

where in the last equation we have used the substitution $l = 2m + k$. Suppose we have a function $f(t)$ that can be accurately represented at resolution $j + 1$ by some scaling function $\phi(t)$. We assume that the scaling function and its dilations and translations form an orthonormal set. The coefficients c_{j+1} can be obtained by

$$c_{j+1,k} = \int f(t)\phi_{j+1,k}dt. \quad (15.48)$$

If we can represent $f(t)$ accurately at resolution $j + 1$ with a linear combination of $\phi_{j+1,k}(t)$, then from the previous section we can decompose it into two functions: one in terms of $\phi_{j,k}(t)$ and one in terms of the j th dilation of the corresponding wavelet $\{\psi_{j,k}(t)\}$. The coefficients $c_{j,k}$ are given by

$$c_{j,k} = \int f(t)\phi_{j,k}(t)dt \quad (15.49)$$

$$= \int f(t)2^{\frac{j}{2}}\phi(2^j t - k)dt. \quad (15.50)$$

Substituting for $\phi(2^j t - k)$ from (15.47), we get

$$c_{j,l} = \int f(t)2^{\frac{j}{2}} \sum_l h_{l-2k} \sqrt{2}\phi(2^{j+1}t - l)dt. \quad (15.51)$$

Interchanging the order of summation and integration, we get

$$c_{j,l} = \sum_l h_{l-2k} \int f(t)2^{\frac{j}{2}} \sqrt{2}\phi(2^{j+1}t - l)dt. \quad (15.52)$$

But the integral is simply $c_{j+1,k}$. Therefore,

$$c_{j,k} = \sum_k h_{k-2m} c_{j+1,k}. \quad (15.53)$$

We have encountered this relationship before in the context of the Haar function. Equation (15.27) provides the relationship between coefficients of the Haar expansion at two resolution levels. In a more general setting, the coefficients $\{h_j\}$ provide a link between the coefficients $\{c_{j,k}\}$ at different resolutions. Thus, given the coefficients at resolution level $j + 1$, we can obtain the coefficients at all other resolution levels. But how do we start the process? Recall that $f(t)$ can be accurately represented at resolution $j + 1$. Therefore, we can replace $c_{j+1,k}$ by the samples of $f(t)$. Let's represent these samples by x_k . Then the coefficients of the low-resolution expansion are given by

$$c_{j,k} = \sum_k h_{k-2m} x_k. \quad (15.54)$$

In Chapter 12, we introduced the input-output relationship of a linear filter as

$$y_m = \sum_k h_k x_{m-k} = \sum_k h_{m-k} x_k. \quad (15.55)$$

Replacing m by $2m$, we get every other sample of the output

$$y_{2m} = \sum_k h_{2m-k} x_k. \quad (15.56)$$

Comparing (15.56) with (15.54), we can see that the coefficients of the low-resolution approximation are every other output of a linear filter whose impulse response is h_{-k} . Recall that $\{h_k\}$ are the coefficients that satisfy the MRA equation. Using the terminology of subband coding, the coefficients $c_{j,k}$ are the downsampled output of the linear filter with impulse response $\{h_{-k}\}$.

The detail portion of the representation is obtained in a similar manner. Again we start from the recursion relationship. This time we use the recursion relationship for the wavelet function as our starting point:

$$\psi(t) = \sum_k w_k \sqrt{2} \phi(2t - k). \quad (15.57)$$

Again substituting $t = 2^j t - m$ and using the same simplifications, we get

$$\psi(2^j t - m) = \sum_k w_{k-2m} \sqrt{2} \phi(2^{j+1} t - k). \quad (15.58)$$

Using the fact that the dilated and translated wavelets form an orthonormal basis, we can obtain the detail coefficients $d_{j,k}$ by

$$d_{j,k} = \int f(t) \psi_{j,k}(t) dt \quad (15.59)$$

$$= \int f(t) 2^{\frac{j}{2}} \psi(2^j t - k) dt \quad (15.60)$$

$$= \int f(t) 2^{\frac{j}{2}} \sum_l w_{l-2k} \sqrt{2} \phi(2^{j+1} t - l) dt \quad (15.61)$$

$$= \sum_l w_{l-2k} \int f(t) 2^{\frac{j+1}{2}} \phi(2^{j+1} t - l) dt \quad (15.62)$$

$$= \sum_l w_{l-2k} c_{j+1,l}. \quad (15.63)$$

Thus, the detail coefficients are the decimated outputs of a filter with impulse response $\{w_{-k}\}$.

At this point we can use exactly the same arguments to further decompose the coefficients $\{c_j\}$.

In order to retrieve $\{c_{j+1,k}\}$ from $\{c_{j,k}\}$ and $\{d_{j,k}\}$, we upsample the lower resolution coefficients and filter, using filters with impulse response $\{h_k\}$ and $\{w_k\}$

$$c_{j+1,k} = \sum_l c_{j,l} b_{k-2l} \sum_l d_{j,l} w_{k-2l}.$$

15.5.1 Scaling and Wavelet Coefficients

In order to implement the wavelet decomposition, the coefficients $\{h_k\}$ and $\{w_k\}$ are of primary importance. In this section we look at some of the properties of these coefficients that will help us in finding different decompositions.

We start with the MRA equation. Integrating both sides of the equation over all t , we obtain

$$\int_{-\infty}^{\infty} \phi(t) dt = \int_{-\infty}^{\infty} \sum_k h_k \sqrt{2} \phi(2t - k) dt. \quad (15.64)$$

Interchanging the summation and integration on the right-hand side of the equation, we get

$$\int_{-\infty}^{\infty} \phi(t) dt = \sum_k h_k \sqrt{2} \int_{-\infty}^{\infty} \phi(2t - k) dt. \quad (15.65)$$

Substituting $x = 2t - k$ with $dx = 2dt$ in the right-hand side of the equation, we get

$$\int_{-\infty}^{\infty} \phi(t) dt = \sum_k h_k \sqrt{2} \int_{-\infty}^{\infty} \phi(x) \frac{1}{2} dx \quad (15.66)$$

$$= \sum_k h_k \frac{1}{\sqrt{2}} \int_{-\infty}^{\infty} \phi(x) dx. \quad (15.67)$$

Assuming that the average value of the scaling function is not zero, we can divide both sides by the integral and we get

$$\sum_k h_k = \sqrt{2}. \quad (15.68)$$

If we normalize the scaling function to have a magnitude of one, we can use the orthogonality condition on the scaling function to get another condition on $\{h_k\}$:

$$\int |\phi(t)|^2 dt = \int \sum_k h_k \sqrt{2} \phi(2t - k) \sum_m h_m \sqrt{2} \phi(2t - m) dt \quad (15.69)$$

$$= \sum_k \sum_m h_k h_m 2 \int \phi(2t - k) \phi(2t - m) dt \quad (15.70)$$

$$= \sum_k \sum_m h_k h_m \int \phi(x - k) \phi(x - m) dx \quad (15.71)$$

where in the last equation we have used the substitution $x = 2t$. The integral on the right-hand side is zero except when $k = m$. When $k = m$, the integral is unity and we obtain

$$\sum_k h_k^2 = 1. \quad (15.72)$$

We can actually get a more general property by using the orthogonality of the translates of the scaling function

$$\int \phi(t) \phi(t - m) dt = \delta_m. \quad (15.73)$$

Rewriting this using the MRA equation to substitute for $\phi(t)$ and $\phi(t - m)$, we obtain

$$\begin{aligned} & \int \left[\sum_k h_k \sqrt{2} \phi(2t - k) \right] \left[\sum_l h_l \sqrt{2} \phi(2t - 2m - l) \right] dt \\ &= \sum_k \sum_l h_k h_l 2 \int \phi(2t - k) \phi(2t - 2m - l) dt. \end{aligned} \quad (15.74)$$

Substituting $x = 2t$, we get

$$\int \phi(t)\phi(t-m)dt = \sum_k \sum_l h_k h_l \int \phi(x-k)\phi(x-2m-l)dx \quad (15.75)$$

$$= \sum_k \sum_l h_k h_l \delta_{k-(2m+l)} \quad (15.76)$$

$$= \sum_k h_k h_{k-2m}. \quad (15.77)$$

Therefore, we have

$$\sum_k h_k h_{k-2m} = \delta_m \quad (15.78)$$

Notice that this is the same relationship we had to satisfy for perfect reconstruction in the previous chapter.

Using these relationships, we can generate scaling coefficients for filters of various lengths.

Example 15.5.1:

For $k = 2$, we have from (15.68) and (15.72)

$$h_0 + h_1 = \sqrt{2} \quad (15.79)$$

$$h_0^2 + h_1^2 = 1. \quad (15.80)$$

These equations are uniquely satisfied by

$$h_0 = h_1 = \frac{1}{\sqrt{2}},$$

which is the Haar scaling function. \blacklozenge

An orthogonal expansion does not exist for all lengths. In the following example, we consider the case of $k = 3$.

Example 15.5.2:

For $k = 3$, from the three conditions (15.68), (15.72), and (15.78), we have

$$h_0 + h_1 + h_2 = \sqrt{2} \quad (15.81)$$

$$h_0^2 + h_1^2 + h_2^2 = 1 \quad (15.82)$$

$$h_0 h_2 = 0 \quad (15.83)$$

The last condition can only be satisfied if $h_0 = 0$ or $h_2 = 0$. In either case we will be left with the two-coefficient filter for the Haar scaling function. \blacklozenge

In fact, we can see that for k odd, we will always end up with a condition that will force one of the coefficients to zero, thus leaving an even number of coefficients. When the

number of coefficients gets larger than the number of conditions, we end up with an infinite number of solutions.

Example 15.5.3:

Consider the case when $k = 4$. The three conditions give us the following three equations:

$$h_0 + h_1 + h_2 + h_3 = \sqrt{2} \quad (15.84)$$

$$h_0^2 + h_1^2 + h_2^2 + h_3^2 = 1 \quad (15.85)$$

$$h_0 h_2 + h_1 h_3 = 0 \quad (15.86)$$

We have three equations and four unknowns; that is, we have one degree of freedom. We can use this degree of freedom to impose further conditions on the solution. The solutions to these equations include the Daubechies four-tap solution:

$$h_0 = \frac{1 + \sqrt{3}}{4\sqrt{2}}, \quad h_1 = \frac{3 + \sqrt{3}}{4\sqrt{2}}, \quad h_2 = \frac{3 - \sqrt{3}}{4\sqrt{2}}, \quad h_3 = \frac{1 - \sqrt{3}}{4\sqrt{2}}. \quad \blacklozenge$$

Given the close relationship between the scaling function and the wavelet, it seems reasonable that we should be able to obtain the coefficients for the wavelet filter from the coefficients of the scaling filter. In fact, if the wavelet function is orthogonal to the scaling function at the same scale

$$\int \phi(t - k)\psi(t - m)dt = 0, \quad (15.87)$$

then

$$w_k = \pm(-1)^k h_{N-k} \quad (15.88)$$

and

$$\sum_k h_k w_{n-2k} = 0. \quad (15.89)$$

Furthermore,

$$\sum_k w_k = 0. \quad (15.90)$$

The proof of these relationships is somewhat involved [207].

15.5.2 Families of Wavelets

Let's move to the more practical aspects of compression using wavelets. We have said that there is an infinite number of possible wavelets. Which one is best depends on the application. In this section we list different wavelets and their corresponding filters. You are encouraged to experiment with these to find those best suited to your application.

The 4-tap, 12-tap, and 20-tap Daubechies filters are shown in Tables 15.1–15.3. The 6-tap, 12-tap, and 18-tap Coiflet filters are shown in Tables 15.4–15.6.

TABLE 15.1 Coefficients for the 4-tap Daubechies low-pass filter.

| | |
|-------|---------------------|
| h_0 | 0.4829629131445341 |
| h_1 | 0.8365163037378079 |
| h_2 | 0.2241438680420134 |
| h_3 | -0.1294095225512604 |

TABLE 15.2 Coefficients for the 12-tap Daubechies low-pass filter.

| | |
|----------|-----------------|
| h_0 | 0.111540743350 |
| h_1 | 0.494623890398 |
| h_2 | 0.751133908021 |
| h_3 | 0.315250351709 |
| h_4 | -0.226264693965 |
| h_5 | -0.129766867567 |
| h_6 | 0.097501605587 |
| h_7 | 0.027522865530 |
| h_8 | -0.031582039318 |
| h_9 | 0.000553842201 |
| h_{10} | 0.004777257511 |
| h_{11} | -0.001077301085 |

TABLE 15.3 Coefficients for the 20-tap Daubechies low-pass filter.

| | |
|----------|-----------------|
| h_0 | 0.026670057901 |
| h_1 | 0.188176800078 |
| h_2 | 0.527201188932 |
| h_3 | 0.688459039454 |
| h_4 | 0.281172343661 |
| h_5 | -0.249846424327 |
| h_6 | -0.195946274377 |
| h_7 | 0.127369340336 |
| h_8 | 0.093057364604 |
| h_9 | -0.071394147166 |
| h_{10} | -0.029457536822 |
| h_{11} | 0.033212674059 |
| h_{12} | 0.003606553567 |
| h_{13} | -0.010733175483 |
| h_{14} | 0.001395351747 |
| h_{15} | 0.001992405295 |
| h_{16} | -0.000685856695 |
| h_{17} | -0.000116466855 |
| h_{18} | 0.000093588670 |
| h_{19} | -0.000013264203 |

TABLE 15.4 Coefficients for the 6-tap Coiflet low-pass filter.

| | |
|-------|-----------------|
| h_0 | -0.051429728471 |
| h_1 | 0.238929728471 |
| h_2 | 0.602859456942 |
| h_3 | 0.272140543058 |
| h_4 | -0.051429972847 |
| h_5 | -0.011070271529 |

TABLE 15.5 Coefficients for the 12-tap Coiflet low-pass filter.

| | |
|----------|-----------------|
| h_0 | 0.011587596739 |
| h_1 | -0.029320137980 |
| h_2 | -0.047639590310 |
| h_3 | 0.273021046535 |
| h_4 | 0.574682393857 |
| h_5 | 0.294867193696 |
| h_6 | -0.054085607092 |
| h_7 | -0.042026480461 |
| h_8 | 0.016744410163 |
| h_9 | 0.003967883613 |
| h_{10} | -0.001289203356 |
| h_{11} | -0.000509505539 |

TABLE 15.6 Coefficients for the 18-tap Coiflet low-pass filter.

| | |
|----------|-----------------|
| h_0 | -0.002682418671 |
| h_1 | 0.005503126709 |
| h_2 | 0.016583560479 |
| h_3 | -0.046507764479 |
| h_4 | -0.043220763560 |
| h_5 | 0.286503335274 |
| h_6 | 0.561285256870 |
| h_7 | 0.302983571773 |
| h_8 | -0.050770140755 |
| h_9 | -0.058196250762 |
| h_{10} | 0.024434094321 |
| h_{11} | 0.011229240962 |
| h_{12} | -0.006369601011 |
| h_{13} | -0.001820458916 |
| h_{14} | 0.000790205101 |
| h_{15} | 0.000329665174 |
| h_{16} | -0.000050192775 |
| h_{17} | -0.000024465734 |

15.6 Image Compression

One of the most popular applications of wavelets has been to image compression. The JPEG 2000 standard, which is designed to update and replace the current JPEG standard, will use wavelets instead of the DCT to perform decomposition of the image. During our discussion we have always referred to the signal to be decomposed as a one-dimensional signal; however, images are two-dimensional signals. There are two approaches to the subband decomposition of two-dimensional signals: using two-dimensional filters, or using separable transforms that can be implemented using one-dimensional filters on the rows first and then on the columns (or vice versa). Most approaches, including the JPEG 2000 verification model, use the second approach.

In Figure 15.10 we show how an image can be decomposed using subband decomposition. We begin with an $N \times M$ image. We filter each row and then downsample to obtain two $N \times \frac{M}{2}$ images. We then filter each column and subsample the filter output to obtain four $\frac{N}{2} \times \frac{M}{2}$ images. Of the four subimages, the one obtained by low-pass filtering the rows and columns is referred to as the LL image; the one obtained by low-pass filtering the rows and high-pass filtering the columns is referred to as the LH image; the one obtained by high-pass filtering the rows and low-pass filtering the columns is called the HL image; and the subimage obtained by high-pass filtering the rows and columns is referred to as the HH image. This decomposition is sometimes represented as shown in Figure 15.11. Each of the subimages obtained in this fashion can then be filtered and subsampled to obtain four more subimages. This process can be continued until the desired subband structure is obtained. Three popular structures are shown in Figure 15.12. In the structure in Figure 15.12a, the LL subimage has been decomposed after each decomposition into four more subimages, resulting in a total of 10 subimages. This is one of the more popular decompositions.

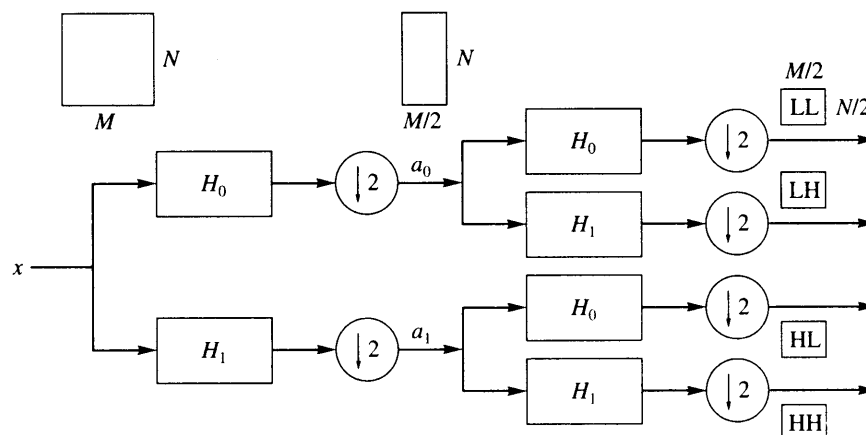


FIGURE 15.10 Subband decomposition of an $N \times M$ image.

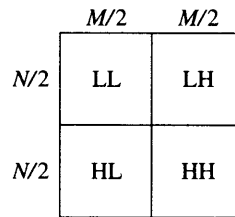


FIGURE 15.11 First-level decomposition.

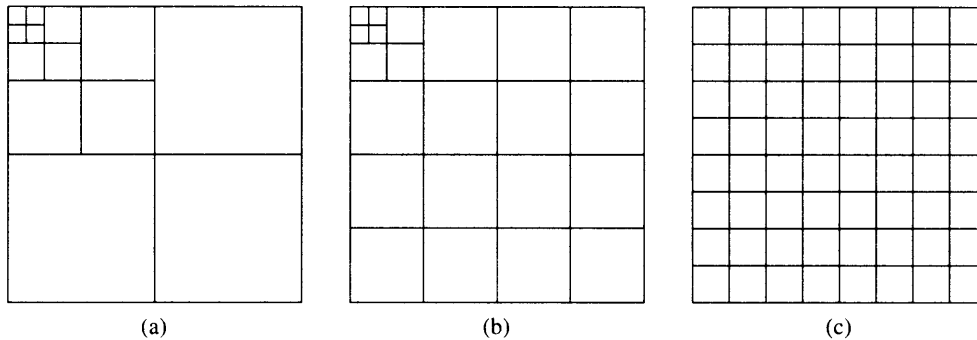


FIGURE 15.12 Three popular subband structures.

Example 15.6.1:

Let's use the Daubechies wavelet filter to repeat what we did in Examples 14.12.2 and 14.12.3 using the Johnston and the Smith-Barnwell filters. If we use the 4-tap Daubechies filter, we obtain the decomposition shown in Figure 15.13. Notice that even though we are only using a 4-tap filter, we get results comparable to the 16-tap Johnston filter and the 8-tap Smith-Barnwell filters.

If we now encode this image at the rate of 0.5 bits per pixel, we get the reconstructed image shown in Figure 15.14. Notice that the quality is comparable to that obtained using filters requiring two or four times as much computation. ♦

In this example we used a simple scalar quantizer for quantization of the coefficients. However, if we use strategies that are motivated by the properties of the coefficients themselves, we can obtain significant performance improvements. In the next sections we examine two popular quantization strategies developed specifically for wavelets.

