sequences the bits used to encode the motion vectors may make up most of the bits used. If we use small blocks, the number of motion vectors goes up, as does the bit rate. Because of the variety of sizes and shapes available to it, the H.264 algorithm provides a high level of accuracy and efficiency in its prediction. It uses small block sizes in regions of activity and larger block sizes in stationary regions. The availability of rectangular shapes allows the algorithm to focus more precisely on regions of activity.

The motion compensation is accomplished using quarter-pixel accuracy. To do this the reference picture is "expanded" by interpolating twice between neighboring pixels. This results in a much smoother residual. The prediction process is also enhanced by the use of filters on the 4 block edges. The standard allows for searching of up to 32 pictures to find the best matching block. The selection of the reference picture is done on the macroblock partion level, so all sub-macroblock partitions use the same reference picture.

As in H.263, the motion vectors are differentially encoded. The basic scheme is the same. The median values of the three neighboring motion vectors are used to predict the current motion vector. This basic strategy is modified if the block used for motion compensation is a 16×16, 16×8, or 8×16 block.

For **B** pictures, as in the case of the previous standards, two motion vectors are allowed for each macroblock or sub-macroblock partition. The prediction for each pixel is the weighted average of the two prediction pixels.

Finally, a $\mathbf{P}_{skip}$ type macroblock is defined for which 16×16 motion compensation is used and the prediction error is not transmitted. This type of macroblock is useful for regions of little change as well as for slow pans.

18.11.2 The Transform

Unlike the previous video coding schemes, the transform used is not an 8×8 DCT. For most blocks the transform used is a 4×4 integer DCT-like matrix. The transform matrix is given by

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & 2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix}$$

The inverse transform matrix is given by

$$H^I = \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix}$$

The transform operations can be implemented using addition and shifts. Multiplication by 2 is a single-bit left shift and division by 2 is a single-bit right shift. However, there is a price for the simplicity. Notice that the norm of the rows is not the same and the product of the forward and inverse transforms does not result in the identity matrix. This discrepancy is compensated for by the use of scale factors during quantization. There are several advantages to using a smaller integer transform. The integer nature makes the implementation simple and also avoids error accumulation in the transform process. The smaller size allows better representation of small stationary regions of the image. The smaller blocks are less likely to contain a wide range of values. Where there are sharp transitions in the blocks, any ringing effect is contained within a small number of pixels.

18.11.3 Intra Prediction

In the previous standards the **I** pictures were transform coded without any decorrelation. This meant that the number of bits required for the **I** frames is substantially higher than for the other pictures. When asked why he robbed banks, the notorious robber Willie Sutton is supposed to have said simply, "because that's where the money is." Because most of the bits in video coding are expended in encoding the **I** frame, it made a lot of sense for the JVT to look at improving the compression of the **I** frame in order to substantially reduce the bitrate.

The H.264 standard contains a number of spatial prediction modes. For 4×4 blocks there are nine prediction modes. Eight of these are summarized in Figure 18.17. The sixteen pixels in the block a–p are predicted using the thirteen pixels on the boundary (and extending from it).[1] The arrows corresponding to the mode numbers show the direction of prediction. For example, mode 0 corresponds to the downward pointing arrow. In this case pixel A is used to predict pixels a, e, i, m, pixel B is used to predict pixels b, f, j, n, pixel C is used

[1] The jump from pixel L to Q is a historical artifact. In an earlier version of the standard, pixels below L were also used in some prediction modes.

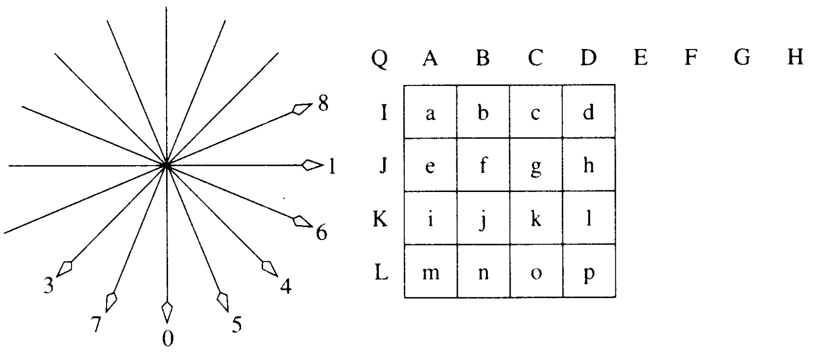

FIGURE 18.17 **Prediction modes for 4 × 4 intra prediction.**

to predict pixels c, g, k, o, and pixel D is used to predict pixels d, h, l, p. In mode 3, also called the diagonal down/left mode, B is used to predict a, C is used to predict b, e, pixel D is used to predict pixels c, f, i, pixel E is used to predict pixels d, g, j, m, pixel F is used to predict pixels h, k, n, pixel G is used to predict pixels l, o, and pixel H is used to predict pixel p. If pixels E, F, G, and H are not available, pixel D is repeated four times. Notice that no direction is availble for mode 2. This is called the DC mode, in which the average of the left and top boundary pixels is used as a prediction for all sixteen pixels in the 4×4 block. In most cases the prediction modes used for all the 4×4 blocks in a macroblock are heavily correlated. The standard uses this correlation to efficiently encode the mode information.

In smooth regions of the picture it is more convenient to use prediction on a macroblock basis. In case of the full macroblock there are four prediction modes. Three of them correspond to modes 0, 1, and 2 (vertical, horizontal, and DC). The fourth prediction mode is called the planar prediction mode. In this mode a three-parameter plane is fitted to the pixel values of the macroblock.

18.11.4 Quantization

The H.264 standard uses a uniform scalar quantizer for quantizing the coefficients. There are 52 scalar quantizers indexed by Q_{step}. The step size doubles for every sixth Q_{step}. The quantization incorporates scaling necessitated by the approximations used to make the transform simple. If $\alpha_{i,j}(Q_{step})$ are the weights for the $(i, j)^{th}$ coefficient then

$$l_{i,j} = sign(\theta_{i,j}) \left\lfloor \frac{|\theta_{i,j}|\alpha_{i,j}(Q_{step})}{Q_{step}} \right\rfloor$$

In order to broaden the quantization interval around the origin we add a small value in the numerator.

$$l_{i,j} = sign(\theta_{i,j}) \left\lfloor \frac{|\theta_{i,j}|\alpha_{i,j}(Q_{step}) + f(Q_{step})}{Q_{step}} \right\rfloor$$

In actual implementation we do away with divisions and the quantization is implemented as [255]

$$l_{i,j} = sign(\theta_{i,j})[|\theta_{i,j}|M(Q_M, r) + f2^{17+Q_E}] >> 17 + Q_E$$

where

$$Q_M = Q_{step}(mod 6)$$

$$Q_E = \left\lfloor \frac{Q_{step}}{6} \right\rfloor$$

$$r = \begin{cases} 0 & i, j \text{ even} \\ 1 & i, j \text{ odd} \\ 2 & \text{otherwise} \end{cases}$$

and M is given in Table 18.9

The inverse quantization is given by

$$\hat{\theta}_{i,j} = l_{i,j} S(Q_M, r) << Q_E$$

where S is given in Table 18.10

Prior to quantization, the transforms of the 16 × 16 luminance residuals and the 8 × 8 chrominance residuals of the macroblock-based intra prediction are processed to further remove redundancy. Recall that macroblock-based prediction is used in smooth regions of the **I** picture. Therefore, it is very likely that the DC coefficients of the 4 × 4 transforms

TABLE 18.9 $M(Q_M, r)$ **values in H.264.**

Q_M	$r = 0$	$r = 1$	$r = 2$
0	13107	5243	8066
1	11916	4660	7490
2	10082	4194	6554
3	9362	3647	5825
4	8192	3355	5243
5	7282	2893	4559

TABLE 18.10 $S(Q_M, r)$ **values in H.264.**

Q_M	$r = 0$	$r = 1$	$r = 2$
0	10	16	13
1	11	18	14
2	13	20	16
3	14	23	18
4	16	25	20
5	18	29	23

are heavily correlated. To remove this redundancy, a discrete Walsh-Hadamard transform is used on the DC coefficients in the macroblock. In the case of the luminance block, this is a 4×4 transform for the sixteen DC coefficients. The smaller chrominance block contains four DC coefficients, so we use a 2×2 discrete Walsh-Hadamard transform.

18.11.5 Coding

The H.264 standard contains two options for binary coding. The first uses exponential Golomb codes to encode the parameters and a context-adaptive variable length code (CAVLC) to encode the quantizer labels [255]. The second binarizes all the values and then uses a context-adaptive binary arithmetic code (CABAC) [256].

An exponential Golomb code for a positive number x can be obtained as the unary code for $M = \lfloor \log_2(x+1) \rfloor$ concatenated with the M bit natural binary code for $x+1$. The unary code for a number x is given as x zeros followed by a 1. The exponential Golomb code for zero is 1.