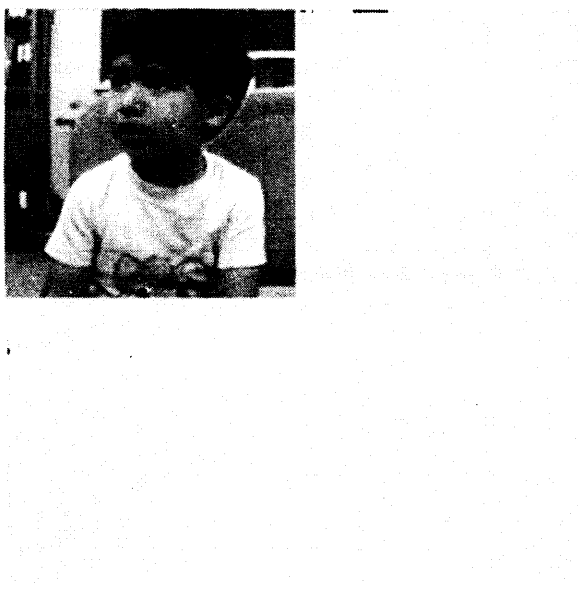


FIGURE 15. 13 Decomposition of Sinan image using the four-tap Daubechies filter.

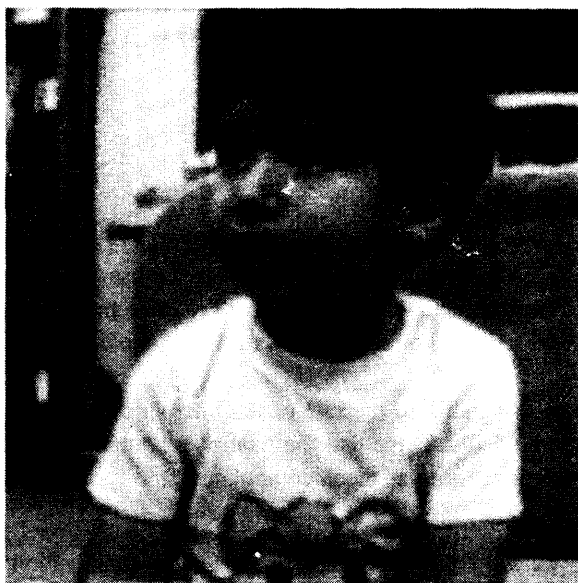


FIGURE 15. 14 Reconstruction of Sinan image encoded using 0.5 bits per pixel and the four-tap Daubechies filter.

15.7 Embedded Zerotree Coder

The embedded zerotree wavelet (EZW) coder was introduced by Shapiro [208]. It is a quantization and coding strategy that incorporates some characteristics of the wavelet decomposition. Just as the quantization and coding approach used in the JPEG standard, which were motivated by the characteristics of the coefficients, were superior to the generic zonal coding algorithms, the EZW approach and its descendants significantly outperform some of the generic approaches. The particular characteristic used by the EZW algorithm is that there are wavelet coefficients in different subbands that represent the same spatial location in the image. If the decomposition is such that the size of the different subbands is different (the first two decompositions in Figure 15.12), then a single coefficient in the smaller subband may represent the same spatial location as multiple coefficients in the other subbands.

In order to put our discussion on more solid ground, consider the 10-band decomposition shown in Figure 15.15. The coefficient a in the upper-left corner of band I represents the same spatial location as coefficients a_1 in band II, a_2 in band III, and a_3 in band IV. In turn, the coefficient a_1 represents the same spatial location as coefficients a_{11} , a_{12} , a_{13} , and a_{14} in band V. Each of these pixels represents the same spatial location as four pixels in band VIII, and so on. In fact, we can visualize the relationships of these coefficients in the form of a tree: The coefficient a forms the root of the tree with three descendants a_1 , a_2 , and a_3 . The coefficient a_1 has descendants a_{11} , a_{12} , a_{13} , and a_{14} . The coefficient a_2 has descendants a_{21} , a_{22} , a_{23} , and a_{24} , and the coefficient a_3 has descendants a_{31} , a_{32} , a_{33} , and a_{34} . Each of these coefficients in turn has four descendants, making a total of 64 coefficients in this tree. A pictorial representation of the tree is shown in Figure 15.16.

Recall that when natural images are decomposed in this manner most of the energy is compacted into the lower bands. Thus, in many cases the coefficients closer to the root of the tree have higher magnitudes than coefficients further away from the root. This means that often if a coefficient has a magnitude less than a given threshold, all its descendants will have magnitudes less than that threshold. In a scalar quantizer, the outer levels of the quantizer correspond to larger magnitudes. Consider the 3-bit quantizer shown in Figure 15.17. If we determine that all coefficients arising from a particular root have magnitudes smaller than T_0 and we inform the decoder of this situation, then for all coefficients in that tree we need only use 2 bits per sample, while getting the same performance as we would have obtained using the 3-bit quantizer. If the binary coding scheme used in Figure 15.17 is used, in which the first bit is the sign bit and the next bit is the most significant bit of the magnitude, then the information that a set of coefficients has value less than T_0 is the same as saying that the most significant bit of the magnitude is 0. If there are N coefficients in the tree, this is a savings of N bits minus however many bits are needed to inform the decoder of this situation.

Before we describe the EZW algorithm, we need to introduce some terminology. Given a threshold T , if a given coefficient has a magnitude greater than T , it is called a *significant* coefficient at level T . If the magnitude of the coefficient is less than T (it is insignificant), and all its descendants have magnitudes less than T , then the coefficient is called a *zerotree root*. Finally, it might happen that the coefficient itself is less than T but some of its descendants have a value greater than T . Such a coefficient is called an *isolated zero*.

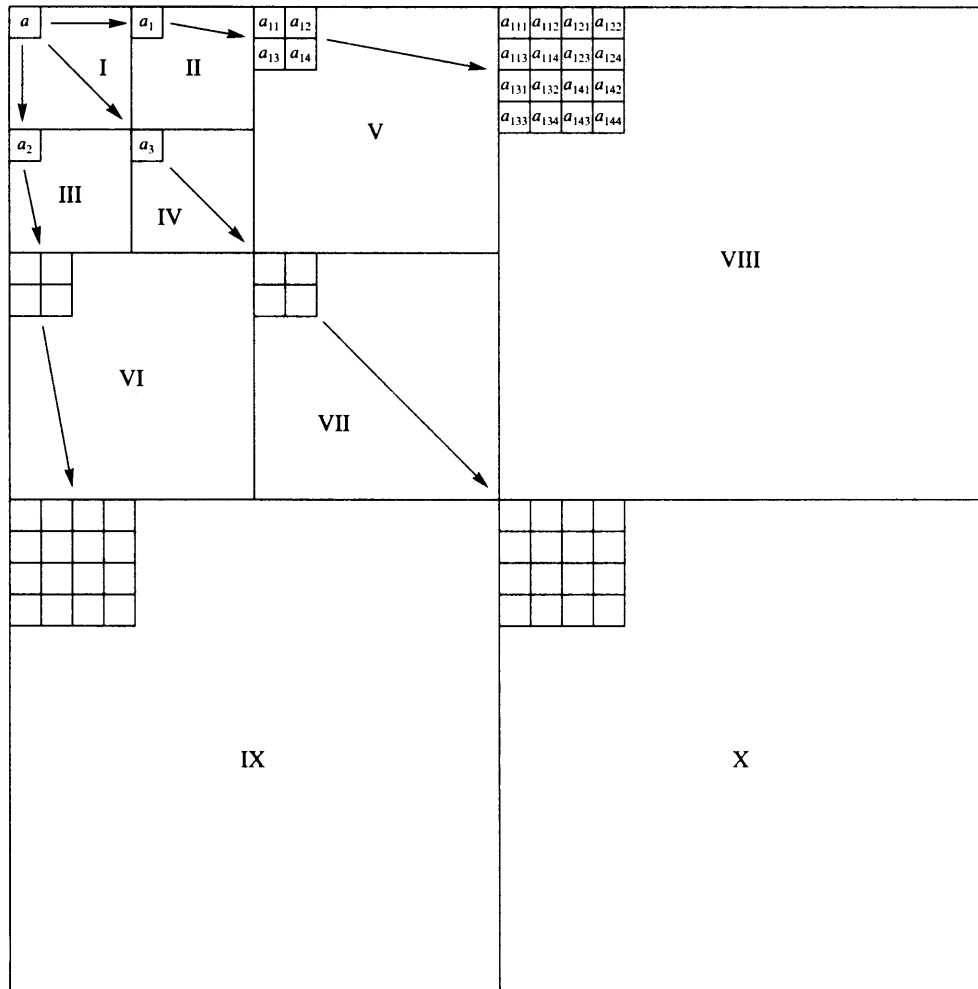


FIGURE 15.15 A 10-band wavelet decomposition.

The EZW algorithm is a multiple-pass algorithm, with each pass consisting of two steps: *significance map encoding* or the *dominant pass*, and *refinement* or the *subordinate pass*. If c_{\max} is the value of the largest coefficient, the initial value of the threshold T_0 is given by

$$T_0 = 2^{\lceil \log_2 c_{\max} \rceil}. \tag{15.91}$$

This selection guarantees that the largest coefficient will lie in the interval $[T_0, 2T_0)$. In each pass, the threshold T_i is reduced to half the value it had in the previous pass:

$$T_i = \frac{1}{2} T_{i-1}. \tag{15.92}$$

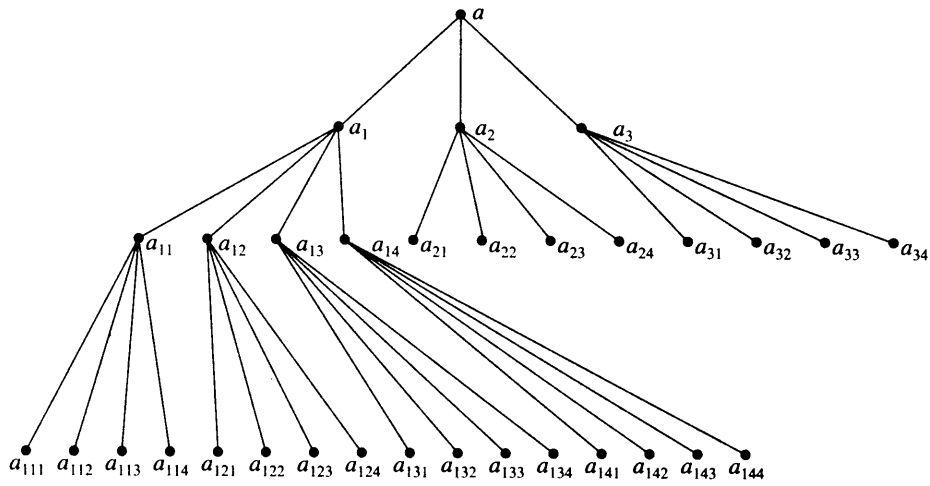


FIGURE 15. 16 Data structure used in the EZW coder.

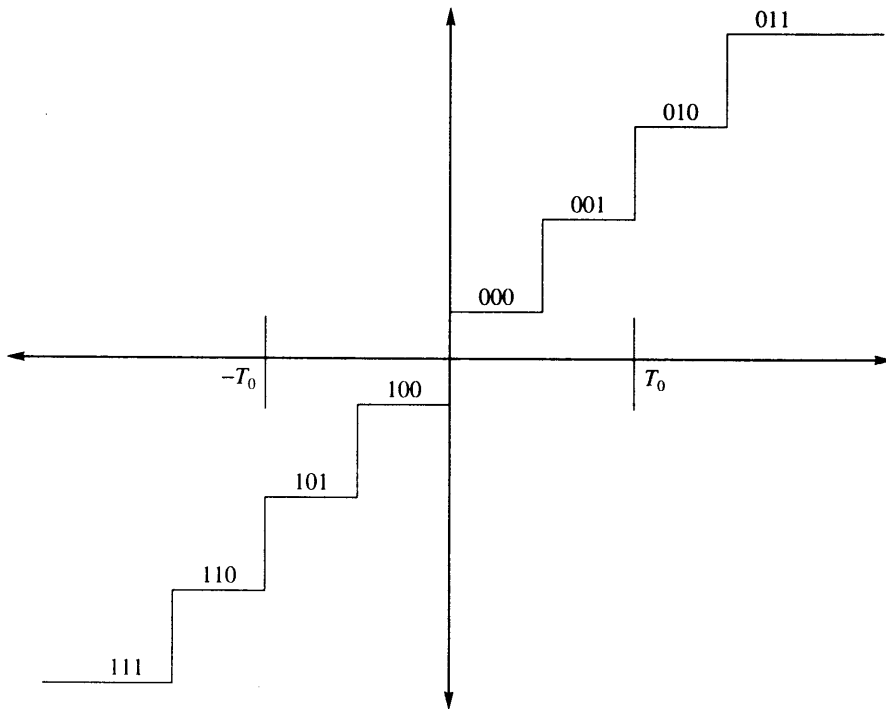


FIGURE 15. 17 A 3-bit midrise quantizer.

For a given value of T_i , we assign one of four possible labels to the coefficients: *significant positive* (*sp*), *significant negative* (*sn*), *zerotree root* (*zr*), and *isolated zero* (*iz*). If we used a fixed-length code, we would need 2 bits to represent each of the labels. Note that when a coefficient has been labeled a zerotree root, we do not need to label its descendants. This assignment is referred to as *significance map coding*.

We can view the significance map coding in part as quantization using a three-level midtread quantizer. This situation is shown in Figure 15.18. The coefficients labeled *significant* are simply those that fall in the outer levels of the quantizer and are assigned an initial reconstructed value of $1.5T_i$ or $-1.5T_i$, depending on whether the coefficient is positive or negative. Note that selecting T_i according to (15.91) and (15.92) guarantees the significant coefficients will lie in the interval $[T, 2T)$. Once a determination of significance has been made, the significant coefficients are included in a list for further refinement in the refinement or subordinate passes. In the refinement pass, we determine whether the coefficient lies in the upper or lower half of the interval $[T, 2T)$. In successive refinement passes, as the value of T is reduced, the interval containing the significant coefficient is narrowed still further and the reconstruction is updated accordingly. An easy way to perform the refinement is to take the difference between the coefficient value and its reconstruction and quantize it using a two-level quantizer with reconstruction values $\pm T/4$. This quantized value is then added on to the current reconstruction value as a correction term.

The wavelet coefficients that have not been previously determined significant are scanned in the manner depicted in Figure 15.19, with each parent node in a tree scanned before its offspring. This makes sense because if the parent is determined to be a zerotree root, we would not need to encode the offspring.

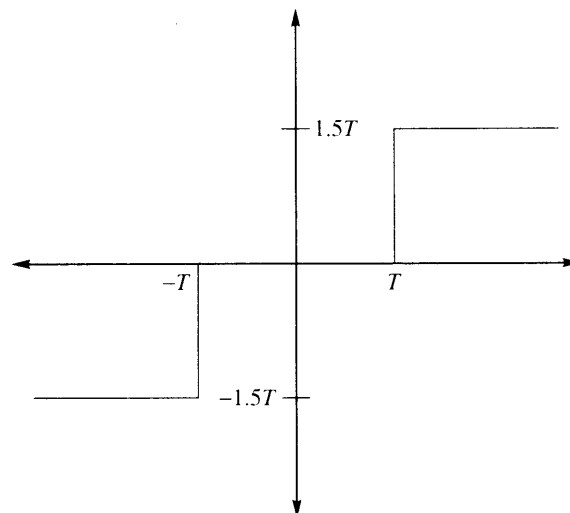


FIGURE 15.18 A three-level midtread quantizer.

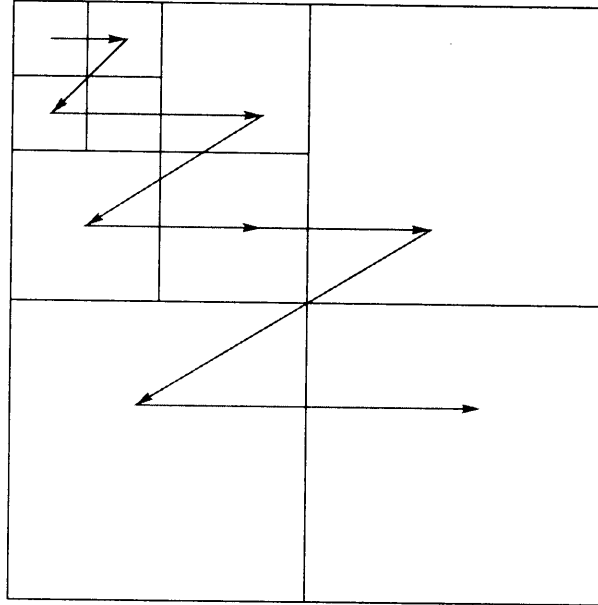


FIGURE 15.19 Scanning of wavelet coefficients for encoding using the EZW algorithm.

Although this may sound confusing, in order to see how simple the encoding procedure actually is, let's use an example.

Example 15.7.1:

Let's use the seven-level decomposition shown below to demonstrate the various steps of EZW:

| | | | |
|----|----|----|----|
| 26 | 6 | 13 | 10 |
| -7 | 7 | 6 | 4 |
| 4 | -4 | 4 | -3 |
| 2 | -2 | -2 | 0 |

To obtain the initial threshold value T_0 , we find the maximum magnitude coefficient, which in this case is 26. Then

$$T_0 = 2^{\lceil \log_2 26 \rceil} = 16.$$

Comparing the coefficients against 16, we find 26 is greater than 16 so we send *sp*. The next coefficient in the scan is 6, which is less than 16. Furthermore, its descendants (13, 10, 6, and 4) are all less than 16. Therefore, 6 is a zerotree root, and we encode this entire set with the label *zr*. The next coefficient in the scan is -7 , which is also a zerotree root, as is 7, the final element in the scan. We do not need to encode the rest of the coefficients separately because they have already been encoded as part of the various zerotrees. The sequence of labels to be transmitted at this point is

sp zr zr zr

Since each label requires 2 bits (for fixed-length encoding), we have used up 8 bits from our bit budget. The only significant coefficient in this pass is the coefficient with a value of 26. We include this coefficient in our list to be refined in the subordinate pass. Calling the subordinate list L_S , we have

$$L_S = \{26\}.$$

The reconstructed value of this coefficient is $1.5T_0 = 24$, and the reconstructed bands look like this:

| | | | |
|----|---|---|---|
| 24 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

The next step is the subordinate pass, in which we obtain a correction term for the reconstruction value of the significant coefficients. In this case, the list L_S contains only one element. The difference between this element and its reconstructed value is $26 - 24 = 2$. Quantizing this with a two-level quantizer with reconstruction levels $\pm T_0/4$, we obtain a correction term of 4. Thus, the reconstruction becomes $24 + 4 = 28$. Transmitting the correction term costs a single bit, therefore at the end of the first pass we have used up 9 bits. Using only these 9 bits, we would obtain the following reconstruction:

| | | | |
|----|---|---|---|
| 28 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

We now reduce the value of the threshold by a factor of two and repeat the process. The value of T_1 is 8. We rescan the coefficients that have not yet been deemed significant. To

emphasize the fact that we do not consider the coefficients that have been deemed significant in the previous pass, we replace them with \star :

| | | | |
|---------|----|----|----|
| \star | 6 | 13 | 10 |
| -7 | 7 | 6 | 4 |
| 4 | -4 | 4 | -3 |
| 2 | -2 | -2 | 0 |

The first coefficient we encounter has a value of 6. This is less than the threshold value of 8; however, the descendants of this coefficient include coefficients with values of 13 and 10. Therefore, this coefficient cannot be classified as a zerotree root. This is an example of what we defined as an isolated zero. The next two coefficients in the scan are -7 and 7 . Both of these coefficients have magnitudes less than the threshold value of 8. Furthermore, all their descendants also have magnitudes less than 8. Therefore, these two coefficients are coded as *zr*. The next two elements in the scan are 13 and 10, which are both coded as *sp*. The final two elements in the scan are 6 and 4. These are both less than the threshold, but they do not have any descendants. We code these coefficients as *iz*. Thus, this dominant pass is coded as

$$iz\ zr\ zr\ sp\ sp\ iz\ iz$$

which requires 14 bits, bringing the total number of bits used to 23. The significant coefficients are reconstructed with values $1.5T_1 = 12$. Thus, the reconstruction at this point is

| | | | |
|----|---|----|----|
| 28 | 0 | 12 | 12 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

We add the new significant coefficients to the subordinate list:

$$L_S = \{26, 13, 10\}.$$

In the subordinate pass, we take the difference between the coefficients and their reconstructions and quantize these to obtain the correction or refinement values for these coefficients. The possible values for the correction terms are $\pm T_1/4 = \pm 2$:

$$\begin{aligned} 26 - 28 &= -2 \Rightarrow \text{Correction term} = -2 \\ 13 - 12 &= 1 \Rightarrow \text{Correction term} = 2 \\ 10 - 12 &= -2 \Rightarrow \text{Correction term} = -2 \end{aligned} \tag{15.93}$$

Each correction requires a single bit, bringing the total number of bits used to 26. With these corrections, the reconstruction at this stage is

| | | | |
|----|---|----|----|
| 26 | 0 | 14 | 10 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

If we go through one more pass, we reduce the threshold value to 4. The coefficients to be scanned are

| | | | |
|----|----|----|----|
| * | 6 | * | * |
| -7 | 7 | 6 | 4 |
| 4 | -4 | 4 | -3 |
| 2 | -2 | -2 | 0 |

The dominant pass results in the following coded sequence:

sp sn sp sp sp sp sn iz iz sp iz iz iz

This pass cost 26 bits, equal to the total number of bits used previous to this pass. The reconstruction upon decoding of the dominant pass is

| | | | |
|----|----|----|----|
| 26 | 6 | 14 | 10 |
| -6 | 6 | 6 | 6 |
| 6 | -6 | 6 | 0 |
| 0 | 0 | 0 | 0 |

The subordinate list is

$$L_S = \{26, 13, 10, 6 - 7, 7, 6, 4, 4, -4, 4\}$$

By now it should be reasonably clear how the algorithm works. We continue encoding until we have exhausted our bit budget or until some other criterion is satisfied. ♦

There are several observations we can make from this example. Notice that the encoding process is geared to provide the most bang for the bit at each step. At each step the bits

are used to provide the maximum reduction in the reconstruction error. If at any time the encoding is interrupted, the reconstruction using this (interrupted) encoding is the best that the algorithm could have provided using this many bits. The encoding improves as more bits are transmitted. This form of coding is called *embedded coding*. In order to enhance this aspect of the algorithm, we can also sort the subordinate list at the end of each pass using information available to both encoder and decoder. This would increase the likelihood of larger coefficients being encoded first, thus providing for a greater reduction in the reconstruction error.

Finally, in the example we determined the number of bits used by assuming fixed-length encoding. In practice, arithmetic coding is used, providing a further reduction in rate.

15.8 Set Partitioning in Hierarchical Trees

The SPIHT (Set Partitioning in Hierarchical Trees) algorithm is a generalization of the EZW algorithm and was proposed by Amir Said and William Pearlman [209]. Recall that in EZW we transmit a lot of information for little cost when we declare an entire subtree to be insignificant and represent all the coefficients in it with a zerotree root label zr . The SPIHT algorithm uses a partitioning of the trees (which in SPIHT are called *spatial orientation trees*) in a manner that tends to keep insignificant coefficients together in larger subsets. The partitioning decisions are binary decisions that are transmitted to the decoder, providing a significance map encoding that is more efficient than EZW. In fact, the efficiency of the significance map encoding in SPIHT is such that arithmetic coding of the binary decisions provides very little gain. The thresholds used for checking significance are powers of two, so in essence the SPIHT algorithm sends the binary representation of the integer value of the wavelet coefficients. As in EZW, the significance map encoding, or set partitioning and ordering step, is followed by a refinement step in which the representations of the significant coefficients are refined.

Let's briefly describe the algorithm and then look at some examples of its operation. However, before we do that we need to get familiar with some notation. The data structure used by the SPIHT algorithm is similar to that used by the EZW algorithm—although not the same. The wavelet coefficients are again divided into trees originating from the lowest resolution band (band I in our case). The coefficients are grouped into 2×2 arrays that, except for the coefficients in band I, are offsprings of a coefficient of a lower resolution band. The coefficients in the lowest resolution band are also divided into 2×2 arrays. However, unlike the EZW case, all but one of them are root nodes. The coefficient in the top-left corner of the array does not have any offsprings. The data structure is shown pictorially in Figure 15.20 for a seven-band decomposition.

The trees are further partitioned into four types of sets, which are sets of coordinates of the coefficients:

- $\mathcal{O}(i, j)$ This is the set of coordinates of the offsprings of the wavelet coefficient at location (i, j) . As each node can either have four offsprings or none, the size of $\mathcal{O}(i, j)$ is either zero or four. For example, in Figure 15.20 the set $\mathcal{O}(0, 1)$ consists of the coordinates of the coefficients $b_1, b_2, b_3,$ and b_4 .

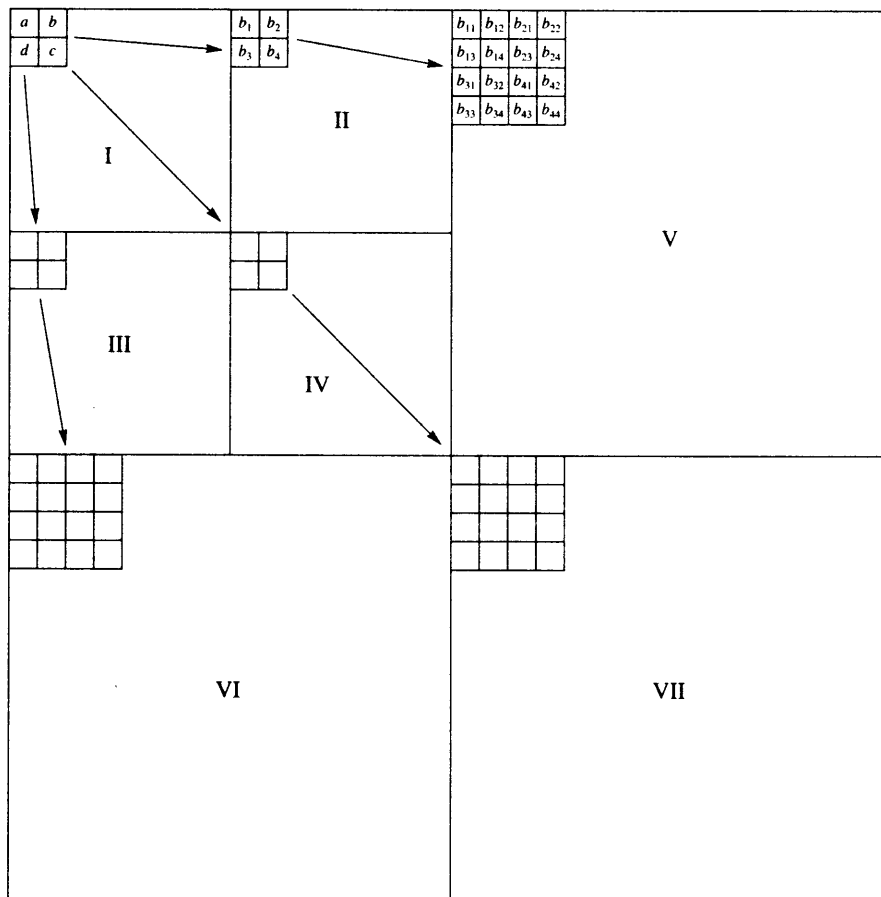


FIGURE 15.20 Data structure used in the SPIHT algorithm.

- $\mathcal{D}(i, j)$ This is the set of all descendants of the coefficient at location (i, j) . Descendants include the offsprings, the offsprings of the offsprings, and so on. For example, in Figure 15.20 the set $\mathcal{D}(0, 1)$ consists of the coordinates of the coefficients $b_1, \dots, b_4, b_{11}, \dots, b_{14}, \dots, b_{44}$. Because the number of offsprings can either be zero or four, the size of $\mathcal{D}(i, j)$ is either zero or a sum of powers of four.
- \mathcal{H} This is the set of all root nodes—essentially band I in the case of Figure 15.20.
- $\mathcal{L}(i, j)$ This is the set of coordinates of all the descendants of the coefficient at location (i, j) except for the immediate offsprings of the coefficient at location (i, j) . In other words,

$$\mathcal{L}(i, j) = \mathcal{D}(i, j) - \mathcal{O}(i, j).$$

In Figure 15.20 the set $\mathcal{L}(0, 1)$ consists of the coordinates of the coefficients $b_{11}, \dots, b_{14}, \dots, b_{44}$.

A set $\mathcal{D}(i, j)$ or $\mathcal{L}(i, j)$ is said to be significant if any coefficient in the set has a magnitude greater than the threshold. Finally, thresholds used for checking significance are powers of two, so in essence the SPIHT algorithm sends the binary representation of the integer value of the wavelet coefficients. The bits are numbered with the least significant bit being the zeroth bit, the next bit being the first significant bit, and the k th bit being referred to as the $k - 1$ most significant bit.

With these definitions under our belt, let us now describe the algorithm. The algorithm makes use of three lists: the *list of insignificant pixels* (LIP), the *list of significant pixels* (LSP), and the *list of insignificant sets* (LIS). The LSP and LIS lists will contain the coordinates of coefficients, while the LIS will contain the coordinates of the roots of sets of type \mathcal{D} or \mathcal{L} . We start by determining the initial value of the threshold. We do this by calculating

$$n = \lfloor \log_2 c_{\max} \rfloor$$

where c_{\max} is the maximum magnitude of the coefficients to be encoded. The LIP list is initialized with the set \mathcal{H} . Those elements of \mathcal{H} that have descendants are also placed in LIS as type \mathcal{D} entries. The LSP list is initially empty.

In each pass, we will first process the members of LIP, then the members of LIS. This is essentially the significance map encoding step. We then process the elements of LSP in the refinement step.

We begin by examining each coordinate contained in LIP. If the coefficient at that coordinate is significant (that is, it is greater than 2^n), we transmit a 1 followed by a bit representing the sign of the coefficient (we will assume 1 for positive, 0 for negative). We then move that coefficient to the LSP list. If the coefficient at that coordinate is not significant, we transmit a 0.

After examining each coordinate in LIP, we begin examining the sets in LIS. If the set at coordinate (i, j) is not significant, we transmit a 0. If the set is significant, we transmit a 1. What we do after that depends on whether the set is of type \mathcal{D} or \mathcal{L} .

If the set is of type \mathcal{D} , we check each of the offsprings of the coefficient at that coordinate. In other words, we check the four coefficients whose coordinates are in $\mathcal{O}(i, j)$. For each coefficient that is significant, we transmit a 1, the sign of the coefficient, and then move the coefficient to the LSP. For the rest we transmit a 0 and add their coordinates to the LIP. Now that we have removed the coordinates of $\mathcal{O}(i, j)$ from the set, what is left is simply the set $\mathcal{L}(i, j)$. If this set is not empty, we move it to the end of the LIS and mark it to be of type \mathcal{L} . Note that this new entry into the LIS has to be examined during *this* pass. If the set is empty, we remove the coordinate (i, j) from the list.

If the set is of type \mathcal{L} , we add each coordinate in $\mathcal{O}(i, j)$ to the end of the LIS as the root of a set of type \mathcal{D} . Again, note that these new entries in the LIS have to be examined during this pass. We then remove (i, j) from the LIS.

Once we have processed each of the sets in the LIS (including the newly formed ones), we proceed to the refinement step. In the refinement step we examine each coefficient that was in the LSP *prior to the current pass* and output the n th most significant bit of $|c_{i,j}|$.

We ignore the coefficients that have been added to the list in this pass because, by declaring them significant at this particular level, we have already informed the decoder of the value of the n th most significant bit.

This completes one pass. Depending on the availability of more bits or external factors, if we decide to continue with the coding process, we decrement n by one and continue. Let's see the functioning of this algorithm on an example.

Example 15.8.1:

Let's use the same example we used for demonstrating the EZW algorithm:

| | | | |
|----|----|----|----|
| 26 | 6 | 13 | 10 |
| -7 | 7 | 6 | 4 |
| 4 | -4 | 4 | -3 |
| 2 | -2 | -2 | 0 |

We will go through three passes at the encoder and generate the transmitted bitstream, then decode this bitstream.

First Pass The value for n in this case is 4. The three lists at the encoder are

$$\text{LIP: } \{(0, 0) \rightarrow 26, (0, 1) \rightarrow 6, (1, 0) \rightarrow -7, (1, 1) \rightarrow 7\}$$

$$\text{LIS: } \{(0, 1)\mathcal{D}, (1, 0)\mathcal{D}, (1, 1)\mathcal{D}\}$$

$$\text{LSP: } \{\}$$

In the listing for LIP, we have included the $\rightarrow \#$ to make it easier to follow the example. Beginning our algorithm, we examine the contents of LIP. The coefficient at location $(0, 0)$ is greater than 16. In other words, it is significant; therefore, we transmit a 1, then a 0 to indicate the coefficient is positive and move the coordinate to LSP. The next three coefficients are all insignificant at this value of the threshold; therefore, we transmit a 0 for each coefficient and leave them in LIP. The next step is to examine the contents of LIS. Looking at the descendants of the coefficient at location $(0, 1)$ (13, 10, 6, and 4), we see that none of them are significant at this value of the threshold so we transmit a 0. Looking at the descendants of c_{10} and c_{11} , we can see that none of these are significant at this value of the threshold. Therefore, we transmit a 0 for each set. As this is the first pass, there are no elements from the previous pass in LSP; therefore, we do not do anything in the refinement pass. We have transmitted a total of 8 bits at the end of this pass (10000000), and the situation of the three lists is as follows:

$$\text{LIP: } \{(0, 1) \rightarrow 6, (1, 0) \rightarrow -7, (1, 1) \rightarrow 7\}$$

$$\text{LIS: } \{(0, 1)\mathcal{D}, (1, 0)\mathcal{D}, (1, 1)\mathcal{D}\}$$

$$\text{LSP: } \{(0, 0) \rightarrow 26\}$$

Second Pass For the second pass we decrement n by 1 to 3, which corresponds to a threshold value of 8. Again, we begin our pass by examining the contents of LIP. There are three elements in LIP. Each is insignificant at this threshold so we transmit three 0s. The next step is to examine the contents of LIS. The first element of LIS is the set containing the descendants of the coefficient at location $(0, 1)$. Of this set, both 13 and 10 are significant at this value of the threshold; in other words, the set $\mathcal{D}(0, 1)$ is significant. We signal this by sending a 1 and examine the offsprings of c_{01} . The first offspring has a value of 13, which is significant and positive, so we send a 1 followed by a 0. The same is true for the second offspring, which has a value of 10. So we send another 1 followed by a 0. We move the coordinates of these two to the LSP. The next two offsprings are both insignificant at this level; therefore, we move these to LIP and transmit a 0 for each. As $\mathcal{L}(0, 1) = \{\}$, we remove $(0, 1)\mathcal{D}$ from LIS. Looking at the other elements of LIS, we can clearly see that both of these are insignificant at this level; therefore, we send a 0 for each. In the refinement pass we examine the contents of LSP from the previous pass. There is only one element in there that is not from the current sorting pass, and it has a value of 26. The third MSB of 26 is 1; therefore, we transmit a 1 and complete this pass. In the second pass we have transmitted 13 bits: 0001101000001. The condition of the lists at the end of the second pass is as follows:

$$\begin{aligned} \text{LIP} &: \{(0, 1) \rightarrow 6, (1, 0) \rightarrow -7, (1, 1) \rightarrow 7, (1, 2) \rightarrow 6, (1, 3) \rightarrow 4\} \\ \text{LIS} &: \{(1, 0)\mathcal{D}, (1, 1)\mathcal{D}\} \\ \text{LSP} &: \{(0, 0) \rightarrow 26, (0, 2) \rightarrow 13, (0, 3) \rightarrow 10\} \end{aligned}$$

Third Pass The third pass proceeds with $n = 2$. As the threshold is now smaller, there are significantly more coefficients that are deemed significant, and we end up sending 26 bits. You can easily verify for yourself that the transmitted bitstream for the third pass is 101110101011001100110000010. The condition of the lists at the end of the third pass is as follows:

$$\begin{aligned} \text{LIP} &: \{(3, 0) \rightarrow 2, (3, 1) \rightarrow -2, (2, 3) \rightarrow -3, (3, 2) \rightarrow -2, (3, 3) \rightarrow 0\} \\ \text{LIS} &: \{\} \\ \text{LSP} &: \{(0, 0) \rightarrow 26, (0, 2) \rightarrow 13, (0, 3) \rightarrow 10, (0, 1) \rightarrow 6, (1, 0) \rightarrow -7, (1, 1) \rightarrow 7, \\ & (1, 2) \rightarrow 6, (1, 3) \rightarrow 4, (2, 0) \rightarrow 4, (2, 1) \rightarrow -4, (2, 2) \rightarrow 4\} \end{aligned}$$

Now for decoding this sequence. At the decoder we also start out with the same lists as the encoder:

$$\begin{aligned} \text{LIP} &: \{(0, 0), (0, 1), (1, 0), (1, 1)\} \\ \text{LIS} &: \{(0, 1)\mathcal{D}, (1, 0)\mathcal{D}, (1, 1)\mathcal{D}\} \\ \text{LSP} &: \{\} \end{aligned}$$

We assume that the initial value of n is transmitted to the decoder. This allows us to set the threshold value at 16. Upon receiving the results of the first pass (10000000), we can see

that the first element of LIP is significant and positive and no other coefficient is significant at this level. Using the same reconstruction procedure as in EZW, we can reconstruct the coefficients at this stage as

| | | | |
|----|---|---|---|
| 24 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

and, following the same procedure as at the encoder, the lists can be updated as

$$\text{LIP: } \{(0, 1), (1, 0), (1, 1)\}$$

$$\text{LIS: } \{(0, 1)\mathcal{D}, (1, 0)\mathcal{D}, (1, 1)\mathcal{D}\}$$

$$\text{LSP: } \{(0, 0)\}$$

For the second pass we decrement n by one and examine the transmitted bitstream: 0001101000001. Since the first 3 bits are 0 and there are only three entries in LIP, all the entries in LIP are still insignificant. The next 9 bits give us information about the sets in LIS. The fourth bit of the received bitstream is 1. This means that the set with root at coordinate $(0,1)$ is significant. Since this set is of type \mathcal{D} , the next bits relate to its offsprings. The 101000 sequence indicates that the first two offsprings are significant at this level and positive and the last two are insignificant. Therefore, we move the first two offsprings to LSP and the last two to LIP. We can also approximate these two significant coefficients in our reconstruction by $1.5 \times 2^3 = 12$. We also remove $(0, 1)\mathcal{D}$ from LIS. The next two bits are both 0, indicating that the two remaining sets are still insignificant. The final bit corresponds to the refinement pass. It is a 1, so we update the reconstruction of the $(0, 0)$ coefficient to $24 + 8/2 = 28$. The reconstruction at this stage is

| | | | |
|----|---|----|----|
| 28 | 0 | 12 | 12 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

and the lists are as follows:

$$\text{LIP: } \{(0, 1), (1, 0), (1, 1), (1, 2), (1, 3)\}$$

$$\text{LIS: } \{(1, 0)\mathcal{D}, (1, 1)\mathcal{D}\}$$

$$\text{LSP: } \{(0, 0), (0, 2), (0, 3)\}$$

For the third pass we again decrement n , which is now 2, giving a threshold value of 4. Decoding the bitstream generated during the third pass (10111010101101100110000010), we update our reconstruction to

| | | | |
|----|----|----|----|
| 26 | 6 | 14 | 10 |
| -6 | 6 | 6 | 6 |
| 6 | -6 | 6 | 0 |
| 0 | 0 | 0 | 0 |

and our lists become

LIP : { (3, 0), (3, 1) }

LIS : { }

LSP : { (0, 0), (0, 2), (0, 3), (0, 1), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (3, 2) }

At this stage we do not have any sets left in LIS and we simply update the values of the coefficients. ♦

Finally, let's look at an example of an image coded using SPIHT. The image shown in Figure 15.21 is the reconstruction obtained from a compressed representation that used 0.5

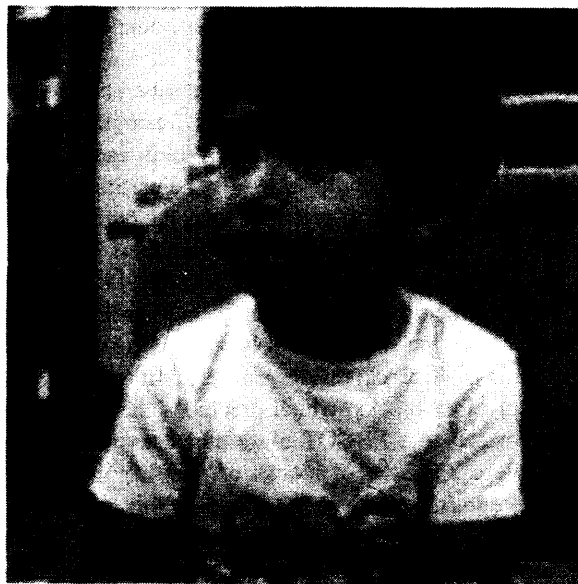


FIGURE 15. 21 Reconstruction of Sinan image encoded using SPIHT at 0.5 bits per pixel.

bits per pixel. (The programs used to generate this image were obtained from the authors) Comparing this with Figure 15.14, we can see a definite improvement in the quality.