The quantizer labels are first scanned in a zigzag fashion. In many cases the last nonzero labels in the zigzag scan have a magnitude of 1. The number N of nonzero labels and the number T of trailing ones are used as an index into a codebook that is selected based on the values of N and T for the neighboring blocks. The maximum allowed value of T is 3. If the number of trailing labels with a magnitude of 1 is greater than 3, the remaining are encoded in the same manner as the other nonzero labels. The nonzero labels are then coded in reverse order. That is, the quantizer labels corresponding to the higher-frequency coefficients are encoded first. First the signs of the trailing 1s are encoded with 0s signifying positive values and 1s signifying negative values. Then the remaining quantizer labels are encoded in reverse scan order. After this, the total number of 0s in the scan between the beginning of the scan and the last nonzero label is encoded. This will be a number between 0 and $16-N$. Then the run of zeros before each label, starting with the last nonzero label is encoded until we run out of zeros or coefficients. The number of bits used to code each zero run will depend on the number of zeros remaining to be assigned.

In the second technique, which provides higher compression, all values are first converted to binary strings. This binarization is performed, depending on the data type, using unary codes, truncated unary codes, exponential Golomb codes, and fixed-length codes, plus five specific binary trees for encoding macroblock and sub-macroblock types. The binary string is encoded in one of two ways. Redundant strings are encoded using a context-adaptive binary arithmetic code. Binary strings that are random, such as the suffixes of the exponential Golomb codes, bypass the arithmetic coder. The arithmetic coder has 399 contexts available to it, with 325 of these contexts used for encoding the quantizer labels. These numbers include contexts for both frame and field slices. In a pure frame or field slice only 277 of the 399 context models are used. These context models are simply Cum_Count tables for use with a binary arithmetic coder. The H.264 standard recommends a multiplication-free implementation of binary arithmetic coding.

The H.264 standard is substantially more flexible than previous standards, with a much broader range of applications. In terms of performance, it claims a 50% reduction in bit rate over previous standards for equivalent perceptual quality [255].

18.12 MPEG-4 Part 2

The MPEG-4 standard provides a more abstract approach to the coding of multimedia. The standard views a multimedia "scene" as a collection of objects. These objects can be visual, such as a still background or a talking head, or aural, such as speech, music, background noise, and so on. Each of these objects can be coded independently using different techniques to generate separate elementary bitstreams. These bitstreams are multiplexed along with a scene description. A language called the Binary Format for Scenes (BIFS) based on the Virtual Reality Modeling Language (VRML) has been developed by MPEG for scene descriptions. The decoder can use the scene description and additional input from the user to combine or compose the objects to reconstruct the original scene or create a variation on it. The protocol for managing the elementary streams and their multiplexed version, called the Delivery Multimedia Integration Framework (DMIF), is an important part of MPEG-4. However, as our focus in this book is on compression, we will not discuss the protocol (for details, see the standard [213]).

A block diagram for the basic video coding algorithm is shown in Figure 18.18. Although shape coding occupies a very small portion of the diagram, it is a major part of the algorithm. The different objects that make up the scene are coded and sent to the multiplexer. The information about the presence of these objects is also provided to the motion-compensated

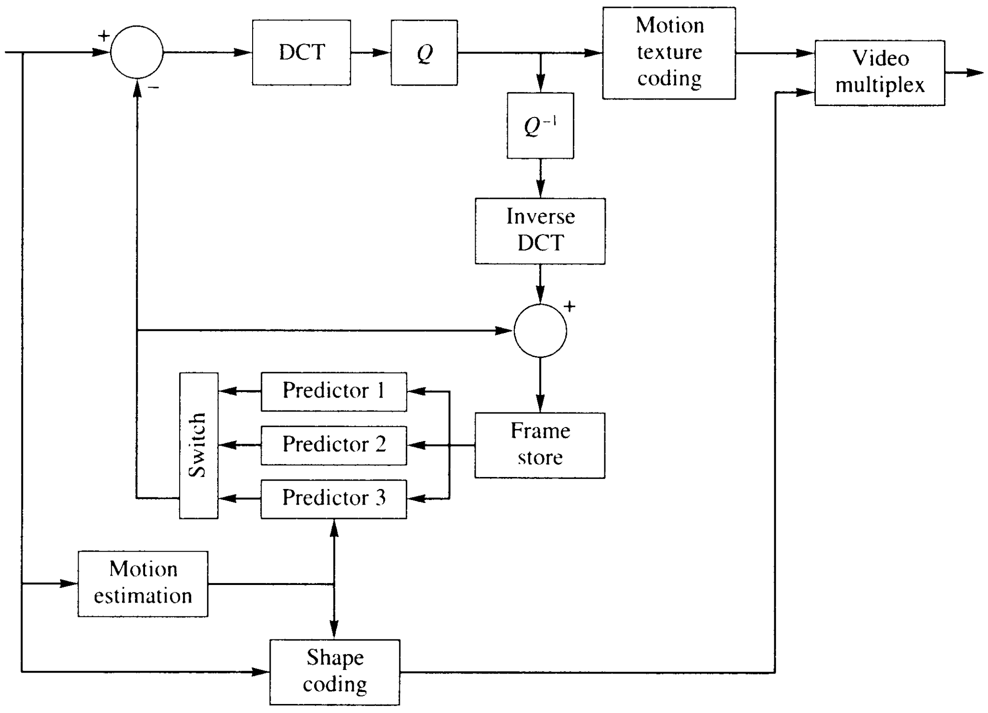

FIGURE 18.18 A block diagram for video coding.

predictor, which can use object-based motion compensation algorithms to improve the compression efficiency. What is left after the prediction can be transmitted using a DCT-based coder. The video coding algorithm can also use a background "sprite"—generally a large panoramic still image that forms the background for the video sequence. The sprite is transmitted once, and the moving foreground video objects are placed in front of different portions of the sprite based on the information about the background provided by the encoder.

The MPEG-4 standard also envisions the use of model-based coding, where a triangular mesh representing the moving object is transmitted followed by texture information for covering the mesh. Information about movement of the mesh nodes can then be transmitted to animate the video object. The texture coding technique suggested by the standard is the embedded zerotree wavelet (EZW) algorithm. In particular, the standard envisions the use of a facial animation object to render an animated face. The shape, texture, and expressions of the face are controlled using facial definition parameters (FDPs) and facial action parameters (FAPs). BIFS provides features to support custom models and specialized interpretation of FAPs.

The MPEG-2 standard allows for SNR and spatial scalability. The MPEG-4 standard also allows for object scalability, in which certain objects may not be sent in order to reduce the bandwidth requirement.

18.13 Packet Video

The increasing popularity of communication over networks has led to increased interest in the development of compression schemes for use over networks. In this section we look at some of the issues involved in developing video compression schemes for use over networks.

18.14 ATM Networks

With the explosion of information, we have also seen the development of new ways of transmitting the information. One of the most efficient ways of transferring information among a large number of users is the use of asynchronous transfer mode (ATM) technology. In the past, communication has usually taken place over dedicated channels; that is, in order to communicate between two points, a channel was dedicated only to transferring information between those two points. Even if there was no information transfer going on during a particular period, the channel could not be used by anyone else. Because of the inefficiency of this approach, there is an increasing movement away from it. In an ATM network, the users divide their information into packets, which are transmitted over channels that can be used by more than one user.

We could draw an analogy between the movement of packets over a communication network and the movement of automobiles over a road network. If we break up a message into packets, then the movement of the message over the network is like the movement of a number of cars on a highway system going from one point to the other. Although two cars may not occupy the same position at the same time, they can occupy the same road at the same time. Thus, more than one group of cars can use the road at any given time.

Furthermore, not all the cars in the group have to take the same route. Depending on the amount of traffic on the various roads that run between the origin of the traffic and the destination, different cars can take different routes. This is a more efficient utilization of the road than if the entire road was blocked off until the first group of cars completed its traversal of the road.

Using this analogy, we can see that the availability of transmission capacity, that is, the number of bits per second that we can transmit, is affected by factors that are outside our control. If at a given time there is very little traffic on the network, the available capacity will be high. On the other hand, if there is congestion on the network, the available capacity will be low. Furthermore, the ability to take alternate routes through the network also means that some of the packets may encounter congestion, leading to a variable amount of delay through the network. In order to prevent congestion from impeding the flow of vital traffic, networks will prioritize the traffic, with higher-priority traffic being permitted to move ahead of lower-priority traffic. Users can negotiate with the network for a fixed amount of guaranteed traffic. Of course, such guarantees tend to be expensive, so it is important that the user have some idea about how much high-priority traffic they will be transmitting over the network.

18.14.1 Compression Issues in ATM Networks

In video coding, this situation provides both opportunities and challenges. In the video compression algorithms discussed previously, there is a buffer that smooths the output of the compression algorithm. Thus, if we encounter a high-activity region of the video and generate more than the average number of bits per second, in order to prevent the buffer from overflowing, this period has to be followed by a period in which we generate fewer bits per second than the average. Sometimes this may happen naturally, with periods of low activity following periods of high activity. However, it is quite likely that this would not happen, in which case we have to reduce the quality by increasing the step size or dropping coefficients, or maybe even entire frames.