Wavelet decomposition has been finding its way into various standards. The earliest example was the FBI fingerprint image compression standard. The latest is the new image compression being developed by the JPEG committee, commonly referred to as JPEG 2000. We take a brief look at the current status of JPEG 2000.

15.9 JPEG 2000

The current JPEG standard provides excellent performance at rates above 0.25 bits per pixel. However, at lower rates there is a sharp degradation in the quality of the reconstructed image. To correct this and other shortcomings, the JPEG committee initiated work on another standard, commonly known as JPEG 2000. The JPEG 2000 is the standard will be based on wavelet decomposition.

There are actually two types of wavelet filters that are included in the standard. One type is the wavelet filters we have been discussing in this chapter. Another type consists of filters that generate integer coefficients; this type is particularly useful when the wavelet decomposition is part of a lossless compression scheme.

The coding scheme is based on a scheme, originally proposed by Taubman [210] and Taubman and Zakhor [211], known as EBCOT. The acronym EBCOT stands for “Embedded Block Coding with Optimized Truncation,” which nicely summarizes the technique. It is a block coding scheme that generates an embedded bitstream. The block coding is independently performed on nonoverlapping blocks within individual subbands. Within a subband all blocks that do not lie on the right or lower boundaries are required to have the same dimensions. A dimension cannot exceed 256.

Embedding and independent block coding seem inherently contradictory. The way EBCOT resolves this contradiction is to organize the bitstream in a succession of layers. Each layer corresponds to a certain distortion level. Within each layer each block is coded with a variable number of bits (which could be zero). The partitioning of bits between blocks is obtained using a Lagrangian optimization that dictates the partitioning or truncation points. The quality of the reproduction is proportional to the numbers of layers received.

The embedded coding scheme is similar in philosophy to the EZW and SPIHT algorithms; however, the data structures used are different. The EZW and SPIHT algorithms used trees of coefficients from the same spatial location across different bands. In the case of the EBCOT algorithm, each block resides entirely within a subband, and each block is coded independently of other blocks, which precludes the use of trees of the type used by EZW and SPIHT. Instead, the EBCOT algorithm uses a quadtree data structure. At the lowest level, we have a 2×2 set of blocks of coefficients. These are, in turn, organized into sets of 2×2 *quads*, and so on. A node in this tree is said to be significant at level n if any of its descendants are significant at that level. A coefficient c_{ij} is said to be significant at level n if $|c_{ij}| \geq 2^n$. As in the case of EZW and SPIHT, the algorithm makes multiple passes, including significance map encoding passes and a magnitude refinement pass. The bits generated during these procedures are encoded using arithmetic coding.

15.10 Summary

In this chapter we have introduced the concepts of wavelets and multiresolution analysis, and we have seen how we can use wavelets to provide an efficient decomposition of signals prior to compression. We have also described several compression techniques based on wavelet decomposition. Wavelets and their applications are currently areas of intensive research.

Further Reading

1. There are a number of excellent introductory books on wavelets. The one I found most accessible was *Introduction to Wavelets and Wavelet Transforms—A Primer*, by C.S. Burrus, R.A. Gopinath, and H. Guo [207].
2. Probably the best mathematical source on wavelets is the book *Ten Lectures on Wavelets*, by I. Daubechies [58].
3. There are a number of tutorials on wavelets available on the Internet. The best source for all matters related to wavelets (and more) on the Internet is “The Wavelet Digest” (<http://www.wavelet.org>). This site includes pointers to many other interesting and useful sites dealing with different applications of wavelets.
4. The JPEG 2000 standard is covered in detail in *JPEG 2000: Image Compression Fundamentals, Standards and Practice*, by D. Taubman and M. Marcellin [212].

15.11 Projects and Problems

1. In this problem we consider the boundary effects encountered when using the short-term Fourier transform. Given the signal

$$f(t) = \sin(2t)$$

- (a) Find the Fourier transform $F(\omega)$ of $f(t)$.
- (b) Find the STFT $F_1(\omega)$ of $f(t)$ using a rectangular window

$$g(t) = \begin{cases} 1 & -2 \leq t \leq 2 \\ 0 & \text{otherwise} \end{cases}$$

for the interval $[-2, 2]$.

- (c) Find the STFT $F_2(\omega)$ of $f(t)$ using a window

$$g(t) = \begin{cases} 1 + \cos(\frac{\pi}{2}t) & -2 \leq t \leq 2 \\ 0 & \text{otherwise.} \end{cases}$$

- (d) Plot $|F(\omega)|$, $|F_1(\omega)|$, and $|F_2(\omega)|$. Comment on the effect of using different window functions.

2. For the function

$$f(t) = \begin{cases} 1 + \sin(2t) & 0 \leq t \leq 1 \\ \sin(2t) & \text{otherwise} \end{cases}$$

using the Haar wavelet find and plot the coefficients $\{c_{j,k}\}$, $j = 0, 1, 2$; $k = 0, \dots, 10$.

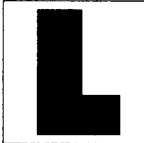
3. For the seven-level decomposition shown below:

| | | | |
|----|----|----|----|
| 21 | 6 | 15 | 12 |
| -6 | 3 | 6 | 3 |
| 3 | -3 | 0 | -3 |
| 3 | 0 | 0 | 0 |

- (a) Find the bitstream generated by the EZW coder.
- (b) Decode the bitstream generated in the previous step. Verify that you get the original coefficient values.
4. Using the coefficients from the seven-level decomposition in the previous problem:
- (a) Find the bitstream generated by the SPIHT coder.
- (b) Decode the bitstream generated in the previous step. Verify that you get the original coefficient values.

Audio Coding

16.1 Overview

ossy compression schemes can be based on a source model, as in the case of speech compression, or a user or sink model, as is somewhat the case in image compression. In this chapter we look at audio compression approaches that are explicitly based on the model of the user. We will look at audio compression approaches in the context of audio compression standards. Principally, we will examine the different MPEG standards for audio compression. These include MPEG Layer I, Layer II, Layer III (or *mp3*) and the Advanced Audio Coding Standard. As with other standards described in this book, the goal here is not to provide all the details required for implementation. Rather the goal is to provide the reader with enough familiarity so that they can then find it much easier to understand these standards.

16.2 Introduction

The various speech coding algorithms we studied in the previous chapter rely heavily on the speech production model to identify structures in the speech signal that can be used for compression. Audio compression systems have taken, in some sense, the opposite tack. Unlike speech signals, audio signals can be generated using a large number of different mechanisms. Lacking a unique model for audio production, the audio compression methods have focused on the unique model for audio perception, a psychoacoustic model for hearing. At the heart of the techniques described in this chapter is a psychoacoustic model of human perception. By identifying what can and, more important what cannot be heard, the schemes described in this chapter obtain much of their compression by discarding information that cannot be perceived. The motivation for the development of many of these perceptual coders was their potential application in broadcast multimedia. However, their major impact has been in the distribution of audio over the Internet.

We live in an environment rich in auditory stimuli. Even an environment described as quiet is filled with all kinds of natural and artificial sounds. The sounds are always present and come to us from all directions. Living in this stimulus-rich environment, it is essential that we have mechanisms for ignoring some of the stimuli and focusing on others. Over the course of our evolutionary history we have developed limitations on what we can hear. Some of these limitations are physiological, based on the machinery of hearing. Others are psychological, based on how our brain processes auditory stimuli. The insight of researchers in audio coding has been the understanding that these limitations can be useful in selecting information that needs to be encoded and information that can be discarded. The limitations of human perception are incorporated into the compression process through the use of psychoacoustic models. We briefly describe the auditory model used by the most popular audio compression approaches. Our description is necessarily superficial and we refer readers interested in more detail to [97, 194].

The machinery of hearing is frequency dependent. The variation of what is perceived as equally loud at different frequencies was first measured by Fletcher and Munson at Bell Labs in the mid-1930s [96]. These measurements of perceptual equivalence were later refined by Robinson and Dadson. This dependence is usually displayed as a set of equal loudness curves, where the sound pressure level (SPL) is plotted as a function of frequency for tones perceived to be equally loud. Clearly, what two people think of as equally loud will be different. Therefore, these curves are actually averages and serve as a guide to human auditory perception. The particular curve that is of special interest to us is the threshold-of-hearing curve. This is the SPL curve that delineates the boundary of audible and inaudible sounds at different frequencies. In Figure 16.1 we show a plot of this audibility threshold in quiet. Sounds that lie below the threshold are not perceived by humans. Thus, we can see that a low amplitude sound at a frequency of 3 kHz may be perceptible while the same level of sound at 100 Hz would not be perceived.

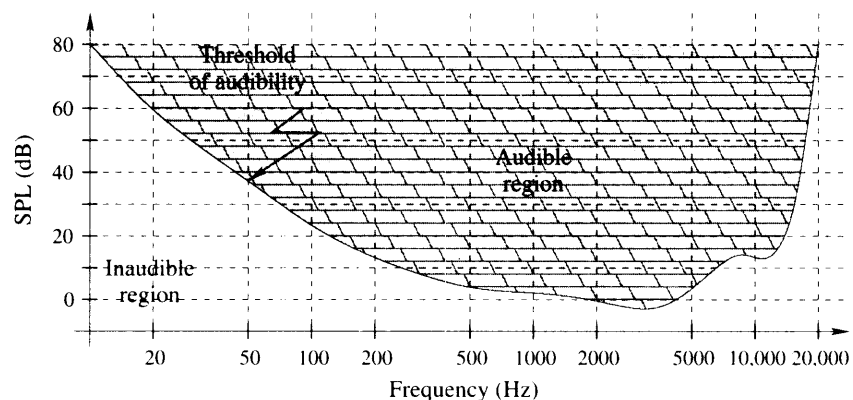


FIGURE 16.1 A typical plot of the audibility threshold.

16.2.1 Spectral Masking

Lossy compression schemes require the use of quantization at some stage. Quantization can be modeled as an additive noise process in which the output of the quantizer is the input plus the quantization noise. To hide quantization noise, we can make use of the fact that signals below a particular amplitude at a particular frequency are not audible. If we select the quantizer step size such that the quantization noise lies below the audibility threshold, the noise will not be perceived. Furthermore, the threshold of audibility is not absolutely fixed and typically rises when multiple sounds impinge on the human ear. This phenomenon gives rise to *spectral masking*. A tone at a certain frequency will raise the threshold in a *critical band* around that frequency. These critical bands have a constant Q , which is the ratio of frequency to bandwidth. Thus, at low frequencies the critical band can have a bandwidth as low as 100 Hz, while at higher frequencies the bandwidth can be as large as 4 kHz. This increase of the threshold has major implications for compression. Consider the situation in Figure 16.2. Here a tone at 1 kHz has raised the threshold of audibility so that the adjacent tone above it in frequency is no longer audible. At the same time, while the tone at 500 Hz is audible, because of the increase in the threshold the tone can be quantized more crudely. This is because increase of the threshold will allow us to introduce more quantization noise at that frequency. The degree to which the threshold is increased depends on a variety of factors, including whether the signal is sinusoidal or atonal.

16.2.2 Temporal Masking

Along with spectral masking, the psychoacoustic coders also make use of the phenomenon of temporal masking. The temporal masking effect is the masking that occurs when a sound raises the audibility threshold for a brief interval preceding and following the sound. In Figure 16.3 we show the threshold of audibility close to a masking sound. Sounds that occur in an interval around the masking sound (both after and before the masking tone) can be masked. If the masked sound occurs prior to the masking tone, this is called premasking

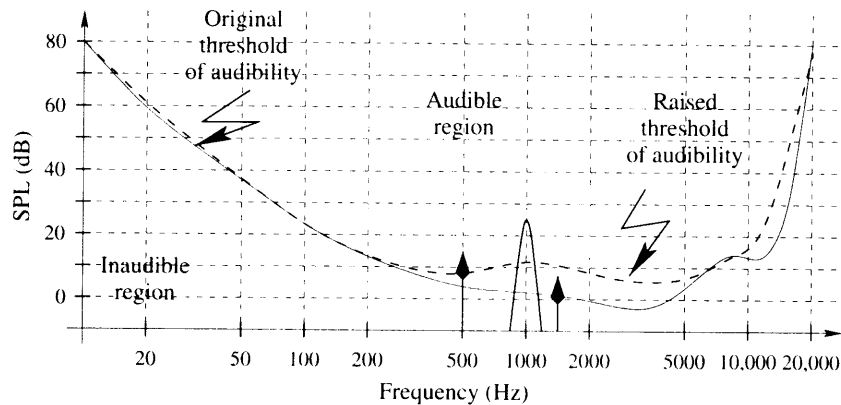


FIGURE 16.2 Change in the audibility threshold.

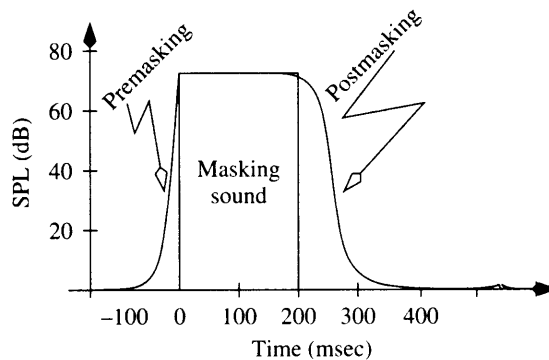


FIGURE 16.3 Change in the audibility threshold in time.

or backward masking, and if the sound being masked occurs after the masking tone this effect is called postmasking or forward masking. The forward masking remains in effect for a much longer time interval than the backward masking.

16.2.3 Psychoacoustic Model

These attributes of the ear are used by all algorithms that use a psychoacoustic model. There are two models used in the MPEG audio coding algorithms. Although they differ in some details, the general approach used in both cases is the same. The first step in the psychoacoustic model is to obtain a spectral profile of the signal being encoded. The audio input is windowed and transformed into the frequency domain using a filter bank or a frequency domain transform. The Sound Pressure Level (SPL) is calculated for each spectral band. If the algorithm uses a subband approach, then the SPL for the band is computed from the SPL for each coefficient X_k . Because tonal and nontonal components have different effects on the masking level, the next step is to determine the presence and location of these components. The presence of any tonal components is determined by first looking for local maxima where a local maximum is declared at location k if $|X_k|^2 > |X_{k-1}|^2$ and $|X_k|^2 \geq |X_{k+1}|^2$. A local maximum is determined to be a tonal component if

$$20 \log_{10} \frac{|X_k|}{|X_{k+j}|} \geq 7$$

where the values j depend on the frequency. The identified tonal maskers are removed from each critical band and the power of the remaining spectral lines in the band is summed to obtain the nontonal masking level. Once all the maskers are identified, those with SPL below the audibility threshold are removed. Furthermore, of those maskers that are very close to each other in frequency, the lower-amplitude masker is removed. The effects of the remaining maskers are obtained using a spreading function that models spectral masking. Finally, the masking due to the audibility level and the maskers is combined to give the final masking thresholds. These thresholds are then used in the coding process.

In the following sections we describe the various audio coding algorithms used in the MPEG standards. Although these algorithms provide audio that is perceptually noiseless, it is important to remember that even if we cannot perceive it, there is quantization noise distorting the original signal. This becomes especially important if the reconstructed audio signal goes through any postprocessing. Postprocessing may change some of the audio components, making the previously masked quantization noise audible. Therefore, if there is any kind of processing to be done, including mixing or equalization, the audio should be compressed only after the processing has taken place. This “hidden noise” problem also prevents multiple stages of encoding and decoding or tandem coding.

16.3 MPEG Audio Coding

We begin with the three separate, stand-alone audio compression strategies that are used in MPEG-1 and MPEG-2 and known as Layer I, Layer II, and Layer III. The Layer III audio compression algorithm is also referred to as *mp3*. Most standards have *normative* sections and *informative* sections. The *normative* actions are those that are required for compliance to the standard. Most current standards, including the MPEG standards, define the bitstream that should be presented to the decoder, leaving the design of the encoder to individual vendors. That is, the bitstream definition is normative, while most guidance about encoding is informative. Thus, two MPEG-compliant bitstreams that encode the same audio material at the same rate but on different encoders may sound very different. On the other hand, a given MPEG bitstream decoded on different decoders will result in essentially the same output.

A simplified block diagram representing the basic strategy used in all three layers is shown in Figure 16.4. The input, consisting of 16-bit PCM words, is first transformed to the frequency domain. The frequency coefficients are quantized, coded, and packed into an MPEG bitstream. Although the overall approach is the same for all layers, the details can vary significantly. Each layer is progressively more complicated than the previous layer and also provides higher compression. The three layers are backward compatible. That is, a decoder for Layer III should be able to decode Layer I- and Layer II-encoded audio. A decoder for Layer II should be able to decode Layer I- encoded audio. Notice the existence of a block labeled *Psychoacoustic model* in Figure 16.4.

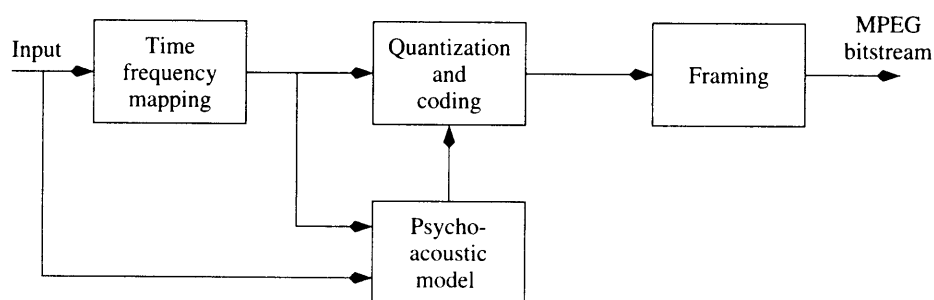


FIGURE 16.4 The MPEG audio coding algorithms.

16.3.1 Layer I Coding

The Layer I coding scheme provides a 4:1 compression. In Layer I coding the time frequency mapping is accomplished using a bank of 32 subband filters. The output of the subband filters is critically sampled. That is, the output of each filter is down-sampled by 32. The samples are divided into groups of 12 samples each. Twelve samples from each of the 32 subband filters, or a total of 384 samples, make up one frame of the Layer I coder. Once the frequency components are obtained the algorithm examines each group of 12 samples to determine a *scalefactor*. The scalefactor is used to make sure that the coefficients make use of the entire range of the quantizer. The subband output is divided by the scalefactor before being linearly quantized. There are a total of 63 scalefactors specified in the MPEG standard. Specification of each scalefactor requires 6 bits.

To determine the number of bits to be used for quantization, the coder makes use of the psychoacoustic model. The inputs to the model include an the Fast Fourier Transform (FFT) of the audio data as well as the signal itself. The model calculates the masking thresholds in each subband, which in turn determine the amount of quantization noise that can be tolerated and hence the quantization step size. As the quantizers all cover the same range, selection of the quantization stepsize is the same as selection of the number of bits to be used for quantizing the output of each subband. In Layer I the encoder has a choice of 14 different quantizers for each band (plus the option of assigning 0 bits). The quantizers are all midtreed quantizers ranging from 3 levels to 65,535 levels. Each subband gets assigned a variable number of bits. However, the total number of bits available to represent all the subband samples is fixed. Therefore, the bit allocation can be an iterative process. The objective is to keep the noise-to-mask ratio more or less constant across the subbands.