The ATM network, if it is not congested, will accommodate the variable rate generated by the compression algorithm. But if the network is congested, the compression algorithm will have to operate at a reduced rate. If the network is well designed, the latter situation will not happen too often, and the video coder can function in a manner that provides uniform quality. However, when the network is congested, it may remain so for a relatively long period. Therefore, the compression scheme should have the ability to operate for significant periods of time at a reduced rate. Furthermore, congestion might cause such long delays that some packets arrive after they can be of any use; that is, the frame they were supposed to be a part of might have already been reconstructed.

In order to deal with these problems, it is useful if the video compression algorithm provides information in a layered fashion, with a low-rate high-priority layer that can be used to reconstruct the video, even though the reconstruction may be poor, and low-priority enhancement layers that enhance the quality of the reconstruction. This is similar to the idea of progressive transmission, in which we first send a crude but low-rate representation of the image, followed by higher-rate enhancements. It is also useful if the bit rate required for the high-priority layer does not vary too much.

18.14.2 Compression Algorithms for Packet Video

Almost any compression algorithm can be modified to perform in the ATM environment, but some approaches seem more suited to this environment. We briefly present two approaches (see the original papers for more details).

One compression scheme that functions in an inherently layered manner is subband coding. In subband coding, the lower-frequency bands can be used to provide the basic reconstruction, with the higher-frequency bands providing the enhancement. As an example, consider the compression scheme proposed for packet video by Karlsson and Vetterli [257]. In their scheme, the video is divided into 11 bands. First, the video signal is divided into two temporal bands. Each band is then split into four spatial bands. The low-low band of the temporal low-frequency band is then split into four spatial bands. A graphical representation of this splitting is shown in Figure 18.19. The subband denoted 1 in the figure contains the basic information about the video sequence. Therefore, it is transmitted with the highest priority. If the data in all the other subbands are lost, it will still be possible to reconstruct the video using only the information in this subband. We can also prioritize the output of the other bands, and if the network starts getting congested and we are required to reduce our rate, we can do so by not transmitting the information in the lower-priority subbands. Subband 1 also generates the least variable data rate. This is very helpful when negotiating with the network for the amount of priority traffic.

Given the similarity of the ideas behind progressive transmission and subband coding, it should be possible to use progressive transmission algorithms as a starting point in the

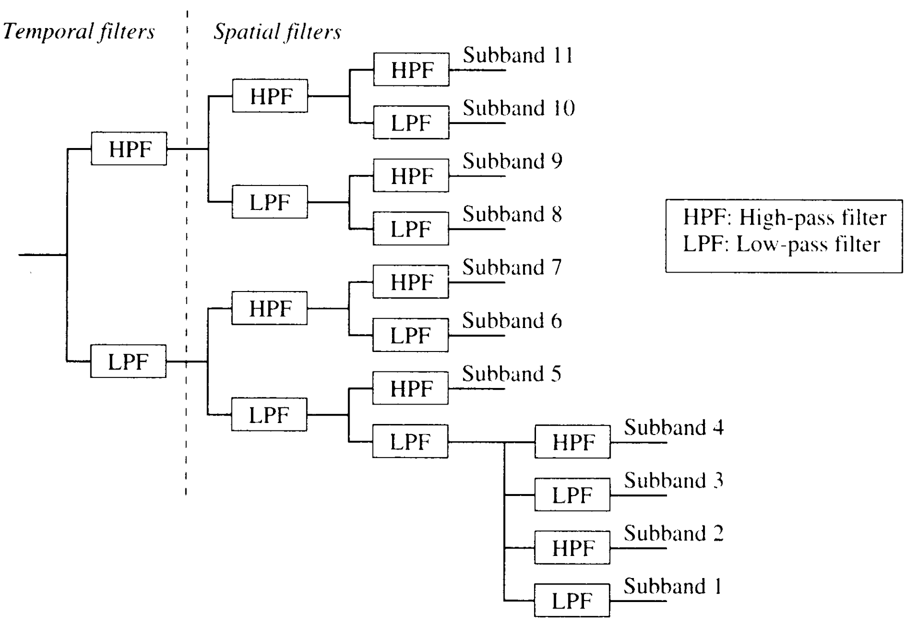

FIGURE 18.19 Analysis filter bank.

design of layered compression schemes for packet video. Chen, Sayood, and Nelson [258] use a DCT-based progressive transmission scheme [259] to develop a compression algorithm for packet video. In their scheme, they first encode the difference between the current frame and the prediction for the current frame using a 16×16 DCT. They only transmit the DC coefficient and the three lowest-order AC coefficients to the receiver. The coded coefficients make up the highest-priority layer.

The reconstructed frame is then subtracted from the original. The sum of squared errors is calculated for each 16×16 block. Blocks with squared error greater than a prescribed threshold are subdivided into four 8×8 blocks, and the coding process is repeated using an 8×8 DCT. The coded coefficients make up the next layer. Because only blocks that fail to meet the threshold test are subdivided, information about which blocks are subdivided is transmitted to the receiver as side information.

The process is repeated with 4×4 blocks, which make up the third layer, and 2×2 blocks, which make up the fourth layer. Although this algorithm is a variable-rate coding scheme, the rate for the first layer is constant. Therefore, the user can negotiate with the network for a fixed amount of high-priority traffic. In order to remove the effect of delayed packets from the prediction, only the reconstruction from the higher-priority layers is used for prediction.

This idea can be used with many different progressive transmission algorithms to make them suitable for use over ATM networks.

18.15 Summary

In this chapter we described a number of different video compression algorithms. The only new information in terms of compression algorithms was the description of motion-compensated prediction. While the compression algorithms themselves have already been studied in previous chapters, we looked at how these algorithms are used under different requirements. The three scenarios that we looked at are teleconferencing, asymmetric applications such as broadcast video, and video over packet networks. Each application has slightly different requirements, leading to different ways of using the compression algorithms. We have by no means attempted to cover the entire area of video compression. However, by now you should have sufficient background to explore the subject further using the following list as a starting point.

Further Reading

1. An excellent source for information about the technical issues involved with digital video is the book *The Art of Digital Video*, by J. Watkinson [260].
2. The MPEG-1 standards document [252], "Information Technology—Coding of Moving Pictures and Associated Audio for Digital Storage Media Up to about 1.5 Mbit/s," has an excellent description of the video compression algorithm.

3. Detailed information about the MPEG-1 and MPEG-2 video standards can also be found in *MPEG Video Compression Standard*, by J.L. Mitchell, W.B. Pennebaker, C.E. Fogg, and D.J. LeGall [261].
4. To find more on model-based coding, see "Model Based Image Coding: Advanced Video Coding Techniques for Very Low Bit-Rate Applications," by K. Aizawa and T.S. Huang [250], in the February 1995 issue of the *Proceedings of the IEEE*.
5. A good place to begin exploring the various areas of research in packet video is the June 1989 issue of the *IEEE Journal on Selected Areas of Communication*.
6. The MPEG 1/2 and MPEG 4 standards are covered in an accesible manner in the book. *The MPEG Handbook* by J. Watkinson [261]. Focal press 2001.
7. A good source for information about H.264 and MPEG-4 is H.264 and MPEG-4 video compression, by I.E.G. Richardson. Wiley, 2003.

18.16 Projects and Problems

1. **(a)** Take the DCT of the Sinan image and plot the average squared value of each coefficient.

   (b) Circularly shift each line of the image by eight pixels. That is, $new\_image[i, j] = old\_image[i, j + 8 \pmod{256}]$. Take the DCT of the difference and plot the average squared value of each coefficient.

   (c) Compare the results in parts (a) and (b) above. Comment on the differences.

Probability and Random Processes

In this appendix we will look at some of the concepts relating to probability and random processes that are important in the study of systems. Our coverage will be highly selective and somewhat superficial, but enough to use probability and random processes as a tool in understanding data compression systems.

A.1 Probability

There are several different ways of defining and thinking about probability. Each approach has some merit; perhaps the best approach is the one that provides the most insight into the problem being studied.

A.1.1 Frequency of Occurrence

The most common way that most people think about probability is in terms of outcomes, or sets of outcomes, of an experiment. Let us suppose we conduct an experiment E that has N possible outcomes. We conduct the experiment n_T times. If the outcome ω_i occurs n_i times, we say that the frequency of occurrence of the outcome ω_i is $\frac{n_i}{n_T}$. We can then define the probability of occurrence of the outcome ω_i as

$$P(\omega_i) = \lim_{n_T \to \infty} \frac{n_i}{n_T}.$$

In practice we do not have the ability to repeat an experiment an infinite number of times, so we often use the frequency of occurrence as an approximation to the probability. To make this more concrete consider a specific experiment. Suppose we turn on a television 1,000,000 times. Of these times, 800,000 times we turn the television on during a commercial and 200,000 times we turn it on and we don't get a commercial. We could say the frequency of occurrence, or the estimate of the probability, of turning on a television set in the middle of a commercial is 0.8. Our experiment E here is the turning on a television set, and the outcomes are *commercial* and *no commercial*. We could have been more careful with noting what was on when we turned on the television set and noticed whether the program was a news program (2000 times), a newslike program (20,000 times), a comedy program (40,000 times), an adventure program (18,000 times), a variety show (20,000 times), a talk show (90,000 times), or a movie (10,000 times), and whether the commercial was for products or services. In this case the outcomes would be *product commercial, service commercial, comedy, adventure, news, pseudonews, variety, talk show*, and *movie*. We could then define an *event* as a set of outcomes. The event *commercial* would consist of the outcomes *product commercial, service commercial*; the event *no commercial* would consist of the outcomes *comedy, adventure, news, pseudonews, variety, talk show, movie*. We could also define other events such as *programs that may contain news*. This set would contain the outcomes *news, pseudonews*, and *talk shows*, and the frequency of occurrence of this set is 0.112.

Formally, when we define an experiment E, associated with the experiment we also define a *sample space* S that consists of the *outcomes* $\{\omega_i\}$. We can then combine these outcomes into sets that are called *events*, and assign probabilities to these events. The largest subset of S (event) is S itself, and the probability of the event S is simply the probability that the experiment will have an outcome. The way we have defined things, this probability is one; that is, $P(S) = 1$.

A.1.2 A Measure of Belief

Sometimes the idea that the probability of an event is obtained through the repetitions of an experiment runs into trouble. What, for example, is the probability of your getting from Logan Airport to a specific address in Boston in a specified period of time? The answer depends on a lot of different factors, including your knowledge of the area, the time of day, the condition of your transport, and so on. You cannot conduct an experiment and get your answer because the moment you conduct the experiment, the conditions have changed, and the answer will now be different. We deal with this situation by defining a priori and a posteriori probabilities. The a priori probability is what you think or believe the probability to be before certain information is received or certain events take place; the a posteriori probability is the probability after you have received further information. Probability is no longer as rigidly defined as in the frequency of occurrence approach but is a somewhat more fluid quantity, the value of which changes with changing experience. For this approach to be useful we have to have a way of describing how the probability evolves with changing information. This is done through the use of *Bayes' rule*, named after the person who first described it. If $P(A)$ is the a priori probability of the event A and $P(A|B)$ is the a posteriori probability of the event A given that the event B has occurred, then

$$P(A|B) = \frac{P(A, B)}{P(B)} \tag{A.1}$$

where $P(A, B)$ is the probability of the event A *and* the event B occurring. Similarly,

$$P(B|A) = \frac{P(A, B)}{P(A)}. \tag{A.2}$$

Combining (A.1) and (A.2) we get

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}. \tag{A.3}$$

If the events A and B do not provide any information about each other, it would be reasonable to assume that

$$P(A|B) = P(A)$$

and therefore from (A.1),

$$P(A, B) = P(A)P(B). \tag{A.4}$$

Whenever (A.4) is satisfied, the events A and B are said to be *statistically independent*, or simply *independent*.

Example A.1.1:

A very common channel model used in digital communication is the *binary symmetric channel*. In this model the input is a random experiment with outcomes 0 and 1. The output of the channel is another random event with two outcomes 0 and 1. Obviously, the two outcomes are connected in some way. To see how, let us first define some events:

A: Input is 0
B: Input is 1
C: Output is 0
D: Output is 1

Let's suppose the input is equally likely to be a 1 or a 0. So $P(A) = P(B) = 0.5$. If the channel was perfect, that is, you got out of the channel what you put in, then we would have

$$P(C|A) = P(D|B) = 1$$

and

$$P(C|B) = P(D|A) = 0.$$

With most real channels this system is seldom encountered, and generally there is a small probability ϵ that the transmitted bit will be received in error. In this case, our probabilities would be

$$\begin{aligned} P(C|A) &= P(D|B) = 1 - \epsilon \\ P(C|B) &= P(D|A) = \epsilon. \end{aligned}$$

How do we interpret $P(C)$ and $P(D)$? These are simply the probability that at any given time the output is a 0 or a 1. How would we go about computing these probabilities given the available information? Using (A.1) we can obtain $P(A, C)$ and $P(B, C)$ from $P(C|A)$, $P(C|B)$, $P(A)$, and $P(B)$. These are the probabilities that the input is 0 and the output is 1, and the input is 1 and the output is 1. The event C—that is, the output is 1—will occur only when one of the two *joint* events occurs, therefore,

$$P(C) = P(A, C) + P(B, C).$$

Similarly,

$$P(D) = P(A, D) + P(B, D).$$

Numerically, this comes out to be

$$P(C) = P(D) = 0.5.$$ ♦

A.1.3 The Axiomatic Approach

Finally, there is an approach that simply defines probability as a measure, without much regard for physical interpretation. We are very familiar with measures in our daily lives. We talk about getting a 9-foot cable or a pound of cheese. Just as length and width measure the extent of certain physical quantities, probability measures the extent of an abstract quantity, a set. The thing that probability measures is the "size" of the event set. The probability measure follows similar rules to those followed by other measures. Just as the length of a physical object is always greater than or equal to zero, the probability of an event is always greater than or equal to zero. If we measure the length of two objects that have no overlap, then the combined length of the two objects is simply the sum of the lengths of the individual objects. In a similar manner the probability of the union of two events that do not have any outcomes in common is simply the sum of the probability of the individual events. So as to keep this definition of probability in line with the other definitions, we normalize this quantity by assigning the largest set, which is the sample space S, the size of 1. Thus, the probability of an event always lies between 0 and 1. Formally, we can write these rules down as the three *axioms* of probability.

Given a sample space S:

- *Axiom 1:* If A is an event in S, then $P(A) \geq 0$.
- *Axiom 2:* The probability of the sample space is 1; that is, $P(S) = 1$.
- *Axiom 3:* If A and B are two events in S and $A \cap B = \phi$, then $P(A \cup B) = P(A) + P(B)$.

Given these three axioms we can come up with all the other rules we need. For example, suppose A^c is the complement of A. What is the probability of A^c? We can get the answer by using Axiom 2 and Axiom 3. We know that

$$A^c \cup A = S$$

and Axiom 2 tells us that $P(S) = 1$, therefore,

$$P(A^c \cup A) = 1. \tag{A.5}$$

We also know that $A^c \cap A = \phi$, therefore, from Axiom 3

$$P(A^c \cup A) = P(A^c) + P(A). \tag{A.6}$$

Combining equations (A.5) and (A.6), we get

$$P(A^c) = 1 - P(A). \tag{A.7}$$

Similarly, we can use the three axioms to obtain the probability of $A \cup B$ when $A \cap B \neq \phi$ as

$$P(A \cup B) = P(A) + P(B) - P(A \cap B). \tag{A.8}$$

In all of the above we have been using two events A and B. We can easily extend these rules to more events.

Example A.1.2:

Find $P(A \cup B \cup C)$ when $A \cap B = A \cap C = \phi$, and $B \cup C \neq \phi$.
Let

$$D = B \cup C.$$

Then

$$A \cap C = \phi, \quad A \cap B = \phi \qquad \Rightarrow \qquad A \cap D = \phi.$$

Therefore, from Axiom 3,

$$P(A \cup D) = P(A) + P(D)$$

and using (A.8)

$$P(D) = P(B) + P(C) - P(B \cap C).$$

Combining everything, we get

$$P(A \cup B \cup C) = P(A) + P(B) + P(C) - P(B \cap C).$$ ♦

The axiomatic approach is especially useful when an experiment does not have discrete outcomes. For example, if we are looking at the voltage on a telephone line, the probability of any specific value of the voltage is zero because there are an uncountably infinite number of different values that the voltage can take, and we can assign nonzero values to only a countably infinite number. Using the axiomatic approach, we can view the sample space as the range of voltages, and events as subsets of this range.

We have given three different interpretations of probability, and in the process described some rules by which probabilities can be manipulated. The rules described here (such as

Bayes' rule, the three axioms, and the other rules we came up with) work the same way regardless of which interpretation you hold dear. The purpose of providing you with three different interpretations was to provide you with a variety of perspectives with which to view a given situation. For example, if someone says that the probability of a head when you flip a coin is 0.5, you might interpret that number in terms of repeated experiments (*if I flipped the coin 1000 times, I would expect to get 500 heads*). However, if someone tells you that the probability of your getting killed while crossing a particular street is 0.1, you might wish to interpret this information in a more subjective manner. The idea is to use the interpretation that gives you the most insight into a particular problem, while remembering that your interpretation will not change the mathematics of the situation.

Now that have expended a lot of verbiage to say what probability is, let's spend a few lines saying what it is not. Probability does not imply certainty. When we say that the probability of an event is one, this does not mean that event *will* happen. On the other hand, when we say that the probability of an event is zero, that does not mean that event *won't* happen. Remember, mathematics only models reality, it is *not* reality.

A.2 Random Variables

When we are trying to mathematically describe an experiment and its outcomes, it is much more convenient if the outcomes are numbers. A simple way to do this is to define a mapping or function that assigns a number to each outcome. This mapping or function is called a *random variable*. To put that more formally: Let S be a sample space with outcomes $\{\omega_i\}$. Then the random variable X is a mapping

$$X : S \rightarrow \mathcal{R} \tag{A.9}$$

where $\mathcal{R}$ denotes the real number line. Another way of saying the same thing is

$$X(\omega) = x; \qquad \omega \in S, x \in \mathcal{R}. \tag{A.10}$$

The random variable is generally represented by an uppercase letter, and this is the convention we will follow. The value that the random variable takes on is called the *realization* of the random variable and is represented by a lowercase letter.