The output of the quantization and bit allocation steps are combined into a frame as shown in Figure 16.5. Because MPEG audio is a streaming format, each frame carries a header, rather than having a single header for the entire audio sequence. The header is made up of 32 bits. The first 12 bits comprise a sync pattern consisting of all 1s. This is followed by a 1-bit version ID, a 2-bit layer indicator, a 1-bit CRC protection. The CRC protection bit is set to 0 if there is no CRC protection and is set to a 1 if there is CRC protection. If the layer and protection information is known, all 16 bits can be used for providing frame synchronization. The next 4 bits make up the bit rate index, which specifies the bit rate in kbits/sec. There

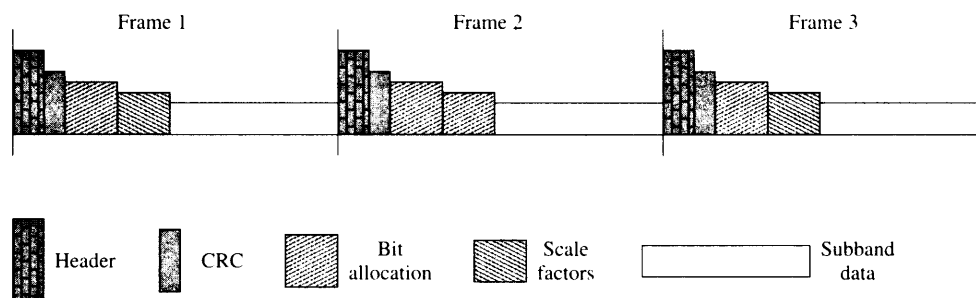


FIGURE 16.5 Frame structure for Layer I.

TABLE 16.1 Allowable sampling frequencies in MPEG-1 and MPEG-2.

| Index | MPEG-1 | MPEG-2 |
|-------|----------|-----------|
| 00 | 44.1 kHz | 22.05 kHz |
| 01 | 48 kHz | 24 kHz |
| 10 | 32 kHz | 16 kHz |
| 11 | Reserved | |

are 14 specified bit rates to choose from. This is followed by 2 bits that indicate the sampling frequency. The sampling frequencies for MPEG-1 and MPEG-2 are different (one of the few differences between the audio coding standards for MPEG-1 and MPEG-2) and are shown in Table 16.1. These bits are followed by a single padding bit. If the bit is “1,” the frame needs an additional bit to adjust the bit rate to the sampling frequency. The next two bits indicate the mode. The possible modes are “stereo,” “joint stereo,” “dual channel,” and “single channel.” The stereo mode consists of two channels that are encoded separately but intended to be played together. The joint stereo mode consists of two channels that are encoded together. The left and right channels are combined to form a *mid* and a *side* signal as follows:

$$M = \frac{L + R}{2}$$

$$S = \frac{L - R}{2}$$

The dual channel mode consists of two channels that are encoded separately and are not intended to be played together, such as a translation channel. These are followed by two mode extension bits that are used in the joint stereo mode. The next bit is a copyright bit (“1” if the material is copy righted, “0” if it is not). The next bit is set to “1” for original media and “0” for copy. The final two bits indicate the type of de-emphasis to be used.

If the CRC bit is set, the header is followed by a 16-bit CRC. This is followed by the bit allocations used by each subband and is in turn followed by the set of 6-bit scalefactors. The scalefactor data is followed by the quantized 384 samples.

16.3.2 Layer II Coding

The Layer II coder provides a higher compression rate by making some relatively minor modifications to the Layer I coding scheme. These modifications include how the samples are grouped together, the representation of the scalefactors, and the quantization strategy. Where the Layer I coder puts 12 samples from each subband into a frame, the Layer II coder groups three sets of 12 samples from each subband into a frame. The total number of samples per frame increases from 384 samples to 1152 samples. This reduces the amount of overhead per sample. In Layer I coding a separate scalefactor is selected for each block of 12 samples. In Layer II coding the encoder tries to share a scale factor among two or all three groups of samples from each subband filter. The only time separate scalefactors are used

for each group of 12 samples is when not doing so would result in a significant increase in distortion. The particular choice used in a frame is signaled through the *scalefactor selection information* field in the bitstream.

The major difference between the Layer I and Layer II coding schemes is in the quantization step. In the Layer I coding scheme the output of each subband is quantized using one of 14 possibilities; the same 14 possibilities for each of the subbands. In Layer II coding the quantizers used for each of the subbands can be selected from a different set of quantizers depending on the sampling rate and the bit rates. For some sampling rate and bit rate combinations, many of the higher subbands are assigned 0 bits. That is, the information from those subbands is simply discarded. Where the quantizer selected has 3, 5, or 9 levels, the Layer II coding scheme uses one more enhancement. Notice that in the case of 3 levels we have to use 2 bits per sample, which would have allowed us to represent 4 levels. The situation is even worse in the case of 5 levels, where we are forced to use 3 bits, wasting three codewords, and in the case of 9 levels where we have to use 4 bits, thus wasting 7 levels. To avoid this situation, the Layer II coder groups 3 samples into a *granule*. If each sample can take on 3 levels, a granule can take on 27 levels. This can be accommodated using 5 bits. If each sample had been encoded separately we would have needed 6 bits. Similarly, if each sample can take on 9 values, a granule can take on 729 values. We can represent 729 values using 10 bits. If each sample in the granule had been encoded separately, we would have needed 12 bits. Using all these savings, the compression ratio in Layer II coding can be increased from 4:1 to 8:1 or 6:1.

The frame structure for the Layer II coder can be seen in Figure 16.6. The only real difference between this frame structure and the frame structure of the Layer I coder is the scalefactor selection information field.

16.3.3 Layer III Coding—mp3

Layer III coding, which has become widely popular under the name *mp3*, is considerably more complex than the Layer I and Layer II coding schemes. One of the problems with the Layer I and coding schemes was that with the 32-band decomposition, the bandwidth of the subbands at lower frequencies is significantly larger than the critical bands. This

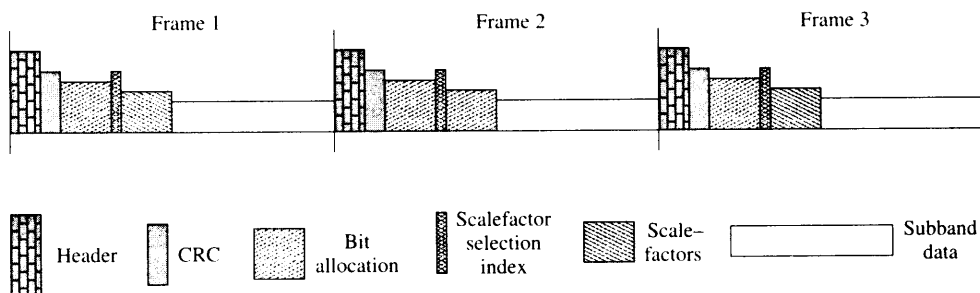


FIGURE 16.6 Frame structure for Layer 2.

makes it difficult to make an accurate judgement of the mask-to-signal ratio. If we get a high amplitude tone within a subband and if the subband was narrow enough, we could assume that it masked other tones in the band. However, if the bandwidth of the subband is significantly higher than the critical bandwidth at that frequency, it becomes more difficult to determine whether other tones in the subband will be masked.

A simple way to increase the spectral resolution would be to decompose the signal directly into a higher number of bands. However, one of the requirements on the Layer III algorithm is that it be backward compatible with Layer I and Layer II coders. To satisfy this backward compatibility requirement, the spectral decomposition in the Layer III algorithm is performed in two stages. First the 32-band subband decomposition used in Layer I and Layer II is employed. The output of each subband is transformed using a modified discrete cosine transform (MDCT) with a 50% overlap. The Layer III algorithm specifies two sizes for the MDCT, 6 or 18. This means that the output of each subband can be decomposed into 18 frequency coefficients or 6 frequency coefficients.

The reason for having two sizes for the MDCT is that when we transform a sequence into the frequency domain, we lose time resolution even as we gain frequency resolution. The larger the block size the more we lose in terms of time resolution. The problem with this is that any quantization noise introduced into the frequency coefficients will get spread over the entire block size of the transform. Backward temporal masking occurs for only a short duration prior to the masking sound (approximately 20 msec). Therefore, quantization noise will appear as a *pre-echo*. Consider the signal shown in Figure 16.7. The sequence consists of 128 samples, the first 118 of which are 0, followed by a sharp increase in value. The 128-point DCT of this sequence is shown in Figure 16.8. Notice that many of these coefficients are quite large. If we were to send all these coefficients, we would have data expansion instead of data compression. If we keep only the 10 largest coefficients, the

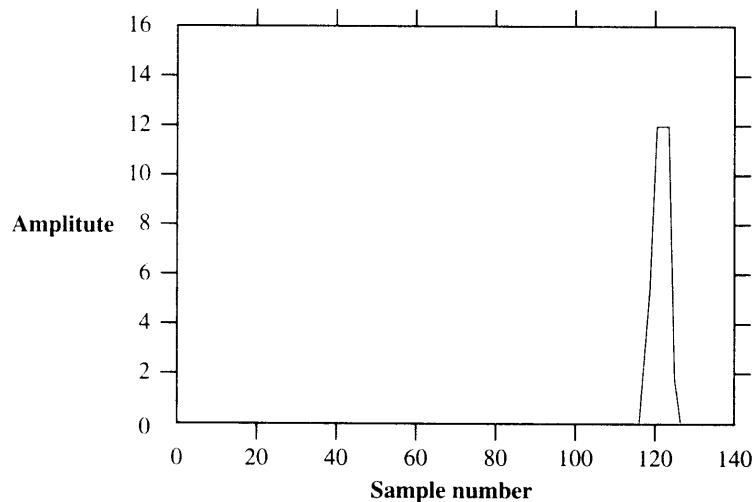


FIGURE 16.7 Source output sequence.

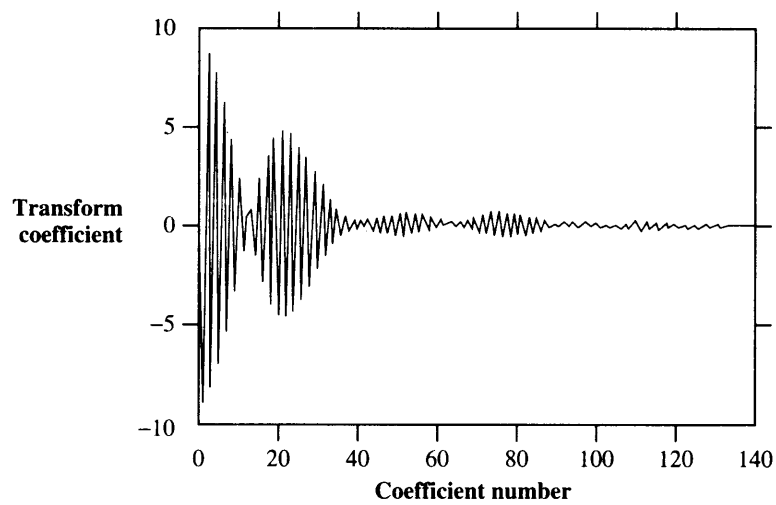


FIGURE 16.8 Transformed sequence.

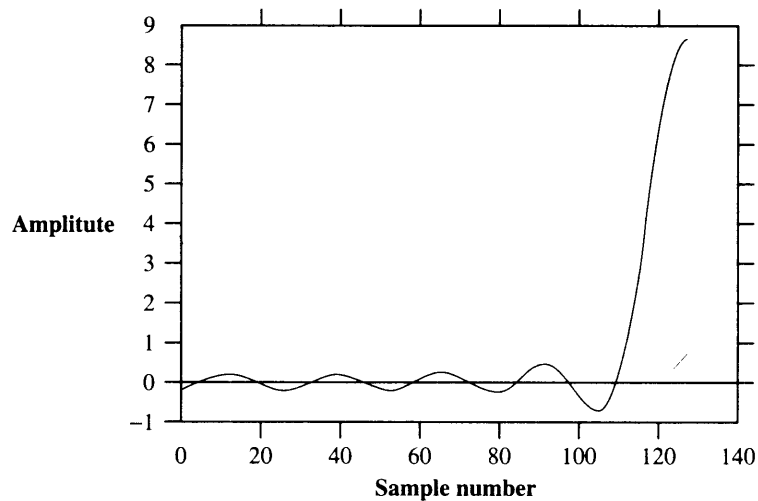


FIGURE 16.9 Reconstructed sequence from 10 DCT coefficients.

reconstructed signal is shown in Figure 16.9. Notice that not only are the nonzero signal values not well represented, there is also error in the samples prior to the change in value of the signal. If this were an audio signal and the large values had occurred at the beginning of the sequence, the forward masking effect would have reduced the perceptibility of the quantization error. In the situation shown in Figure 16.9, backward masking will mask some of the quantization error. However, backward masking occurs for only a short duration prior

to the masking sound. Therefore, if the length of the block in question is longer than the masking interval, the distortion will be evident to the listener.

If we get a sharp sound that is very limited in time (such as the sound of castanets) we would like to keep the block size small enough that it can contain this sharp sound. Then, when we incur quantization noise it will not get spread out of the interval in which the actual sound occurred and will therefore get masked. The Layer III algorithm monitors the input and where necessary substitutes three short transforms for one long transform. What actually happens is that the subband output is multiplied by a window function of length 36 during the stationary periods (that is a blocksize of 18 plus 50% overlap from neighboring blocks). This window is called the *long window*. If a sharp attack is detected, the algorithm shifts to a sequence of three *short windows* of length 12 after a transition window of length 30. This initial transition window is called the *start* window. If the input returns to a more stationary mode, the short windows are followed by another transition window called the *stop* window of length 30 and then the standard sequence of long windows. The process of transitioning between windows is shown in Figure 16.10. A possible set of window transitions is shown in Figure 16.11. For the long windows we end up with 18 frequencies per subband, resulting in a total of 576 frequencies. For the short windows we get 6 coefficients per subband for a total of 192 frequencies. The standard allows for a mixed block mode in which the two lowest subbands use long windows while the remaining subbands use short windows. Notice that while the number of frequencies may change depending on whether we are using long or short windows, the number of samples in a frame stays at 1152. That is 36 samples, or 3 groups of 12, from each of the 32 subband filters.

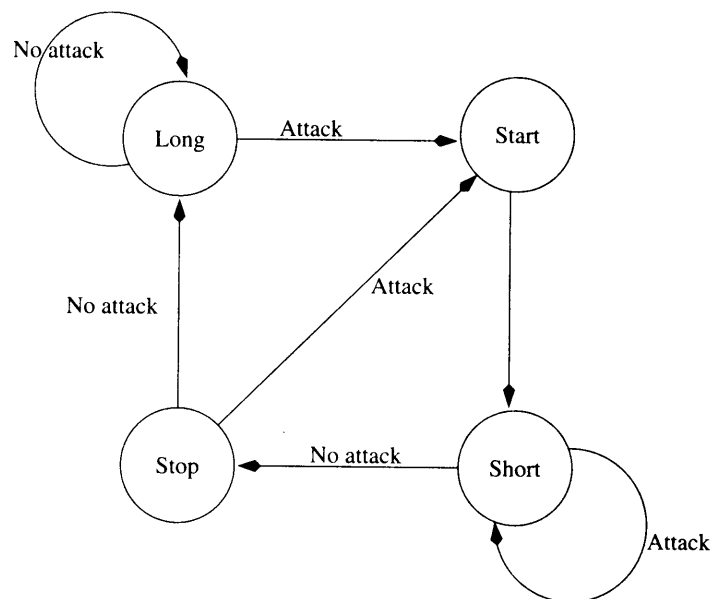


FIGURE 16. 10 State diagram for the window switching process.

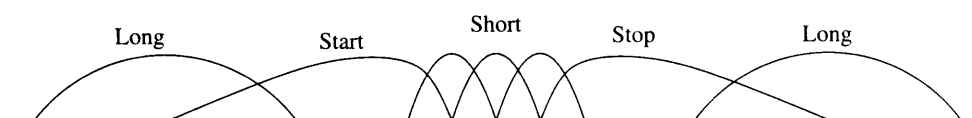


FIGURE 16. 11 Sequence of windows.

The coding and quantization of the output of the MDCT is conducted in an iterative fashion using two nested loops. There is an outer loop called the *distortion control loop* whose purpose is to ensure that the introduced quantization noise lies below the audibility threshold. The scalefactors are used to control the level of quantization noise. In Layer III scalefactors are assigned to groups or “bands” of coefficients in which the bands are approximately the size of critical bands. There are 21 scalefactor bands for long blocks and 12 scalefactor bands for short blocks.

The inner loop is called the *rate control loop*. The goal of this loop is to make sure that a target bit rate is not exceeded. This is done by iterating between different quantizers and Huffman codes. The quantizers used in *mp3* are companded nonuniform quantizers. The scaled MDCT coefficients are first quantized and organized into regions. Coefficients at the higher end of the frequency scale are likely to be quantized to zero. These consecutive zero outputs are treated as a single region and the run-length is Huffman encoded. Below this region of zero coefficients, the encoder identifies the set of coefficients that are quantized to 0 or ± 1 . These coefficients are grouped into groups of four. This set of quadruplets is the second region of coefficients. Each quadruplet is encoded using a single Huffman codeword. The remaining coefficients are divided into two or three subregions. Each subregion is assigned a Huffman code based on its statistical characteristics. If the result of using this variable length coding exceeds the bit budget, the quantizer is adjusted to increase the quantization stepsize. The process is repeated until the target rate is satisfied.

Once the target rate is satisfied, control passes back to the outer, distortion control loop. The psychoacoustic model is used to check whether the quantization noise in any band exceeds the allowed distortion. If it does, the scalefactor is adjusted to reduce the quantization noise. Once all scalefactors have been adjusted, control returns to the rate control loop. The iterations terminate either when the distortion and rate conditions are satisfied or the scalefactors cannot be adjusted any further.

There will be frames in which the number of bits used by the Huffman coder is less than the amount allocated. These bits are saved in a conceptual *bit reservoir*. In practice what this means is that the start of a block of data does not necessarily coincide with the header of the frame. Consider the three frames shown in Figure 16.12. In this example, the main data for the first frame (which includes scalefactor information and the Huffman coded data) does not occupy the entire frame. Therefore, the main data for the second frame starts before the second frame actually begins. The same is true for the remaining data. The main data can begin in the *previous frame*. However, the main data for a particular frame cannot spill over into the *following frame*.

All this complexity allows for a very efficient encoding of audio inputs. The typical *mp3* audio file has a compression ratio of about 10:1. In spite of this high level of compression, most people cannot tell the difference between the original and the compressed representation.

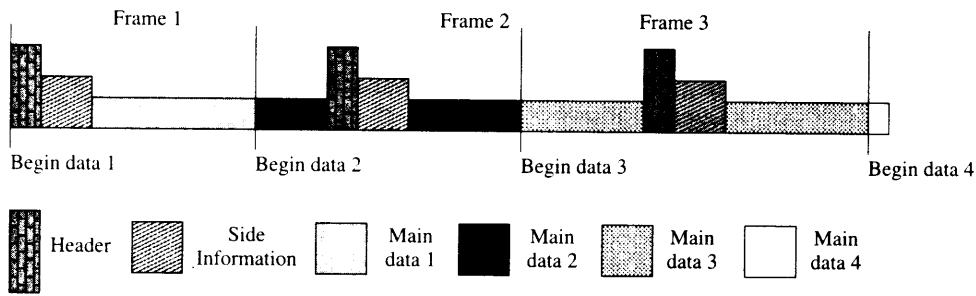


FIGURE 16.12 Sequence of windows.

We say most because trained professionals can at times tell the difference between the original and compressed versions. People who can identify very minute differences between coded and original signals have played an important role in the development of audio coders. By identifying where distortion may be audible they have helped focus effort onto improving the coding process. This development process has made *mp3* the format of choice for compressed music.

16.4 MPEG Advanced Audio Coding

The MPEG Layer III algorithm has been highly successful. However, it had some built-in drawbacks because of the constraints under which it had been designed. The principal constraint was the requirement that it be backward compatible. This requirement for backward compatibility forced the rather awkward decomposition structure involving a subband decomposition followed by an MDCT decomposition. The period immediately following the release of the MPEG specifications also saw major developments in hardware capability. The Advanced Audio Coding (AAC) standard was approved as a higher quality multichannel alternative to the backward compatible MPEG Layer III in 1997.

The AAC approach is a modular approach based on a set of self-contained tools or modules. Some of these tools are taken from the earlier MPEG audio standard while others are new. As with previous standards, the AAC standard actually specifies the decoder. The decoder tools specified in the AAC standard are listed in Table 16.2. As shown in the table, some of these tools are required for all profiles while others are only required for some profiles. By using some or all of these tools, the standard describes three profiles. These are the *main* profile, the *low complexity* profile, and the *sampling-rate-scalable* profile. The AAC approach used in MPEG-2 was later enhanced and modified to provide an audio coding option in MPEG-4. In the following section we first describe the MPEG-2 AAC algorithm, followed by the MPEG-4 AAC algorithm.