Example A.2.1:

Let's take our television example and rewrite it in terms of a random variable X:

$$\begin{aligned} X(\textit{product commercial}) &= 0 \\ X(\textit{service commercial}) &= 1 \\ X(\textit{news}) &= 2 \\ X(\textit{pseudonews}) &= 3 \\ X(\textit{talk show}) &= 4 \end{aligned}$$

$$\begin{aligned} X(variety) &= 5 \\ X(comedy) &= 6 \\ X(adventure) &= 7 \\ X(movie) &= 8 \end{aligned}$$

Now, instead of talking about the probability of certain programs, we can talk about the probability of the random variable X taking on certain values or ranges of values. For example, $P(X(\omega) \leq 1)$ is the probability of seeing a commercial when the television is turned on (generally, we drop the argument and simply write this as $P(X \leq 1)$). Similarly, the $P(programs\ that\ may\ contain\ news)$ could be written as $P(1 < X \leq 4)$, which is substantially less cumbersome. ♦

A.3 Distribution Functions

Defining the random variable in the way that we did allows us to define a special probability $P(X \leq x)$. This probability is called the *cumulative distribution function (cdf)* and is denoted by $F_X(x)$, where the random variable is the subscript and the realization is the argument. One of the primary uses of probability is the modeling of physical processes, and we will find the cumulative distribution function very useful when we try to describe or model different random processes. We will see more on this later.

For now, let us look at some of the properties of the *cdf*:

Property 1: $0 \leq F_X(x) \leq 1$. This follows from the definition of the *cdf*.

Property 2: The *cdf* is a monotonically nondecreasing function. That is,

$$x_1 \geq x_2 \quad \Rightarrow \quad F_X(x_1) \geq F_X(x_2).$$

To show this simply write the *cdf* as the sum of two probabilities:

$$\begin{aligned} F_X(x_1) &= P(X \leq x_1) = P(X \leq x_2) + P(x_2 < X \leq x_1) \\ &= F_X(x_2) + P(x_1 < X \leq x_2) \geq F_X(x_2) \end{aligned}$$

Property 3:

$$\lim_{n \to \infty} F_X(x) = 1.$$

Property 4:

$$\lim_{n \to -\infty} F_X(x) = 0.$$

Property 5: If we define

$$F_X(x^-) = P(X < x)$$

then

$$P(X = x) = F_X(x) - F_X(x^-).$$

Example A.3.1:

Assuming that the frequency of occurrence was an accurate estimate of the probabilities, let us obtain the *cdf* for our television example:

$$F_X(x) = \begin{cases} 0 & x < 0 \\ 0.4 & 0 \le x < 1 \\ 0.8 & 1 \le x < 2 \\ 0.802 & 2 \le x < 3 \\ 0.822 & 3 \le x < 4 \\ 0.912 & 4 \le x < 5 \\ 0.932 & 5 \le x < 6 \\ 0.972 & 6 \le x < 7 \\ 0.99 & 7 \le x < 8 \\ 1.00 & 8 \le x \end{cases}$$

♦

Notice a few things about this *cdf*. First, the *cdf* consists of step functions. This is characteristic of discrete random variables. Second, the function is continuous from the right. This is due to the way the *cdf* is defined.

The *cdf* is somewhat different when the random variable is a continuous random variable. For example, if we sampled a speech signal and then took differences of the samples, the resulting random process would have a *cdf* that would look something like this:

$$F_X(x) = \begin{cases} \frac{1}{2}e^{2x} & x \le 0 \\ 1 - \frac{1}{2}e^{-2x} & x > 0. \end{cases}$$

The thing to notice in this case is that because $F_X(x)$ is continuous

$$P(X = x) = F_X(x) - F_X(x^-) = 0.$$

We can also have processes that have distributions that are continuous over some ranges and discrete over others.

Along with the cumulative distribution function, another function that also comes in very handy is the *probability density function (pdf)*. The *pdf* corresponding to the *cdf* $F_X(x)$ is written as $f_X(x)$. For continuous *cdf*s, the *pdf* is simply the derivative of the *cdf*. For the discrete random variables, taking the derivative of the *cdf* would introduce delta functions, which have problems of their own. So in the discrete case, we obtain the *pdf* through differencing. It is somewhat awkward to have different procedures for obtaining the same function for different types of random variables. It is possible to define a rigorous unified procedure for getting the *pdf* from the *cdf* for all kinds of random variables. However, in order to do so, we need some familiarity with measure theory, which is beyond the scope of this appendix. Let us look at some examples of *pdf*s.

Example A.3.2:

For our television scenario:

$$f_X(x) = \begin{cases} 0.4 & \text{if } X = 0 \\ 0.4 & \text{if } X = 1 \\ 0.002 & \text{if } X = 2 \\ 0.02 & \text{if } X = 3 \\ 0.09 & \text{if } X = 4 \\ 0.02 & \text{if } X = 5 \\ 0.04 & \text{if } X = 6 \\ 0.018 & \text{if } X = 7 \\ 0.01 & \text{if } X = 8 \\ 0 & \text{otherwise} \end{cases}$$

♦

Example A.3.3:

For our speech example, the *pdf* is given by

$$f_X(x) = \frac{1}{2}e^{-2|x|}.$$

♦

A.4 Expectation

When dealing with random processes, we often deal with average quantities, like the signal power and noise power in communication systems, and the mean time between failures in various design problems. To obtain these average quantities, we use something called an *expectation operator*. Formally, the expectation operator $E[\,]$ is defined as follows: The *expected value* of a random variable X is given by

$$E[X] = \sum_i x_i P(X = x_i) \tag{A.11}$$

when X is a discrete random variable with realizations $\{x_i\}$ and by

$$E[X] = \int_{-\infty}^{\infty} x f_X(x)dx \tag{A.12}$$

where $f_X(x)$ is the *pdf* of X.

The expected value is very much like the average value and, if the frequency of occurrence is an accurate estimate of the probability, is identical to the average value. Consider the following example:

Example A.4.1:

Suppose in a class of 10 students the grades on the first test were

$$10, 9, 8, 8, 7, 7, 7, 6, 6, 2$$

The average value is $\frac{70}{10}$, or 7. Now let's use the frequency of occurrence approach to estimate the probabilities of the various grades. (Notice in this case the random variable is an identity mapping, i.e., $X(\omega) = \omega$.) The probability estimate of the various values the random variable can take on are

$$P(10) = P(9) = P(2) = 0.1, \quad P(8) = P(6) = 0.2, \quad P(7) = 0.3,$$
$$P(6) = P(5) = P(4) = P(3) = P(1) = P(0) = 0$$

The expected value is therefore given by

$$E[X] = (0)(0) + (0)(1) + (0.1)(2) + (0)(3) + (0)(4) + (0)(5) + (0.2)(6)$$
$$+ (0.3)(7) + (0.2)(8) + (0.1)(9) + (0.1)(10) = 7. \qquad \blacklozenge$$

It seems that the expected value and the average value *are* exactly the same! But we have made a rather major assumption about the accuracy of our probability estimate. In general the relative frequency is not exactly the same as the probability, and the average expected values are different. To emphasize this difference and similarity, the expected value is sometimes referred to as the *statistical average*, while our everyday average value is referred to as the *sample average*.

We said at the beginning of this section that we are often interested in things such as signal power. The average signal power is often defined as the average of the signal squared. If we say that the random variable is the signal value, then this means that we have to find the expected value of the square of the random variable. There are two ways of doing this. We could define a new random variable $Y = X^2$, then find $f_Y(y)$ and use (A.12) to find $E[Y]$. An easier approach is to use the *fundamental theorem of expectation*, which is

$$E[g(X)] = \sum_i g(x_i)P(X = x_i) \tag{A.13}$$

for the discrete case, and

$$E[g(X)] = \int_{-\infty}^{\infty} g(x) f_X(x) dx \tag{A.14}$$

for the continuous case.

The expected value, because of the way it is defined, is a linear operator. That is,

$$E[\alpha X + \beta Y] = \alpha E[X] + \beta E[Y], \qquad \alpha \text{ and } \beta \text{ are constants.}$$

You are invited to verify this for yourself.

There are several functions $g()$ whose expectations are used so often that they have been given special names.

A.4.1 Mean

The simplest and most obvious function is the identity mapping $g(X) = X$. The expected value $E(X)$ is referred to as the *mean* and is symbolically referred to as μ_X. If we take a random variable X and add a constant value to it, the mean of the new random process is simply the old mean plus the constant. Let

$$Y = X + a$$

where a is a constant value. Then

$$\mu_Y = E[Y] = E[X+a] = E[X] + E[a] = \mu_X + a.$$

A.4.2 Second Moment

If the random variable X is an electrical signal, the total power in this signal is given by $E[X^2]$, which is why we are often interested in it. This value is called the *second moment* of the random variable.

A.4.3 Variance

If X is a random variable with mean μ_X, then the quantity $E[(X-\mu_X)^2]$ is called the *variance* and is denoted by σ_X^2. The square root of this value is called the *standard deviation* and is denoted by σ. The variance and the standard deviation can be viewed as a measure of the "spread" of the random variable. We can show that

$$\sigma_X^2 = E[X^2] - \mu_X^2.$$

If $E[X^2]$ is the total power in a signal, then the variance is also referred to as the total AC power.

A.5 Types of Distribution

There are several specific distributions that are very useful when describing or modeling various processes.

A.5.1 Uniform Distribution

This is the distribution of ignorance. If we want to model data about which we know nothing except its range, this is the distribution of choice. This is not to say that there are not times when the uniform distribution is a good match for the data. The *pdf* of the uniform distribution is given by

$$f_X(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \le X \le b \\ 0 & \text{otherwise.} \end{cases} \tag{A.15}$$

The mean of the uniform distribution can be obtained as

$$\mu_X = \int_a^b x\frac{1}{b-a}dx = \frac{b+a}{2}.$$

Similarly, the variance of the uniform distribution can be obtained as

$$\sigma_X^2 = \frac{(b-a)^2}{12}.$$

Details are left as an exercise.

A.5.2 Gaussian Distribution

This is the distribution of choice in terms of mathematical tractability. Because of its form, it is especially useful with the squared error distortion measure. The probability density function for a random variable with a Gaussian distribution, and mean μ and variance σ^2, is

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp - \frac{(x-\mu)^2}{2\sigma^2} \tag{A.16}$$

where the mean of the distribution is μ and the variance is σ^2.

A.5.3 Laplacian Distribution

Many sources that we will deal with will have probability density functions that are quite peaked at zero. For example, speech consists mainly of silence; therefore, samples of speech will be zero or close to zero with high probability. Image pixels themselves do not have any attraction to small values. However, there is a high degree of correlation among pixels. Therefore, a large number of the pixel-to-pixel differences will have values close to zero. In these situations, a Gaussian distribution is not a very close match to the data. A closer match is the Laplacian distribution, which has a *pdf* that is peaked at zero. The density function for a zero mean random variable with Laplacian distribution and variance σ^2 is

$$f_X(x) = \frac{1}{\sqrt{2\sigma^2}} \exp \frac{-\sqrt{2}\,|x|}{\sigma}. \tag{A.17}$$

A.5.4 Gamma Distribution

A distribution with a *pdf* that is even more peaked, though considerably less tractable than the Laplacian distribution, is the Gamma distribution. The density function for a Gamma distributed random variable with zero mean and variance σ^2 is given by

$$f_X(x) = \frac{\sqrt[4]{3}}{\sqrt{8\pi\sigma\,|x|}} \exp \frac{-\sqrt{3}\,|x|}{2\sigma}. \tag{A.18}$$

A.6 Stochastic Process

We are often interested in experiments whose outcomes are a function of time. For example, we might be interested in designing a system that encodes speech. The outcomes are particular patterns of speech that will be encountered by the speech coder. We can mathematically describe this situation by extending our definition of a random variable. Instead of the random variable mapping an outcome of an experiment to a number, we map it to a function of time. Let S be a sample space with outcomes $\{\omega_i\}$. Then the random or stochastic process X is a mapping

$$X : S \rightarrow \mathcal{F} \tag{A.19}$$

where $\mathcal{F}$ denotes the set of functions on the real number line. In other words,

$$X(\omega) = x(t); \qquad \omega \in S, \; x \in \mathcal{F}, \; -\infty < t < \infty. \tag{A.20}$$

The functions $x(t)$ are called the *realizations* of the random process, and the collection of functions $\{x_\omega(t)\}$ indexed by the outcomes ω is called the *ensemble* of the stochastic process. We can define the mean and variance of the ensemble as

$$\mu(t) = E[X(t)] \tag{A.21}$$

$$\sigma^2(t) = E[(X(t) - \mu(t))^2]. \tag{A.22}$$

If we sample the ensemble at some time t_0, we get a set of numbers $\{x_\omega(t_0)\}$ indexed by the outcomes ω, which by definition is a random variable. By sampling the ensemble at different times t_i, we get different random variables $\{x_\omega(t_i)\}$. For simplicity we often drop the ω and t and simply refer to these random variables as $\{x_i\}$.

Associated with each of these random variables, we will have a distribution function. We can also define a joint distribution function for two or more of these random variables: Given a set of random variables $\{x_1, x_2, \ldots, x_N\}$, the *joint* cumulative distribution function is defined as

$$F_{X_1 X_2 \cdots X_N}(x_1, x_2, \ldots, x_N) = P(X_1 < x_1, X_2 < x_2, \ldots, X_N < x_N) \tag{A.23}$$

Unless it is clear from the context what we are talking about, we will refer to the *cdf* of the individual random variables X_i as the *marginal cdf* of X_i.

We can also define the joint probability density function for these random variables $f_{X_1 X_2 \cdots X_N}(x_1, x_2, \ldots, x_N)$ in the same manner as we defined the *pdf* in the case of the single random variable. We can classify the relationships between these random variables in a number of different ways. In the following we define some relationships between two random variables. The concepts are easily extended to more than two random variables.

Two random variables X_1 and X_2 are said to be *independent* if their joint distribution function can be written as the product of the marginal distribution functions of each random variable; that is,

$$F_{X_1 X_2}(x_1, x_2) = F_{X_1}(x_1) F_{X_2}(x_2). \tag{A.24}$$

This also implies that

$$f_{X_1 X_2}(x_1, x_2) = f_{X_1}(x_1) f_{X_2}(x_2). \tag{A.25}$$

If all the random variables $X_1, X_2, \ldots$ are independent and they have the same distribution, they are said to be *independent, identically distributed* (*iid*).

Two random variables X_1 and X_2 are said to be *orthogonal* if

$$E[X_1 X_2] = 0. \tag{A.26}$$

Two random variables X_1 and X_2 are said to be *uncorrelated* if

$$E[(X_1 - \mu_1)(X_2 - \mu_2)] = 0 \tag{A.27}$$

where $\mu_1 = E[X_1]$ and $\mu_2 = E[X_2]$.

The *autocorrelation function* of a random process is defined as

$$R_{xx}(t_i, t_2) = E[X_1 X_2]. \tag{A.28}$$

For a given value of N, suppose we sample the stochastic process at N times $\{t_i\}$ to get the N random variables $\{X_i\}$ with *cdf* $F_{X_1 X_2 \ldots X_N}(x_1, x_2, \ldots, x_N)$, and another N times $\{t_i + T\}$ to get the random variables $\{X_i'\}$ with *cdf* $F_{X_1' X_2' \ldots X_N'}(x_1', x_2', \ldots, x_N')$. If

$$F_{X_1 X_2 \ldots X_N}(x_1, x_2, \ldots, x_N) = F_{X_1' X_2' \ldots X_N'}(x_1', x_2', \ldots, x_N') \tag{A.29}$$

for all N and T, the process is said to be *stationary*.

The assumption of stationarity is a rather important assumption because it is a statement that the statistical characteristics of the process under investigation do not change with time. Thus, if we design a system for an input based on the statistical characteristics of the input today, the system will still be useful tomorrow because the input will not change its characteristics. The assumption of stationarity is also a very strong assumption, and we can usually make do quite well with a weaker condition, *wide sense* or *weak sense* stationarity.

A stochastic process is said to be wide sense or weak sense stationary if it satisfies the following conditions:

1. The mean is constant; that is, $\mu(t) = \mu$ for all t.
2. The variance is finite.
3. The autocorrelation function $R_{xx}(t_1, t_2)$ is a function only of the difference between t_1 and t_2, and not of the individual values of t_1 and t_2; that is,

$$R_{xx}(t_1, t_2) = R_{xx}(t_1 - t_2) = R_{xx}(t_2 - t_1). \tag{A.30}$$

Further Reading

1. The classic books on probability are the two-volume set *An Introduction to Probability Theory and Its Applications*, by W. Feller [171].
2. A commonly used text for an introductory course on probability and random processes is *Probability, Random Variables, and Stochastic Processes*, by A. Papoulis [172].

A.7 Projects and Problems

1. If $A \cap B \neq \phi$, show that

$$P(A \cup B) = P(A) + P(B) - P(A \cap B).$$

2. Show that expectation is a linear operator in both the discrete and the continuous case.
3. If a is a constant, show that $E[a] = a$.
4. Show that for a random variable X,

$$\sigma_X^2 = E[X^2] - \mu_X^2.$$

5. Show that the variance of the uniform distribution is given by

$$\sigma_X^2 = \frac{(b-a)^2}{12}.$$

A Brief Review of Matrix Concepts

In this appendix we will look at some of the basic concepts of matrix algebra. Our intent is simply to familiarize you with some basic matrix operations that we will need in our study of compression. Matrices are very useful for representing linear systems of equations, and matrix theory is a powerful tool for the study of linear operators. In our study of compression techniques we will use matrices both in the solution of systems of equations and in our study of linear transforms.