16.4.1 MPEG-2 AAC

A block diagram of an MPEG-2 AAC encoder is shown in Figure 16.13. Each block represents a tool. The psychoacoustic model used in the AAC encoder is the same as the

TABLE 16.2 AAC Decoder Tools [213].

| Tool Name | |
|---------------------------------|----------|
| Bitstream Formatter | Required |
| Huffman Decoding | Required |
| Inverse Quantization | Required |
| Rescaling | Required |
| M/S | Optional |
| Interblock Prediction | Optional |
| Intensity | Optional |
| Dependently Switched Coupling | Optional |
| TNS | Optional |
| Block switching / MDCT | Required |
| Gain Control | Optional |
| Independently Switched Coupling | Optional |

model used in the MPEG Layer III encoder. As in the Layer III algorithm, the psychoacoustic model is used to trigger switching in the blocklength of the MDCT transform and to produce the threshold values used to determine scalefactors and quantization thresholds. The audio data is fed in parallel to both the acoustic model and to the modified Discrete Cosine Transform.

Block Switching and MDCT

Because the AAC algorithm is not backward compatible it does away with the requirement of the 32-band filterbank. Instead, the frequency decomposition is accomplished by a Modified Discrete Cosine Transform (MDCT). The MDCT is described in Chapter 13. The AAC algorithm allows switching between a window length of 2048 samples and 256 samples. These window lengths include a 50% overlap with neighboring blocks. So 2048 time samples are used to generate 1024 spectral coefficients, and 256 time samples are used to generate 128 frequency coefficients. The k^{th} spectral coefficient of block i , $X_{i,k}$ is given by:

$$X_{i,k} = 2 \sum_{n=0}^{N-1} z_{i,n} \cos \left(\frac{2\pi(n+n_o)}{N} \left(k + \frac{1}{2} \right) \right)$$

where $z_{i,n}$ is the n^{th} time sample of the i^{th} block, N is the window length and

$$n_o = \frac{N/2 + 1}{2}.$$

The longer block length allows the algorithm to take advantage of stationary portions of the input to get significant improvements in compression. The short block length allows the algorithm to handle sharp attacks without incurring substantial distortion and rate penalties. Short blocks occur in groups of eight in order to avoid framing issues. As in the case of MPEG Layer III, there are four kinds of windows: long, short, start, and stop. The decision about whether to use a group of short blocks is made by the psychoacoustic model. The coefficients

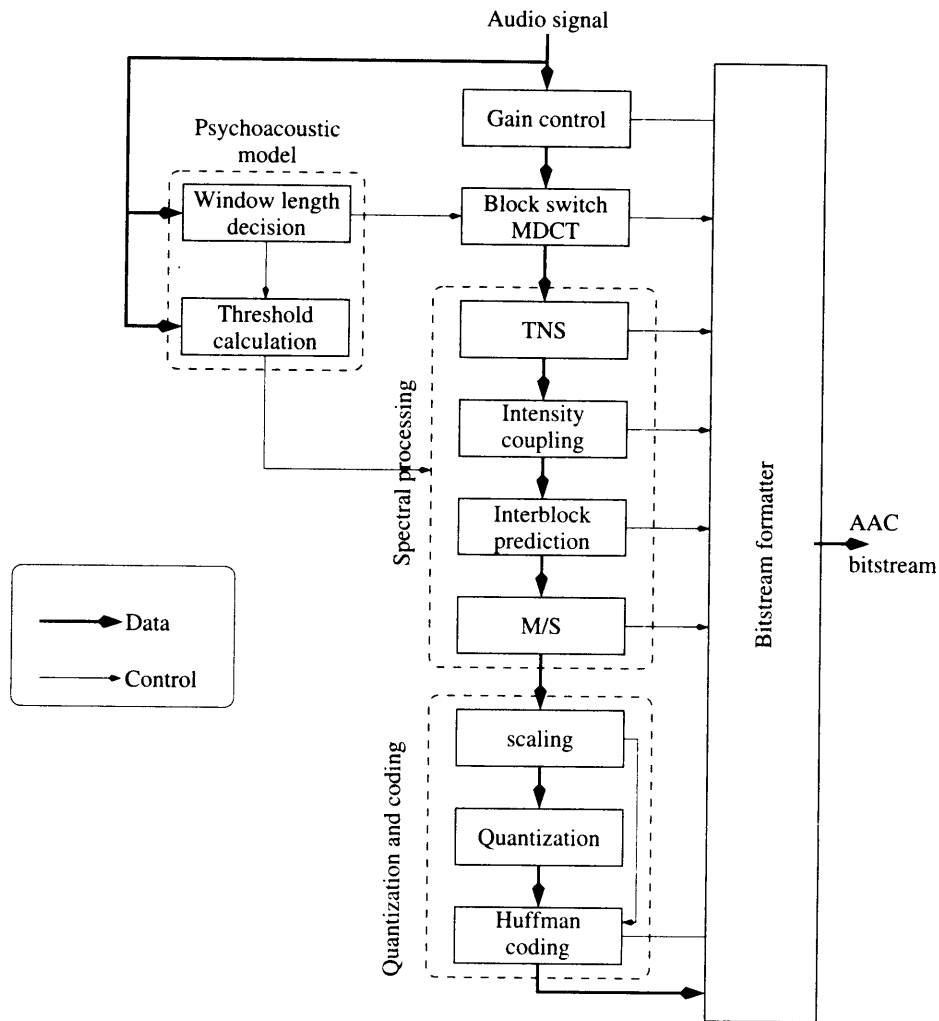


FIGURE 16.13 An MPEG-2 AAC encoder [213].

are divided into scalefactor bands in which the number of coefficients in the bands reflects the critical bandwidth. Each scalefactor band is assigned a single scalefactor. The exact division of the coefficients into scalefactor bands for the different windows and different sampling rates is specified in the standard [213].

Spectral Processing

In MPEG Layer III coding the compression gain is mainly achieved through the unequal distribution of energy in the different frequency bands, the use of the psychoacoustic model,

and Huffman coding. The unequal distribution of energy allows use of fewer bits for spectral bands with less energy. The psychoacoustic model is used to adjust the quantization step size in a way that masks the quantization noise. The Huffman coding allows further reductions in the bit rate. All these approaches are also used in the AAC algorithm. In addition, the algorithm makes use of prediction to reduce the dynamic range of the coefficients and thus allow further reduction in the bit rate.

Recall that prediction is generally useful only in stationary conditions. By their very nature, transients are almost impossible to predict. Therefore, generally speaking, predictive coding would not be considered for signals containing significant amounts of transients. However, music signals have exactly this characteristic. Although they may contain long periods of stationary signals, they also generally contain a significant amount of transient signals. The AAC algorithm makes clever use of the time frequency duality to handle this situation. The standard contains two kinds of predictors, an intrablock predictor, referred to as Temporal Noise Shaping (TNS), and an interblock predictor. The interblock predictor is used during stationary periods. During these periods it is reasonable to assume that the coefficients at a certain frequency do not change their value significantly from block to block. Making use of this characteristic, the AAC standard implements a set of parallel DPCM systems. There is one predictor for each coefficient up to a maximum number of coefficients. The maximum is different for different sampling frequencies. Each predictor is a backward adaptive two-tap predictor. This predictor is really useful only in stationary periods. Therefore, the psychoacoustic model monitors the input and determines when the output of the predictor is to be used. The decision is made on a scalefactor band by scalefactor band basis. Because notification of the decision that the predictors are being used has to be sent to the decoder, this would increase the rate by one bit for each scalefactor band. Therefore, once the preliminary decision to use the predicted value has been made, further calculations are made to check if the savings will be sufficient to offset this increase in rate. If the savings are determined to be sufficient, a *predictor_data_present* bit is set to 1 and one bit for each scalefactor band (called the *prediction_used* bit) is set to 1 or 0 depending on whether prediction was deemed effective for that scalefactor band. If not, the *predictor_data_present* bit is set to 0 and the *prediction_used* bits are not sent. Even when a predictor is disabled, the adaptive algorithm is continued so that the predictor coefficients can track the changing coefficients. However, because this is a streaming audio format it is necessary from time to time to reset the coefficients. Resetting is done periodically in a staged manner and also when a short frame is used.

When the audio input contains transients, the AAC algorithm uses the intraband predictor. Recall that narrow pulses in time correspond to wide bandwidths. The narrower a signal in time, the broader its Fourier transform will be. This means that when transients occur in the audio signal, the resulting MDCT output will contain a large number of correlated coefficients. Thus, unpredictability in time translates to a high level of predictability in terms of the frequency components. The AAC uses neighboring coefficients to perform prediction. A target set of coefficients is selected in the block. The standard suggests a range of 1.5 kHz to the uppermost scalefactor band as specified for different profiles and sampling rates. A set of linear predictive coefficients is obtained using any of the standard approaches, such as the Levinson-Durbin algorithm described in Chapter 15. The maximum order of the filter ranges from 12 to 20 depending on the profile. The process of obtaining the filter

coefficients also provides the expected prediction gain g_p . This expected prediction gain is compared against a threshold to determine if intrablock prediction is going to be used. The standard suggests a value of 1.4 for the threshold. The order of the filter is determined by the first PARCOR coefficient with a magnitude smaller than a threshold (suggested to be 0.1). The PARCOR coefficients corresponding to the predictor are quantized and coded for transfer to the decoder. The reconstructed LPC coefficients are then used for prediction. In the time domain predictive coders, one effect of linear prediction is the spectral shaping of the quantization noise. The effect of prediction in the frequency domain is the *temporal* shaping of the quantization noise, hence the name Temporal Noise Shaping. The shaping of the noise means that the noise will be higher during time periods when the signal amplitude is high and lower when the signal amplitude is low. This is especially useful in audio signals because of the masking properties of human hearing.

Quantization and Coding

The quantization and coding strategy used in AAC is similar to what is used in MPEG Layer III. Scalefactors are used to control the quantization noise as a part of an outer *distortion control loop*. The quantization step size is adjusted to accommodate a target bit rate in an inner *rate control loop*. The quantized coefficients are grouped into *sections*. The section boundaries have to coincide with scalefactor band boundaries. The quantized coefficients in each section are coded using the same Huffman codebook. The partitioning of the coefficients into sections is a dynamic process based on a greedy merge procedure. The procedure starts with the maximum number of sections. Sections are merged if the overall bit rate can be reduced by merging. Merging those sections will result in the maximum reduction in bit rate. This iterative procedure is continued until there is no further reduction in the bit rate.

Stereo Coding

The AAC scheme uses multiple approaches to stereo coding. Apart from independently coding the audio channels, the standard allows Mid/Side (M/S) coding and intensity stereo coding. Both stereo coding techniques can be used at the same time for different frequency ranges. Intensity coding makes use of the fact that at higher frequencies two channels can be represented by a single channel plus some directional information. The AAC standard suggests using this technique for scalefactor bands above 6 kHz. The M/S approach is used to reduce noise imaging. As described previously in the joint stereo approach, the two channels (L and R) are combined to generate sum and difference channels.

Profiles

The main profile of MPEG-2 AAC uses all the tools except for the gain control tool of Figure 16.13. The low complexity profile in addition to the gain control tool the interblock prediction tool is also dropped. In addition the maximum prediction order for intra-band prediction (TNS) for long windows is 12 for the low complexity profile as opposed to 20 for the main profile.

The Scalable Sampling Rate profile does not use the coupling and interband prediction tools. However this profile does use the gain control tool. In the scalable-sampling profile the MDCT block is preceded by a bank of four equal width 96 tap filters. The filter coefficients are provided in the standard. The use of this filterbank allows for a reduction in rate and decoder complexity. By ignoring one or more of the filterbank outputs the output bandwidth can be reduced. This reduction in bandwidth and sample rate also leads to a reduction in the decoder complexity. The gain control allows for the attenuation and amplification of different bands in order to reduce perceptual distortion.

16.4.2 MPEG-4 AAC

The MPEG-4 AAC adds a perceptual noise substitution (PNS) tool and substitutes a long term prediction (LTP) tool for the interband prediction tool in the spectral coding block. In the quantization and coding section the MPEG-4 AAC adds the options of Transform-Domain Weighted Interleave Vector Quantization (TwinVQ) and Bit Sliced Arithmetic Coding (BSAC).

Perceptual Noise Substitution (PNS)

There are portions of music that sound like noise. Although this may sound like a harsh (or realistic) subjective evaluation, that is not what is meant here. What is meant by noise here is a portion of audio where the MDCT coefficients are stationary without containing tonal components [214]. This kind of noise-like signal is the hardest to compress. However, at the same time it is very difficult to distinguish one noise-like signal from another. The MPEG-4 AAC makes use of this fact by not transmitting such noise-like scalefactor bands. Instead the decoder is alerted to this fact and the power of the noise-like coefficients in this band is sent. The decoder generates a noise-like sequence with the appropriate power and inserts it in place of the unsent coefficients.

Long Term Prediction

The interband prediction in MPEG-2 AAC is one of the more computationally expensive parts of the algorithm. MPEG-4 AAC replaces that with a cheaper long term prediction (LTP) module.

TwinVQ

The Transform-Domain Weighted Interleave Vector Quantization (TwinVQ) [215] option is suggested in the MPEG-4 AAC scheme for low bit rates. Developed at NTT in the early 1990s, the algorithm uses a two-stage process for flattening the MDCT coefficients. In the first stage, a linear predictive coding algorithm is used to obtain the LPC coefficients for the audio data corresponding to the MDCT coefficients. These coefficients are used to obtain the spectral envelope for the audio data. Dividing the MDCT coefficients with this spectral envelope results in some degree of “flattening” of the coefficients. The spectral envelope computed from the LPC coefficients reflects the gross features of the envelope

of the MDCT coefficients. However, it does not reflect any of the fine structure. This fine structure is predicted from the previous frame and provides further flattening of the MDCT coefficients. The flattened coefficients are interleaved and grouped into subvectors and quantized. The flattening process reduces the dynamic range of the coefficients, allowing them to be quantized using a smaller VQ codebook than would otherwise have been possible. The flattening process is reversed in the decoder as the LPC coefficients are transmitted to the decoder.

Bit Sliced Arithmetic Coding (BSAC)

In addition to the Huffman coding scheme of the MPEG-2 AAC scheme, the MPEG-4 AAC scheme also provides the option of using binary arithmetic coding. The binary arithmetic coding is performed on the bitplanes of the magnitudes of the quantized MDCT coefficients. By bitplane we mean the corresponding bit of each coefficient. Consider the sequence of 4-bit coefficients x_n : 5, 11, 8, 10, 3, 1. The most significant bitplane would consist of the MSBs of these numbers, 011100. The next bitplane would be 100000. The next bitplane is 010110. The least significant bitplane is 110011.

The coefficients are divided into *coding bands* of 32 coefficients each. One probability table is used to encode each coding band. Because we are dealing with binary data, the probability table is simply the number of zeros. If a coding band contains only zeros, this is indicated to the decoder by selecting the probability table 0. The sign bits associated with the nonzero coefficients are sent after the arithmetic code when the coefficient has a 1 for the first time.

The scalefactor information is also arithmetic coded. The maximum scalefactor is coded as an 8-bit integer. The differences between scalefactors are encoded using an arithmetic code. The first scalefactor is encoded using the difference between it and the maximum scalefactor.

16.5 Dolby AC3 (Dolby Digital)

Unlike the MPEG algorithms described in the previous section, the Dolby AC-3 method became a de facto standard. It was developed in response to the standardization activities of the *Grand Alliance*, which was developing a standard for HDTV in the United States. However, even before it was accepted as the recommendation for HDTV audio, Dolby-AC3 had already made its debut in the movie industry. It was first released in a few theaters during the showing of *Star Trek IV* in 1991 and was formally released with the movie *Batman Returns* in 1992. It was accepted by the *Grand Alliance* in October of 1993 and became an Advanced Television Systems Committee (ATSC) standard in 1995. Dolby AC-3 had the multichannel capability required by the movie industry along with the ability to downmix the channels to accommodate the varying capabilities of different applications. The 5.1 channels include right, center, left, left rear, and right rear, and a narrowband low-frequency effects channel (the 0.1 channel). The scheme supports downmixing the 5.1 channels to 4, 3, 2, or 1 channel. It is now the standard used for DVDs as well as for Direct Broadcast Satellites (DBS) and other applications.

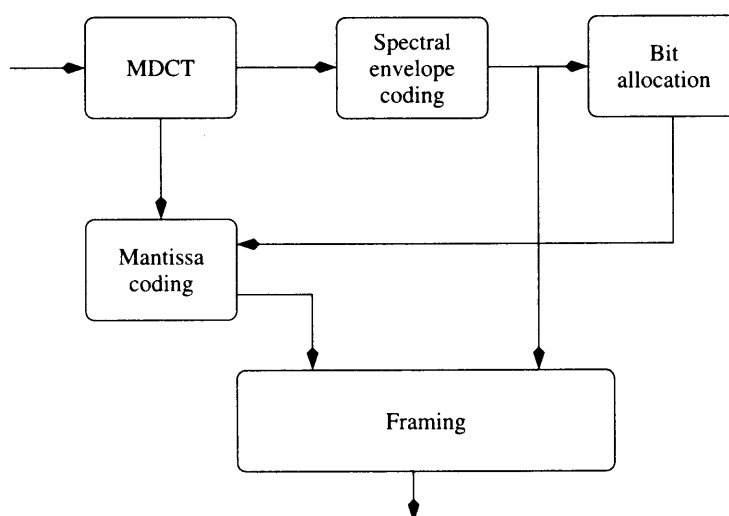


FIGURE 16.14 The Dolby AC3 algorithm.

A block diagram of the Dolby-AC3 algorithm is shown in Figure 16.14. Much of the Dolby-AC3 scheme is similar to what we have already described for the MPEG algorithms. As in the MPEG schemes, the Dolby-AC3 algorithm uses the modified DCT (MDCT) with 50% overlap for frequency decomposition. As in the case of MPEG, there are two different sizes of windows used. For the stationary portions of the audio a window of size 512 is used to get a 256 coefficient. A surge in the power of the high frequency coefficients is used to indicate the presence of a transient and the 512 window is replaced by two windows of size 256. The one place where the Dolby-AC3 algorithm differs significantly from the algorithm described is in the bit allocation.

16.5.1 Bit Allocation

The Dolby-AC3 scheme has a very interesting method for bit allocation. Like the MPEG schemes, it uses a psychoacoustic model that incorporates the hearing thresholds and the presence of noise and tone maskers. However, the input to the model is different. In the MPEG schemes the audio sequence being encoded is provided to the bit allocation procedure and the bit allocation is sent to the decoder as side information. In the Dolby-AC3 scheme the signal itself is not provided to the bit allocation procedure. Instead a crude representation of the spectral envelope is provided to both the decoder and the bit allocation procedure. As the decoder then possesses the information used by the encoder to generate the bit allocation, the allocation itself is not included in the transmitted bitstream.

The representation of the spectral envelope is obtained by representing the MDCT coefficients in binary exponential notation. The binary exponential notation of a number 110.101 is 0.110101×2^3 , where 110101 is called the mantissa and 3 is the exponent. Given a sequence of numbers, the exponents of the binary exponential representation provide

an estimate of the relative magnitude of the numbers. The Dolby-AC3 algorithm uses the exponents of the binary exponential representation of the MDCT coefficients as the representation of the spectral envelope. This encoding is sent to the bit allocation algorithm, which uses this information in conjunction with a psychoacoustic model to generate the number of bits to be used to quantize the mantissa of the binary exponential representation of the MDCT coefficients. To reduce the amount of information that needs to be sent to the decoder, the spectral envelope coding is not performed for every audio block. Depending on how stationary the audio is, the algorithm uses one of three strategies [194].

The D15 Method

When the audio is relatively stationary, the spectral envelope is coded once for every six audio blocks. Because a frame in Dolby-AC3 consists of six blocks, during each block we get a new spectral envelope and hence a new bit allocation. The spectral envelope is coded differentially. The first exponent is sent as is. The difference between exponents is encoded using one of five values $\{0, \pm 1, \pm 2\}$. Three differences are encoded using a 7-bit word. Note that three differences can take on 125 different combinations. Therefore, using 7 bits, which can represent 128 different values, is highly efficient.