B.1 A Matrix

A collection of real or complex elements arranged in M rows and N columns is called a matrix of order $M \times N$

$$\mathbf{A} = \begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0N-1} \\ a_{10} & a_{11} & \cdots & a_{1N-1} \\ \vdots & \vdots & & \vdots \\ a_{(M-1)0} & a_{(M-1)1} & \cdots & a_{M-1N-1} \end{bmatrix} \tag{B.1}$$

where the first subscript denotes the row that an element belongs to and the second subscript denotes the column. For example, the element a_{02} belongs in row 0 and column 2, and the element a_{32} belongs in row 3 and column 2. The generic ijth element of a matrix $\mathbf{A}$ is sometimes represented as $[\mathbf{A}]_{ij}$. If the number of rows is equal to the number of columns $(N = M)$, then the matrix is called a *square matrix.* A special square matrix that we will be using is the *identity matrix* $\mathbf{I}$, in which the elements on the diagonal of the matrix are 1 and all other elements are 0:

$$[\mathbf{I}]_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j. \end{cases} \tag{B.2}$$

If a matrix consists of a single column ($N = 1$), it is called a *column matrix* or *vector* of dimension M. If it consists of a single row ($M = 1$), it is called a *row matrix* or *vector* of dimension N.

The *transpose* $\mathbf{A}^T$ of a matrix $\mathbf{A}$ is the $N \times M$ matrix obtained by writing the rows of the matrix as columns and the columns as rows:

$$\mathbf{A}^T = \begin{bmatrix} a_{00} & a_{10} & \cdots & a_{(M-1)0} \\ a_{01} & a_{11} & \cdots & a_{(M-1)1} \\ \vdots & \vdots & & \vdots \\ a_{0(N-1)} & a_{1(N-1)} & \cdots & a_{M-1N-1} \end{bmatrix} \tag{B.3}$$

The transpose of a column matrix is a row matrix and vice versa.

Two matrices $\mathbf{A}$ and $\mathbf{B}$ are said to be equal if they are of the same order and their corresponding elements are equal; that is,

$$\mathbf{A} = \mathbf{B} \quad \Leftrightarrow \quad a_{ij} = b_{ij}, i = 0, 1, \ldots M - 1; j = 0, 1, \ldots N - 1. \tag{B.4}$$

B.2 Matrix Operations

You can add, subtract, and multiply matrices, but since matrices come in all shapes and sizes, there are some restrictions as to what operations you can perform with what kind of matrices. In order to add or subtract two matrices, their dimensions have to be identical—same number of rows and same number of columns. In order to multiply two matrices, the order in which they are multiplied is important. In general $\mathbf{A} \times \mathbf{B}$ is not equal to $\mathbf{B} \times \mathbf{A}$. Multiplication is only defined for the case where the number of columns of the first matrix is equal to the number of rows of the second matrix. The reasons for these restrictions will become apparent when we look at how the operations are defined.

When we add two matrices, the resultant matrix consists of elements that are the sum of the corresponding entries in the matrices being added. Let us add two matrices $\mathbf{A}$ and $\mathbf{B}$ where

$$\mathbf{A} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix}$$

and

$$\mathbf{B} = \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \end{bmatrix}$$

The sum of the two matrices, $\mathbf{C}$, is given by

$$\mathbf{C} = \begin{bmatrix} c_{00} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix} = \begin{bmatrix} a_{00} + b_{00} & a_{01} + b_{01} & a_{02} + b_{02} \\ a_{10} + b_{10} & a_{11} + b_{11} & a_{12} + b_{12} \end{bmatrix} \tag{B.5}$$

Notice that each element of the resulting matrix $\mathbf{C}$ is the sum of corresponding elements of the matrices $\mathbf{A}$ and $\mathbf{B}$. In order for the two matrices to have corresponding elements, the dimension of the two matrices has to be the same. Therefore, addition is only defined for matrices with identical dimensions (i.e., same number of rows and same number of columns).

Subtraction is defined in a similar manner. The elements of the difference matrix are made up of term-by-term subtraction of the matrices being subtracted.

We could have generalized matrix addition and matrix subtraction from our knowledge of addition and subtraction of numbers. Multiplication of matrices is another kettle of fish entirely. It is easiest to describe matrix multiplication with an example. Suppose we have two different matrices **A** and **B** where

$$\mathbf{A} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix}$$

and

$$\mathbf{B} = \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \\ b_{20} & b_{21} \end{bmatrix} \tag{B.6}$$

The product is given by

$$\mathbf{C} = \mathbf{AB} = \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} = \begin{bmatrix} a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20} & a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21} \\ a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20} & a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21} \end{bmatrix}$$

You can see that the i, j element of the product is obtained by adding term by term the product of elements in the ith row of the first matrix with those of the jth column of the second matrix. Thus, the element c_{10} in the matrix **C** is obtained by summing the term-by-term products of row 1 of the first matrix **A** with column 0 of the matrix **B**. We can also see that the resulting matrix will have as many rows as the matrix to the left and as many columns as the matrix to the right.

What happens if we reverse the order of the multiplication? By the rules above we will end up with a matrix with three rows and three columns.

$$\begin{bmatrix} b_{00}a_{00} + b_{01}a_{10} & b_{00}a_{01} + +b_{01}a_{11} & b_{00}a_{02} + b_{01}a_{12} \\ b_{10}a_{00} + b_{11}a_{10} & b_{10}a_{01} + +b_{11}a_{11} & b_{10}a_{02} + b_{11}a_{12} \\ b_{20}a_{00} + b_{21}a_{10} & b_{20}a_{01} + +b_{21}a_{11} & b_{20}a_{02} + b_{21}a_{12} \end{bmatrix}$$

The elements of the two product matrices are different as are the dimensions.

As we can see, multiplication between matrices follows some rather different rules than multiplication between real numbers. The sizes have to match up—the number of columns of the first matrix has to be equal to the number of rows of the second matrix, and the order of multiplication is important. Because of the latter fact we often talk about premultiplying or postmultiplying. Premultiplying **B** by **A** results in the product **AB**, while postmultiplying **B** by **A** results in the product **BA**.

We have three of the four elementary operations. What about the fourth elementary operation, division? The easiest way to present division in matrices is to look at the formal definition of division when we are talking about real numbers. In the real number system, for every number a different from zero, there exists an inverse, denoted by $1/a$ or a^{-1}, such that the product of a with its inverse is one. When we talk about a number b divided by a number a, this is the same as the *multiplication* of b with the inverse of a. Therefore, we could define division by a matrix as the multiplication with the inverse of the matrix. **A**/**B** would be given by $\mathbf{AB}^{-1}$. Once we have the definition of an inverse of a matrix, the rules of multiplication apply.

So how do we define the inverse of a matrix? Following the definition for real numbers, in order to define the inverse of a matrix we need to have the matrix counterpart of 1. In matrices this counterpart is called the *identity matrix*. The identity matrix is a square matrix with diagonal elements being 1 and off-diagonal elements being 0. For example, a 3×3 identity matrix is given by

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{B.7}$$

The identity matrix behaves like the number one in the matrix world. If we multiply any matrix with the identity matrix (of appropriate dimension), we get the original matrix back. Given a square matrix $\mathbf{A}$, we define its inverse, $\mathbf{A}^{-1}$, as the matrix that when premultiplied or postmultiplied by $\mathbf{A}$ results in the identity matrix. For example, consider the matrix

$$\mathbf{A} = \begin{bmatrix} 3 & 4 \\ 1 & 2 \end{bmatrix} \tag{B.8}$$

The inverse matrix is given by

$$\mathbf{A}^{-1} = \begin{bmatrix} 1 & -2 \\ -0.5 & 1.5 \end{bmatrix} \tag{B.9}$$

To check that this is indeed the inverse matrix, let us multiply them:

$$\begin{bmatrix} 3 & 4 \\ 1 & 2 \end{bmatrix}\begin{bmatrix} 1 & -2 \\ -0.5 & 1.5 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{B.10}$$

and

$$\begin{bmatrix} 1 & -2 \\ -0.5 & 1.5 \end{bmatrix}\begin{bmatrix} 3 & 4 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{B.11}$$

If $\mathbf{A}$ is a vector of dimension M, we can define two specific kinds of products. If $\mathbf{A}$ is a column matrix, then the *inner product* or *dot product* is defined as

$$\mathbf{A}^T\mathbf{A} = \sum_{i=0}^{M-1} a_{i0}^2 \tag{B.12}$$

and the *outer product* or *cross product* is defined as

$$\mathbf{AA}^T = \begin{bmatrix} a_{00}a_{00} & a_{00}a_{10} & \cdots & a_{00}a_{(M-1)0} \\ a_{10}a_{00} & a_{10}a_{10} & \cdots & a_{10}a_{(M-1)0} \\ \vdots & \vdots & & \vdots \\ a_{(M-1)0}a_{00} & a_{\{(M-1)1\}}a_{10} & \cdots & a_{(M-1)0}a_{(M-1)0} \end{bmatrix} \tag{B.13}$$

Notice that the inner product results in a scalar, while the outer product results in a matrix.

In order to find the inverse of a matrix, we need the concepts of determinant and cofactor. Associated with each square matrix is a scalar value called the *determinant* of the matrix. The determinant of a matrix **A** is denoted as $|\mathbf{A}|$. To see how to obtain the determinant of an $N \times N$ matrix, we start with a 2×2 matrix. The determinant of a 2×2 matrix is given as

$$|\mathbf{A}| = \begin{vmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{vmatrix} = a_{00}a_{11} - a_{01}a_{10}. \tag{B.14}$$

Finding the determinant of a 2×2 matrix is easy. To explain how to get the determinants of larger matrices, we need to define some terms.

The *minor* of an element a_{ij} of an $N \times N$ matrix is defined to be the determinant of the $N-1 \times N-1$ matrix obtained by deleting the row and column containing a_{ij}. For example, if **A** is a 4×4 matrix

$$\mathbf{A} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \tag{B.15}$$

then the minor of the element a_{12}, denoted by M_{12}, is the determinant

$$M_{12} = \begin{vmatrix} a_{00} & a_{01} & a_{03} \\ a_{20} & a_{21} & a_{23} \\ a_{30} & a_{31} & a_{33} \end{vmatrix} \tag{B.16}$$

The cofactor of a_{ij} denoted by $\mathbf{A}_{ij}$ is given by

$$\mathbf{A}_{ij} = (-1)^{i+j} M_{ij}. \tag{B.17}$$

Armed with these definitions we can write an expression for the determinant of an $N \times N$ matrix as

$$|\mathbf{A}| = \sum_{i=0}^{N-1} a_{ij}\mathbf{A}_{ij} \tag{B.18}$$

or

$$|\mathbf{A}| = \sum_{j=0}^{N-1} a_{ij}\mathbf{A}_{ij} \tag{B.19}$$

where the a_{ij} are taken from a single row or a single column. If the matrix has a particular row or column that has a large number of zeros in it, we would need fewer computations if we picked that particular row or column.

Equations (B.18) and (B.19) express the determinant of an $N \times N$ matrix in terms of determinants of $N-1 \times N-1$ matrices. We can express each of the $N-1 \times N-1$ determinants in terms of $N-2 \times N-2$ determinants, continuing in this fashion until we have everything expressed in terms of 2×2 determinants, which can be evaluated using (B.14).

Now that we know how to compute a determinant, we need one more definition before we can define the inverse of a matrix. The *adjoint* of a matrix **A**, denoted by (**A**), is a matrix whose ijth element is the cofactor $\mathbf{A}_{ji}$. The inverse of a matrix $\mathbf{A}$, denoted by $\mathbf{A}^{-1}$, is given by

$$\mathbf{A}^{-1} = \frac{1}{|\mathbf{A}|}(\mathbf{A}). \tag{B.20}$$

Notice that for the inverse to exist the determinant has to be nonzero. If the determinant for a matrix is zero, the matrix is said to be singular. The method we have described here works well with small matrices; however, it is highly inefficient if N becomes greater than 4. There are a number of efficient methods for inverting matrices; see the books in the Further Reading section for details.

Corresponding to a square matrix $\mathbf{A}$ of size $N \times N$ are N scalar values called the *eigenvalues* of $\mathbf{A}$. The eigenvalues are the N solutions of the equation $|\lambda\mathbf{I} - \mathbf{A}| = 0$. This equation is called the *characteristic equation.*

Example B.2.1:

Let us find the eigenvalues of the matrix

$$\begin{bmatrix} 4 & 5 \\ 2 & 1 \end{bmatrix}$$

$$\begin{aligned} |\lambda\mathbf{I} - \mathbf{A}| &= 0 \\ \left| \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} - \begin{bmatrix} 4 & 5 \\ 2 & 1 \end{bmatrix} \right| &= 0 \\ (\lambda - 4)(\lambda - 1) - 10 &= 0 \\ \lambda_1 = -1 \quad \lambda_2 &= 6 \end{aligned} \tag{B.21}$$

♦

The eigenvectors V_k of an $N \times N$ matrix are the N vectors of dimension N that satisfy the equation

$$\mathbf{A}V_k = \lambda_k V_k. \tag{B.22}$$

Further Reading

1. The subject of matrices is covered at an introductory level in a number of textbooks. A good one is *Advanced Engineering Mathematics*, by E. Kreyszig [129].
2. Numerical methods for manipulating matrices (and a good deal more) are presented in *Numerical Recipes in C*, by W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery [178].

The Root Lattices

Define $\mathbf{e}_i^L$ to be a vector in L dimensions whose ith component is 1 and all other components are 0. Some of the root systems that are used in lattice vector quantization are given as follows:

$$\begin{array}{lll}
D_L & \pm\mathbf{e}_i^L \pm \mathbf{e}_j^L, & i \neq j,\ i, j = 1, 2, \ldots, L \\
A_L & \pm(\mathbf{e}_i^{L+1} - \mathbf{e}_j^{L+1}), & i \neq j,\ i, j = 1, 2, \ldots, L \\
E_L & \pm\mathbf{e}_i^L \pm \mathbf{e}_j^L, & i \neq j,\ i, j = 1, 2, \ldots, L-1, \\
 & \frac{1}{2}(\pm\mathbf{e}_1 \pm \mathbf{e}_2 \cdots \pm \mathbf{e}_{L-1} \pm \sqrt{2 - \frac{(L-1)}{4}}\mathbf{e}_L) & L = 6, 7, 8
\end{array}$$

Let us look at each of these definitions a bit closer and see how they can be used to generate lattices.

D_L Let us start with the D_L lattice. For $L =.2$, the four roots of the D_2 algebra are $\mathbf{e}_1^2 + \mathbf{e}_2^2$, $\mathbf{e}_1^2 - \mathbf{e}_2^2$, $-\mathbf{e}_1^2 + \mathbf{e}_2^2$, and $-\mathbf{e}_1^2 - \mathbf{e}_2^2$, or $(1, 1)$, $(1, -1)$, $(-1, 1)$, and $(-1, -1)$. We can pick any two independent vectors from among these four to form the basis set for the D_2 lattice. Suppose we picked $(1, 1)$ and $(1, -1)$. Then any integral combination of these vectors is a lattice point. The resulting lattice is shown in Figure 10.24 in Chapter 10. Notice that the sum of the coordinates are all even numbers. This makes finding the closest lattice point to an input a relatively simple exercise.

A_L The roots of the A_L lattices are described using $L+1$-dimensional vectors. However, if we select any L independent vectors from this set, we will find that the points that are generated all lie in an L-dimensional slice of the $L+1$-dimensional space. This can be seen from Figure C.1.

We can obtain an L-dimensional basis set from this using a simple algorithm described in [139]. In two dimensions, this results in the generation of the vectors $(1, 0)$ and $(-\frac{1}{2}, \frac{\sqrt{3}}{2})$. The resulting lattice is shown in Figure 10.25 in Chapter 10. To find the closest point to the A_L lattice, we use the fact that in the embedding of the lattice in $L+1$ dimensions, the sum of the coordinates is always zero. The exact procedure can be found in [141, 140].

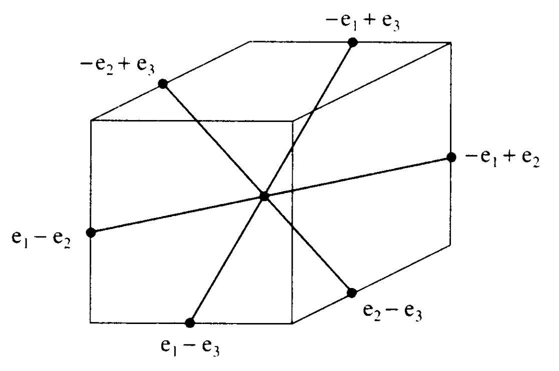

FIGURE C. 1 The A_2 roots embedded in three dimensions.

E_L As we can see from the definition, the E_L lattices go up to a maximum dimension of 8. Each of these lattices can be written as unions of the A_L and D_L lattices and their translated version. For example, the E_8 lattice is the union of the D_8 lattice and the D_8 lattice translated by the vector $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$. Therefore, to find the closest E_8 point to an input $\mathbf{x}$, we find the closest point of D_8 to $\mathbf{x}$, and the closest point of D_8 to $\mathbf{x} - (\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$, and pick the one that is closest to $\mathbf{x}$.

There are several advantages to using lattices as vector quantizers. There is no need to store the codebook, and finding the closest lattice point to a given input is a rather simple operation. However, the quantizer codebook is only a subset of the lattice. How do we know when we have wandered out of this subset, and what do we do about it? Furthermore, how do we generate a binary codeword for each of the lattice points that lie within the boundary? The first problem is easy to solve. Earlier we discussed the selection of a boundary to reduce the effect of the overload error. We can check the location of the lattice point to see if it is within this boundary. If not, we are outside the subset. The other questions are more difficult to resolve. Conway and Sloane [142] have developed a technique that functions by first defining the boundary as one of the quantization regions (expanded many times) of the root lattices. The technique is not very complicated, but it takes some time to set up, so we will not describe it here (see [142] for details).

We have given a sketchy description of lattice quantizers. For a more detailed tutorial review, see [140]. A more theoretical review and overview can be found in [262].