The D25 and D45 Methods

If the audio is not stationary, the spectral envelope is sent more often. To keep the bit rate down, the Dolby-AC3 algorithm uses one of two strategies. In the D25 strategy, which is used for moderate spectral activity, every other coefficient is encoded. In the D45 strategy, used during transients, every fourth coefficient is encoded. These strategies make use of the fact that during a transient the fine structure of the spectral envelope is not that important, allowing for a more crude representation.

16.6 Other Standards

We have described a number of audio compression approaches that make use of the limitations of human audio perception. These are by no means the only ones. Competitors to Dolby Digital include Digital Theater Systems (DTS) and Sony Dynamic Digital Sound (SDDS). Both of these proprietary schemes use psychoacoustic modeling. The Adaptive TRansform Acoustic Coding (ATRAC) algorithm [216] was developed for the minidisc by Sony in the early 1990s, followed by enhancements in ATRAC3 and ATRAC3plus. As with the other schemes described in this chapter, the ATRAC approach uses MDCT for frequency decomposition, though the audio signal is first decomposed into three bands using a two-stage decomposition. As in the case of the other schemes, the ATRAC algorithm recommends the use of the limitations of human audio perception in order to discard information that is not perceptible.

Another algorithm that also uses MDCT and a psychoacoustic model is the open source encoder Vorbis. The Vorbis algorithm also uses vector quantization and Huffman coding to reduce the bit rate.

16.7 Summary

The audio coding algorithms described in this chapter take, in some sense, the opposite tack from the speech coding algorithms described in the previous chapter. Instead of focusing on the source of information, as is the case with the speech coding algorithm, the focus in the audio coding algorithm is on the sink, or user, of the information. By identifying the components of the source signal that are not perceptible, the algorithms reduce the amount of data that needs to be transmitted.

Further Reading

1. The book *Introduction to Digital Audio Coding and Standards* by M. Bosi and R.E. Goldberg [194] provides a detailed accounting of the standards described here as well as a comprehensive look at the process of constructing a psychoacoustic model.
2. *The MPEG Handbook*, by J. Watkinson [214], is an accessible source of information about aspects of audio coding as well as the MPEG algorithms.
3. An excellent tutorial on the MPEG algorithms is the appropriately named *A Tutorial on MPEG/Audio Compression*, by D. Pan [217].
4. A thorough review of audio coding can be found in *Perceptual Coding of Digital Audio*, by T. Painter and A. Spanias [218].
5. The website <http://www.tnt.uni-hannover.de/project/mpeg/audio/faq/> contains information about all the audio coding schemes described here as well as an overview of MPEG-7 audio.

Analysis/Synthesis and Analysis by Synthesis Schemes

17.1 Overview

Analysis/synthesis schemes rely on the availability of a parametric model of the source output generation. When such a model exists, the transmitter analyzes the source output and extracts the model parameters, which are transmitted to the receiver. The receiver uses the model along with the transmitted parameters to synthesize an approximation to the source output. The difference between this approach and the techniques we have looked at in previous chapters is that what is transmitted is not a direct representation of the samples of the source output; instead, the transmitter informs the receiver how to go about regenerating those outputs. For this approach to work, a good model for the source has to be available. Since good models for speech production exist, this approach has been widely used for the low-rate coding of speech. We describe several different analysis/synthesis techniques for speech compression. In recent years the fractal approach to image compression has been gaining in popularity. Because this approach is also one in which the receiver regenerates the source output using “instructions” from the transmitter, we describe it in this chapter.

17.2 Introduction

In the previous chapters we have presented a number of lossy compression schemes that provide an estimate of each source output value to the receiver. Historically, an earlier approach towards lossy compression is to model the source output and send the model parameters to the source instead of the estimates of the source output. The receiver tries to synthesize the source output based on the received model parameters.

Consider an image transmission system that works as follows. At the transmitter, we have a person who examines the image to be transmitted and comes up with a description of the image. At the receiver, we have another person who then proceeds to create that image. For example, suppose the image we wish to transmit is a picture of a field of sunflowers. Instead of trying to send the picture, we simply send the words “field of sunflowers.” The person at the receiver paints a picture of a field of sunflowers on a piece of paper and gives it to the user. Thus, an image of an object is transmitted from the transmitter to the receiver in a highly compressed form. This approach towards compression should be familiar to listeners of sports broadcasts on radio. It requires that both transmitter and receiver work with the same model. In terms of sports broadcasting, this means that the viewer has a mental picture of the sports arena, and both the broadcaster and listener attach the same meaning to the same terminology.

This approach works for sports broadcasting because the source being modeled functions under very restrictive rules. In a basketball game, when the referee calls a dribbling foul, listeners generally don't picture a drooling chicken. If the source violates the rules, the reconstruction would suffer. If the basketball players suddenly decided to put on a ballet performance, the transmitter (sportscaster) would be hard pressed to represent the scene accurately to the receiver. Therefore, it seems that this approach to compression can only be used for artificial activities that function according to man-made rules. Of the sources that we are interested in, only text fits this description, and the rules that govern the generation of text are complex and differ widely from language to language.

Fortunately, while natural sources may not follow man-made rules, they are subject to the laws of physics, which can prove to be quite restrictive. This is particularly true of speech. No matter what language is being spoken, the speech is generated using machinery that is not very different from person to person. Moreover, this machinery has to obey certain physical laws that substantially limit the behavior of outputs. Therefore, speech can be analyzed in terms of a model, and the model parameters can be extracted and transmitted to the receiver. At the receiver the speech can be synthesized using the model. This analysis/synthesis approach was first employed by Homer Dudley at Bell Laboratories, who developed what is known as the channel vocoder (described in the next section). Actually, the synthesis portion had been attempted even earlier by Kempelen Farkas Lovag (1734–1804). He developed a “speaking machine” in which the vocal tract was modeled by a flexible tube whose shape could be modified by an operator. Sound was produced by forcing air through this tube using bellows [219].

Unlike speech, images are generated in a variety of different ways; therefore, the analysis/synthesis approach does not seem very useful for image or video compression. However, if we restrict the class of images to “talking heads” of the type we would encounter in a video-conferencing situation, we might be able to satisfy the conditions required for this approach. When we talk, our facial gestures are restricted by the way our faces are constructed and by the physics of motion. This realization has led to the new field of model-based video coding (see Chapter 16).

A totally different approach to image compression based on the properties of self-similarity is the *fractal coding* approach. While this approach does not explicitly depend on some physical limitations, it fits in with the techniques described in this chapter; that is, what is stored or transmitted is not the samples of the source output, but a method for synthesizing the output. We will study this approach in Section 17.5.1.

17.3 Speech Compression

A very simplified model of speech synthesis is shown in Figure 17.1. As we described in Chapter 7, speech is produced by forcing air first through an elastic opening, the vocal cords, and then through the laryngeal, oral, nasal, and pharynx passages, and finally through the mouth and the nasal cavity. Everything past the vocal cords is generally referred to as the vocal tract. The first action generates the sound, which is then modulated into speech as it traverses through the vocal tract.

In Figure 17.1, the excitation source corresponds to the sound generation, and the vocal tract filter models the vocal tract. As we mentioned in Chapter 7, there are several different sound inputs that can be generated by different conformations of the vocal cords and the associated cartilages.

Therefore, in order to generate a specific fragment of speech, we have to generate a sequence of sound inputs or excitation signals and the corresponding sequence of appropriate vocal tract approximations.

At the transmitter, the speech is divided into segments. Each segment is analyzed to determine an excitation signal and the parameters of the vocal tract filter. In some of the schemes, a model for the excitation signal is transmitted to the receiver. The excitation signal is then synthesized at the receiver and used to drive the vocal tract filter. In other schemes, the excitation signal itself is obtained using an analysis-by-synthesis approach. This signal is then used by the vocal tract filter to generate the speech signal.

Over the years many different analysis/synthesis speech compression schemes have been developed, and substantial research into the development of new approaches and the improvement of existing schemes continues. Given the large amount of information, we can only sample some of the more popular approaches in this chapter. See [220, 221, 222] for more detailed coverage and pointers to the vast literature on the subject.

The approaches we will describe in this chapter include *channel vocoders*, which are of special historical interest; the *linear predictive coder*, which is the U.S. Government standard at the rate of 2.4 kbps; *code excited linear prediction (CELP)* based schemes; *sinusoidal coders*, which provide excellent performance at rates of 4.8 kbps and higher and are also a part of several national and international standards; and *mixed excitation linear prediction*, which is to be the new 2.4 kbps federal standard speech coder. In our description of these approaches, we will use the various national and international standards as examples.

17.3.1 The Channel Vocoder

In the channel vocoder [223], each segment of input speech is analyzed using a bank of band-pass filters called the *analysis filters*. The energy at the output of each filter is estimated at fixed intervals and transmitted to the receiver. In a digital implementation, the energy

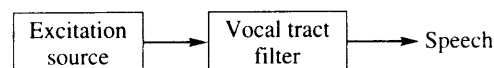


FIGURE 17.1 A model for speech synthesis.

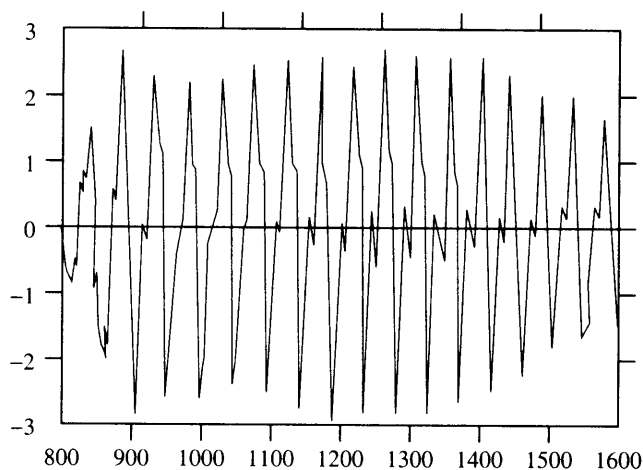


FIGURE 17.2 The sound /e/ in *test*.

estimate may be the average squared value of the filter output. In analog implementations, this is the sampled output of an envelope detector. Generally, an estimate is generated 50 times every second. Along with the estimate of the filter output, a decision is made as to whether the speech in that segment is voiced, as in the case of the sounds /a/ /e/ /o/, or unvoiced, as in the case for the sounds /s/ /f/. Voiced sounds tend to have a pseudoperiodic structure, as seen in Figure 17.2, which is a plot of the /e/ part of a male voice saying the word *test*. The period of the fundamental harmonic is called the *pitch* period. The transmitter also forms an estimate of the pitch period, which is transmitted to the receiver.

Unvoiced sounds tend to have a noiselike structure, as seen in Figure 17.3, which is the /s/ sound in the word *test*.

At the receiver, the vocal tract filter is implemented by a bank of band-pass filters. The bank of filters at the receiver, known as the *synthesis filters*, is identical to the bank of analysis filters. Based on whether the speech segment was deemed to be voiced or unvoiced, either a pseudonoise source or a periodic pulse generator is used as the input to the synthesis filter bank. The period of the pulse input is determined by the pitch estimate obtained for the segment being synthesized at the transmitter. The input is scaled by the energy estimate at the output of the analysis filters. A block diagram of the synthesis portion of the channel vocoder is shown in Figure 17.4.

Since the introduction of the channel vocoder, a number of variations have been developed. The channel vocoder matches the frequency profile of the input speech. There is no attempt to reproduce the speech samples per se. However, not all frequency components of speech are equally important. In fact, as the vocal tract is a tube of nonuniform cross section, it resonates at a number of different frequencies. These frequencies are known as *formants* [105]. The formant values change with different sounds; however, we can identify ranges in which they occur. For example, the first formant occurs in the range 200–800 Hz for a male speaker, and in the range 250–1000 Hz for a female speaker. The importance of these

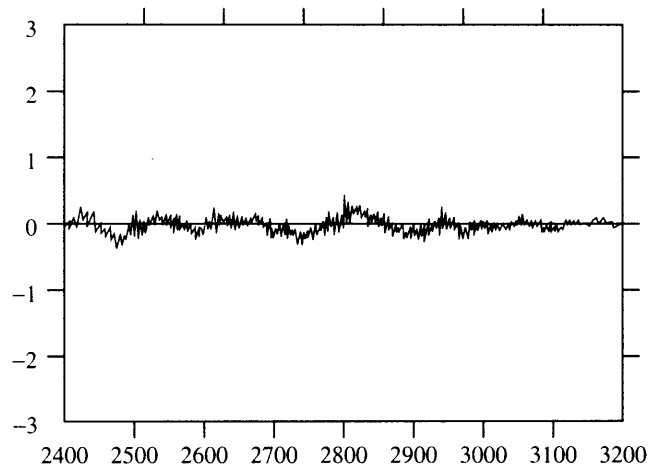


FIGURE 17.3 The sound /s/ in test.

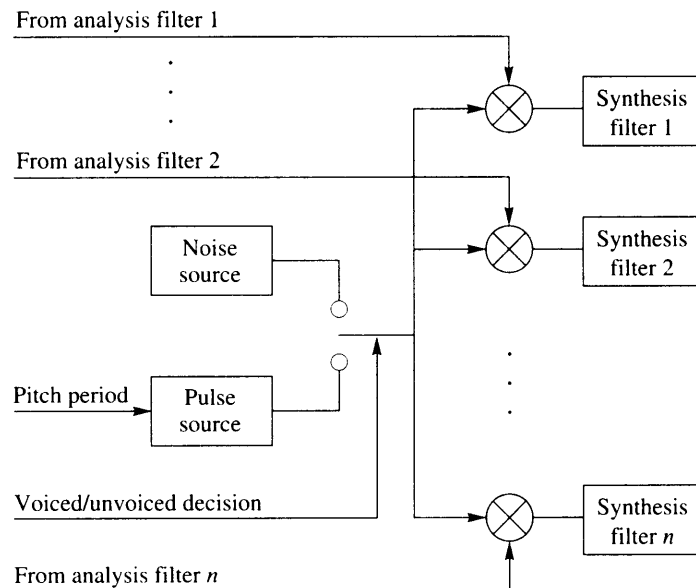


FIGURE 17.4 The channel vocoder receiver.

formants has led to the development of *formant vocoders*, which transmit an estimate of the formant values (usually four formants are considered sufficient) and an estimate of the bandwidth of each formant. At the receiver the excitation signal is passed through tunable filters that are tuned to the formant frequency and bandwidth.

An important development in the history of the vocoders was an understanding of the importance of the excitation signal. Schemes that require the synthesis of the excitation signal at the receiver spend a considerable amount of computational resources to obtain accurate voicing information and accurate pitch periods. This expense can be avoided through the use of voice excitation. In the voice-excited channel vocoder, the voice is first filtered using a narrow-band low-pass filter. The output of the low-pass filter is sampled and transmitted to the receiver. At the receiver, this low-pass signal is passed through a nonlinearity to generate higher-order harmonics that, together with the low-pass signal, are used as the excitation signal. Voice excitation removes the problem of pitch extraction. It also removes the necessity for declaring every segment either voiced or unvoiced. As there are usually quite a few segments that are neither totally voiced or unvoiced, this can result in a substantial increase in quality. Unfortunately, the increase in quality is reflected in the high cost of transmitting the low-pass filtered speech signal.

The channel vocoder, although historically the first approach to analysis/synthesis—indeed the first approach to speech compression—is not as popular as some of the other schemes described here. However, all the different schemes can be viewed as descendants of the channel vocoder.

17.3.2 The Linear Predictive Coder (Government Standard LPC-10)

Of the many descendants of the channel vocoder, the most well-known is the linear predictive coder (LPC). Instead of the vocal tract being modeled by a bank of filters, in the linear predictive coder the vocal tract is modeled as a single linear filter whose output y_n is related to the input ϵ_n by

$$y_n = \sum_{i=1}^M b_i y_{n-i} + G\epsilon_n \quad (17.1)$$

where G is called the gain of the filter. As in the case of the channel vocoder, the input to the vocal tract filter is either the output of a random noise generator or a periodic pulse generator. A block diagram of the LPC receiver is shown in Figure 17.5.

At the transmitter, a segment of speech is analyzed. The parameters obtained include a decision as to whether the segment of speech is voiced or unvoiced, the pitch period if the segment is declared voiced, and the parameters of the vocal tract filter. In this section, we will take a somewhat detailed look at the various components that make up the linear predictive coder. As an example, we will use the specifications for the 2.4-kbit U.S. Government Standard LPC-10.

The input speech is generally sampled at 8000 samples per second. In the LPC-10 standard, the speech is broken into 180 sample segments, corresponding to 22.5 milliseconds of speech per segment.

The Voiced/Unvoiced Decision

If we compare Figures 17.2 and 17.3, we can see there are two major differences. Notice that the samples of the voiced speech have larger amplitude; that is, there is more energy in

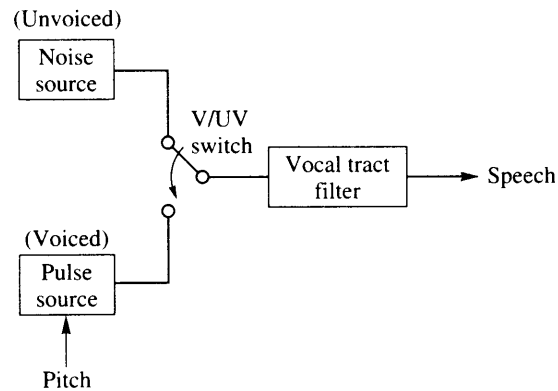


FIGURE 17.5 A model for speech synthesis.

the voiced speech. Also, the unvoiced speech contains higher frequencies. As both speech segments have average values close to zero, this means that the unvoiced speech waveform crosses the $x = 0$ line more often than the voiced speech sample. Therefore, we can get a fairly good idea about whether the speech is voiced or unvoiced based on the energy in the segment relative to background noise and the number of zero crossings within a specified window. In the LPC-10 algorithm, the speech segment is first low-pass filtered using a filter with a bandwidth of 1 kHz. The energy at the output relative to the background noise is used to obtain a tentative decision about whether the signal in the segment should be declared voiced or unvoiced. The estimate of the background noise is basically the energy in the unvoiced speech segments. This tentative decision is further refined by counting the number of zero crossings and checking the magnitude of the coefficients of the vocal tract filter. We will talk more about this latter point later in this section. Finally, it can be perceptually annoying to have a single voiced frame sandwiched between unvoiced frames. The voicing decision of the neighboring frames is considered in order to prevent this from happening.

Estimating the Pitch Period

Estimating the pitch period is one of the most computationally intensive steps of the analysis process. Over the years a number of different algorithms for pitch extraction have been developed. In Figure 17.2, it would appear that obtaining a good estimate of the pitch should be relatively easy. However, we should keep in mind that the segment shown in Figure 17.2 consists of 800 samples, which is considerably more than the samples available to the analysis algorithm. Furthermore, the segment shown here is noise-free and consists entirely of a voiced input. For a machine to extract the pitch from a short noisy segment, which may contain both voiced and unvoiced components, can be a difficult undertaking.

Several algorithms make use of the fact that the autocorrelation of a periodic function $R_{xx}(k)$ will have a maximum when k is equal to the pitch period. Coupled with the fact that the estimation of the autocorrelation function generally leads to a smoothing out of the noise, this makes the autocorrelation function a useful tool for obtaining the pitch period.

Unfortunately, there are also some problems with the use of the autocorrelation. Voiced speech is not exactly periodic, which makes the maximum lower than we would expect from a periodic signal. Generally, a maximum is detected by checking the autocorrelation value against a threshold; if the value is greater than the threshold, a maximum is declared to have occurred. When there is uncertainty about the magnitude of the maximum value, it is difficult to select a value for the threshold. Another problem occurs because of the interference due to other resonances in the vocal tract. There are a number of algorithms that resolve these problems in different ways. (see [105, 104] for details).

In this section, we will describe a closely related technique, employed in the LPC-10 algorithm, that uses the average magnitude difference function (AMDF). The AMDF is defined as

$$AMDF(P) = \frac{1}{N} \sum_{i=k_0+1}^{k_0+N} |y_i - y_{i-P}| \quad (17.2)$$

If a sequence $\{y_n\}$ is periodic with period P_0 , samples that are P_0 apart in the $\{y_n\}$ sequence will have values close to each other, and therefore the AMDF will have a minimum at P_0 . If we evaluate this function using the /e/ and /s/ sequences, we get the results shown in Figures 17.6 and 17.7. Notice that not only do we have a minimum when P equals the pitch period, but any spurious minimums we may obtain in the unvoiced segments are very shallow; that is, the difference between the minimum and average values is quite small. Therefore, the AMDF can serve a dual purpose: it can be used to identify the pitch period as well as the voicing condition.

The job of pitch extraction is simplified by the fact that the pitch period in humans tends to fall in a limited range. Thus, we do not have to evaluate the AMDF for all possible values of P . For example, the LPC-10 algorithm assumes that the pitch period is between 2.5 and

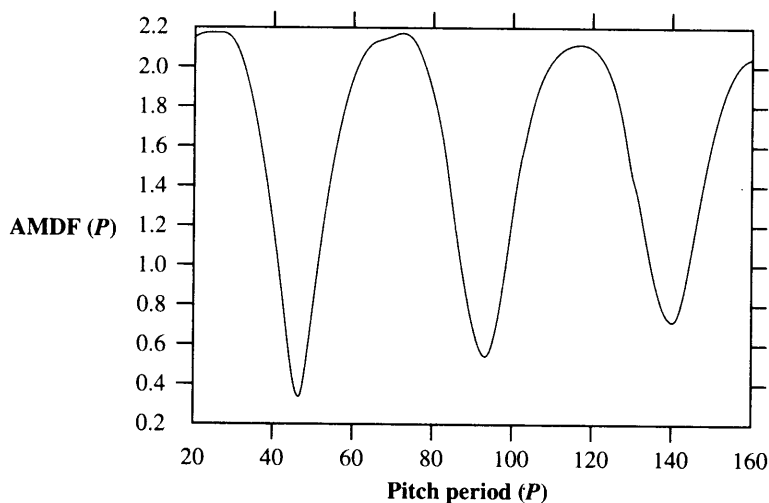


FIGURE 17.6 AMDF function for the sound /e/ in test.

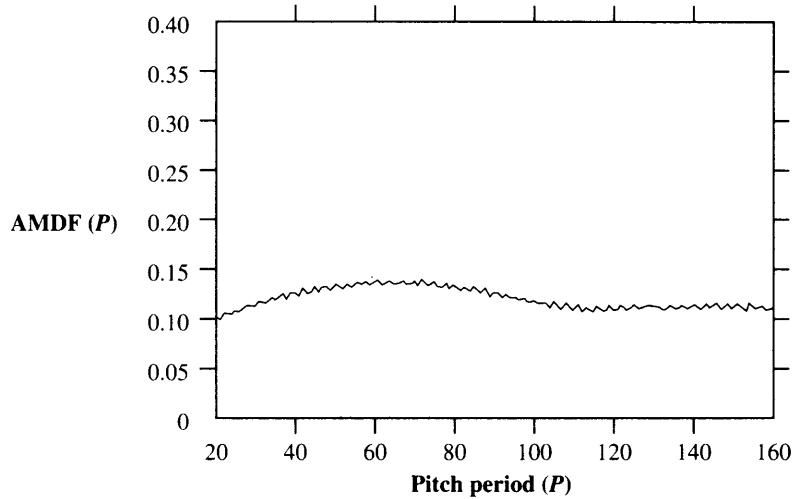


FIGURE 17.7 AMDF function for the sound /s/ in fest.

19.5 milliseconds. Assuming a sampling rate of 8000 samples a second, this means that P is between 20 and 160.

Obtaining the Vocal Tract Filter

In linear predictive coding, the vocal tract is modeled by a linear filter with the input-output relationship shown in Equation (17.1). At the transmitter, during the analysis phase we obtain the filter coefficients that best match the segment being analyzed in a mean squared error sense. That is, if $\{y_n\}$ are the speech samples in that particular segment, then we want to choose $\{a_i\}$ to minimize the average value of e_n^2 where

$$e_n^2 = \left(y_n - \sum_{i=1}^M a_i y_{n-i} - G\epsilon_n \right)^2. \quad (17.3)$$

If we take the derivative of the expected value of e_n^2 with respect to the coefficients $\{a_j\}$, we get a set of M equations:

$$\frac{\delta}{\delta a_j} E \left[\left(y_n - \sum_{i=1}^M a_i y_{n-i} - G\epsilon_n \right)^2 \right] = 0 \quad (17.4)$$

$$\Rightarrow -2E \left[\left(y_n - \sum_{i=1}^M a_i y_{n-i} - G\epsilon_n \right) y_{n-j} \right] = 0 \quad (17.5)$$

$$\Rightarrow \sum_{i=1}^M a_i E [y_{n-i} y_{n-j}] = E [y_n y_{n-j}] \quad (17.6)$$

where in the last step we have made use of the fact that $E[\epsilon_n y_{n-j}]$ is zero for $j \neq 0$. In order to solve (17.6) for the filter coefficients, we need to be able to estimate $E[y_{n-i} y_{n-j}]$. There are two different approaches for estimating these values, called the *autocovariance* approach and the *autocorrelation* approach, each leading to a different algorithm. In the autocorrelation approach, we assume that the $\{y_n\}$ sequence is stationary and therefore

$$E[y_{n-1} y_{n-j}] = R_{yy}(|i-j|) \quad (17.7)$$

Furthermore, we assume that the $\{y_n\}$ sequence is zero outside the segment for which we are calculating the filter parameters. Therefore, the autocorrelation function is estimated as

$$R_{yy}(k) = \sum_{n=n_0+1+k}^{n_0+N} y_n y_{n-k} \quad (17.8)$$

and the M equations of the form of (17.6) can be written in matrix form as

$$\mathbf{R}A = P \quad (17.9)$$

where

$$\mathbf{R} = \begin{bmatrix} R_{yy}(0) & R_{yy}(1) & R_{yy}(2) & \cdots & R_{yy}(M-1) \\ R_{yy}(1) & R_{yy}(0) & R_{yy}(1) & \cdots & R_{yy}(M-2) \\ R_{yy}(2) & R_{yy}(1) & R_{yy}(0) & \cdots & R_{yy}(M-3) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ R_{yy}(M-1) & R_{yy}(M-2) & R_{yy}(M-3) & \cdots & R_{yy}(0) \end{bmatrix} \quad (17.10)$$

$$A = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_M \end{bmatrix} \quad (17.11)$$

and

$$P = \begin{bmatrix} R_{yy}(1) \\ R_{yy}(2) \\ R_{yy}(3) \\ \vdots \\ R_{yy}(M) \end{bmatrix} \quad (17.12)$$

This matrix equation can be solved directly to find the filter coefficients

$$A = \mathbf{R}^{-1}P. \quad (17.13)$$

However, the special form of the matrix \mathbf{R} obviates the need for computing \mathbf{R}^{-1} . Note that not only is \mathbf{R} symmetric, but also each diagonal of \mathbf{R} consists of the same element. For example, the main diagonal contains only the element $R_{yy}(0)$, while the diagonals above and below the main diagonal contain only the element $R_{yy}(1)$. This special type of matrix is called a *Toeplitz matrix*, and there are a number of efficient algorithms available for the inversion of Toeplitz matrices [224]. Because \mathbf{R} is Toeplitz, we can obtain a recursive solution to (17.9) that is computationally very efficient and that has an added attractive feature from the point of view of compression. This algorithm is known as the Levinson-Durbin algorithm [225, 226]. We describe the algorithm without derivation. For details of the derivation, see [227, 105].

In order to compute the filter coefficients of an M th-order filter, the Levinson-Durbin algorithm requires the computation of all filters of order less than M . Furthermore, during the computation of the filter coefficients, the algorithm generates a set of constants k_i known as the *reflection* coefficients, or *partial correlation* (PARCOR) coefficients. In the algorithm description below, we denote the order of the filter using superscripts. Thus, the coefficients of the fifth-order filter would be denoted by $\{a_i^{(5)}\}$. The algorithm also requires the computation of the estimate of the average error $E[e_n^2]$. We will denote the average error using an m th-order filter by E_m . The algorithm proceeds as follows:

1. Set $E_0 = R_{yy}(0)$, $i = 0$.
2. Increment i by one.
3. Calculate $k_i = \left(\sum_{j=1}^{i-1} a_j^{(i-1)} R_{yy}(i-j+1) - R_{yy}(i) \right) / E_{i-1}$.
4. Set $a_i^{(i)} = k_i$.
5. Calculate $a_j^{(i)} = a_j^{(i-1)} + k_i a_{i-j}^{(i-1)}$ for $j = 1, 2, \dots, i-1$.
6. Calculate $E_i = (1 - k_i^2) E_{i-1}$.
7. If $i < M$, go to step 2.

In order to get an effective reconstruction of the voiced segment, the order of the vocal tract filter needs to be sufficiently high. Generally, the order of the filter is 10 or more. Because the filter is an IIR filter, error in the coefficients can lead to instability, especially for the high orders necessary in linear predictive coding. As the filter coefficients are to be transmitted to the receiver, they need to be quantized. This means that quantization error is introduced into the value of the coefficients, and that can lead to instability.

This problem can be avoided by noticing that if we know the PARCOR coefficients, we can obtain the filter coefficients from them. Furthermore, PARCOR coefficients have the property that as long as the magnitudes of the coefficients are less than one, the filter obtained from them is guaranteed to be stable. Therefore, instead of quantizing the coefficients $\{a_i\}$ and transmitting them, the transmitter quantizes and transmits the coefficients $\{k_i\}$. As long as we make sure that all the reconstruction values for the quantizer have magnitudes less than one, it is possible to use relatively high-order filters in the analysis/synthesis schemes.

The assumption of stationarity that was used to obtain (17.6) is not really valid for speech signals. If we discard this assumption, the equations to obtain the filter coefficients change. The term $E[y_{n-i}y_{n-j}]$ is now a function of both i and j . Defining

$$c_{ij} = E[y_{n-i}y_{n-j}] \quad (17.14)$$

we get the equation

$$CA = S \quad (17.15)$$

where

$$\mathbf{C} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & \cdots & c_{1M} \\ c_{21} & c_{22} & c_{23} & \cdots & c_{2M} \\ \vdots & \vdots & \vdots & & \vdots \\ c_{M1} & c_{M2} & c_{M3} & \cdots & c_{MM} \end{bmatrix} \quad (17.16)$$

and

$$S = \begin{bmatrix} c_{10} \\ c_{20} \\ c_{30} \\ \vdots \\ c_{M0} \end{bmatrix} \quad (17.17)$$

The elements c_{ij} are estimated as

$$c_{ij} = \sum_{n=n_0+1}^{n_0+N} y_{n-i}y_{n-j}. \quad (17.18)$$

Notice that we no longer assume that the values of y_n outside of the segment under consideration are zero. This means that in calculating the \mathbf{C} matrix for a particular segment, we use samples from previous segments. This method of computing the filter coefficients is called the *covariance method*.

The \mathbf{C} matrix is symmetric but no longer Toeplitz, so we can't use the Levinson-Durbin recursion to solve for the filter coefficients. The equations are generally solved using a technique called the *Cholesky decomposition*. We will not describe the solution technique here. (You can find it in most texts on numerical techniques; an especially good source is [178].) For an in-depth study of the relationship between the Cholesky decomposition and the reflection coefficients, see [228].

The LPC-10 algorithm uses the covariance method to obtain the reflection coefficients. It also uses the PARCOR coefficients to update the voicing decision. In general, for voiced signals the first two PARCOR coefficients have values close to one. Therefore, if both the first two PARCOR coefficients have very small values, the algorithm sets the voicing decision to unvoiced.

Transmitting the Parameters

Once the various parameters have been obtained, they need to be coded and transmitted to the receiver. There are a variety of ways this can be done. Let us look at how the LPC-10 algorithm handles this task.

The parameters that need to be transmitted include the voicing decision, the pitch period, and the vocal tract filter parameters. One bit suffices to transmit the voicing information. The pitch is quantized to 1 of 60 different values using a log-companded quantizer. The LPC-10 algorithm uses a 10th-order filter for voiced speech and a 4th-order filter for unvoiced speech. Thus, we have to send 11 values (10 reflection coefficients and the gain) for voiced speech and 5 for unvoiced speech.

The vocal tract filter is especially sensitive to errors in reflection coefficients that have magnitudes close to one. As the first few coefficients are most likely to have values close to one, the LPC-10 algorithm specifies the use of nonuniform quantization for k_1 and k_2 . The nonuniform quantization is implemented by first generating the coefficients

$$g_i = \frac{1 + k_i}{1 - k_i} \quad (17.19)$$

which are then quantized using a 5-bit uniform quantizer. The coefficients k_3 and k_4 are both quantized using a 5-bit uniform quantizer. In the voiced segments, coefficients k_5 through k_8 are quantized using a 4-bit uniform quantizer, k_9 is quantized using a 3-bit uniform quantizer, and k_{10} is quantized using a 2-bit uniform quantizer. In the unvoiced segments, the 21 bits used to quantize k_5 through k_{10} in the voiced segments are used for error protection.

The gain G is obtained by finding the root mean squared (rms) value of the segment and quantized using 5-bit log-companded quantization. Including an additional bit for synchronization, we end up with a total of 54 bits per frame. Multiplying this by the total number of frames per second gives us the target rate of 2400 bits per second.

Synthesis

At the receiver, the voiced frames are generated by exciting the received vocal tract filter by a locally stored waveform. This waveform is 40 samples long. It is truncated or padded with zeros depending on the pitch period. If the frame is unvoiced, the vocal tract is excited by a pseudorandom number generator.

The LPC-10 coder provides intelligible reproduction at 2.4 kbits. The use of only two kinds of excitation signals gives an artificial quality to the voice. This approach also suffers when used in noisy environments. The encoder can be fooled into declaring segments of speech unvoiced because of background noise. When this happens, the speech information gets lost.

17.3.3 Code Excited Linear Prediction (CELP)

As we mentioned earlier, one of the most important factors in generating natural-sounding speech is the excitation signal. As the human ear is especially sensitive to pitch errors, a great deal of effort has been devoted to the development of accurate pitch detection algorithms.

However, no matter how accurate the pitch is in a system using the LPC vocal tract filter, the use of a periodic pulse excitation that consists of a single pulse per pitch period leads to a “buzzy twang” [229]. In 1982, Atal and Remde [230] introduced the idea of multipulse linear predictive coding (MP-LPC), in which several pulses were used during each segment. The spacing of these pulses is determined by evaluating a number of different patterns from a codebook of patterns.

A codebook of excitation patterns is constructed. Each entry in this codebook is an excitation sequence that consists of a few nonzero values separated by zeros. Given a segment from the speech sequence to be encoded, the encoder obtains the vocal tract filter using the LPC analysis described previously. The encoder then excites the vocal tract filter with the entries of the codebook. The difference between the original speech segment and the synthesized speech is fed to a perceptual weighting filter, which weights the error using a perceptual weighting criterion. The codebook entry that generates the minimum average weighted error is declared to be the best match. The index of the best-match entry is sent to the receiver along with the parameters for the vocal tract filter.

This approach was improved upon by Atal and Schroeder in 1984 with the introduction of the system that is commonly known as *code excited linear prediction* (CELP). In CELP, instead of having a codebook of pulse patterns, we allow a variety of excitation signals. For each segment the encoder finds the excitation vector that generates synthesized speech that best matches the speech segment being encoded. This approach is closer in a strict sense to a waveform coding technique such as DPCM than to the analysis/synthesis schemes. However, as the ideas behind CELP are similar to those behind LPC, we included CELP in this chapter. The main components of the CELP coder include the LPC analysis, the excitation codebook, and the perceptual weighting filter. Each component of the CELP coder has been investigated in great detail by a large number of researchers. For a survey of some of the results, see [220]. In the rest of the section, we give two examples of very different kinds of CELP coders. The first algorithm is the U.S. Government Standard 1016, a 4.8 kbps coder; the other is the CCITT (now ITU-T) G.728 standard, a low-delay 16 kbps coder.

Besides CELP, the MP-LPC algorithm had another descendant that has become a standard. In 1986, Kroon, Deprettere, and Sluyter [231] developed a modification of the MP-LPC algorithm. Instead of using excitation vectors in which the nonzero values are separated by an arbitrary number of zero values, they forced the nonzero values to occur at regularly spaced intervals. Furthermore, they allowed the nonzero values to take on a number of different values. They called this scheme *regular pulse excitation* (RPE) coding. A variation of RPE, called *regular pulse excitation with long-term prediction* (RPE-LTP) [232], was adopted as a standard for digital cellular telephony by the Group Speciale Mobile (GSM) subcommittee of the European Telecommunications Standards Institute at the rate of 13 kbps.

Federal Standard 1016

The vocal tract filter used by the CELP coder in FS 1016 is given by

$$y_n = \sum_{i=1}^{10} b_i y_{n-i} + \beta y_{n-p} + G\epsilon_n \quad (17.20)$$

where P is the pitch period and the term βy_{n-p} is the contribution due to the pitch periodicity. The input speech is sampled at 8000 samples per second and divided into 30-millisecond frames containing 240 samples. Each frame is divided into four subframes of length 7.5 milliseconds [233]. The coefficients $\{b_i\}$ for the 10th-order short-term filter are obtained using the autocorrelation method.

The pitch period P is calculated once every subframe. In order to reduce the computational load, the pitch value is assumed to lie between 20 and 147 every odd subframe. In every even subframe, the pitch value is assumed to lie within 32 samples of the pitch value in the previous frame.

The FS 1016 algorithm uses two codebooks [234], a stochastic codebook and an adaptive codebook. An excitation sequence is generated for each subframe by adding one scaled element from the stochastic codebook and one scaled element from the adaptive codebook. The scale factors and indices are selected to minimize the perceptual error between the input and synthesized speech.

The stochastic codebook contains 512 entries. These entries are generated using a Gaussian random number generator, the output of which is quantized to -1 , 0 , or 1 . If the input is less than -1.2 , it is quantized to -1 ; if it is greater than 1.2 , it is quantized to 1 ; and if it lies between -1.2 and 1.2 , it is quantized to 0 . The codebook entries are adjusted so that each entry differs from the preceding entry in only two places. This structure helps reduce the search complexity.

The adaptive codebook consists of the excitation vectors from the previous frame. Each time a new excitation vector is obtained, it is added to the codebook. In this manner, the codebook adapts to local statistics.

The FS 1016 coder has been shown to provide excellent reproductions in both quiet and noisy environments at rates of 4.8 kbps and above [234]. Because of the richness of the excitation signals, the reproduction does not suffer from the problem of sounding artificial. The lack of a voicing decision makes it more robust to background noise. The quality of the reproduction of this coder at 4.8 kbps has been shown to be equivalent to a delta modulator operating at 32 kbps [234]. The price for this quality is much higher complexity and a much longer coding delay. We will address this last point in the next section.

CCITT G.728 Speech Standard

By their nature, the schemes described in this chapter have some coding delay built into them. By "coding delay," we mean the time between when a speech sample is encoded to when it is decoded if the encoder and decoder were connected back-to-back (i.e., there were no transmission delays). In the schemes we have studied, a segment of speech is first stored in a buffer. We do not start extracting the various parameters until a complete segment of speech is available to us. Once the segment is completely available, it is processed. If the processing is real time, this means another segment's worth of delay. Finally, once the parameters have been obtained, coded, and transmitted, the receiver has to wait until at least a significant part of the information is available before it can start decoding the first sample. Therefore, if a segment contains 20 milliseconds' worth of data, the coding delay would be approximately somewhere between 40 to 60 milliseconds. This kind of delay may be acceptable for some applications; however, there are other applications where such long

delays are not acceptable. For example, in some situations there are several intermediate tandem connections between the initial transmitter and the final receiver. In such situations, the total delay would be a multiple of the coding delay of a single connection. The size of the delay would depend on the number of tandem connections and could rapidly become quite large.

For such applications, CCITT approved recommendation G.728, a CELP coder with a coder delay of 2 milliseconds operating at 16 kbps. As the input speech is sampled at 8000 samples per second, this rate corresponds to an average rate of 2 bits per sample.

In order to lower the coding delay, the size of each segment has to be reduced significantly because the coding delay will be some multiple of the size of the segment. The G.728 recommendation uses a segment size of five samples. With five samples and a rate of 2 bits per sample, we only have 10 bits available to us. Using only 10 bits, it would be impossible to encode the parameters of the vocal tract filter as well as the excitation vector. Therefore, the algorithm obtains the vocal tract filter parameters in a backward adaptive manner; that is, the vocal tract filter coefficients to be used to synthesize the current segment are obtained by analyzing the previous decoded segments. The CCITT requirements for G.728 included the requirement that the algorithm operate under noisy channel conditions. It would be extremely difficult to extract the pitch period from speech corrupted by channel errors. Therefore, the G.728 algorithm does away with the pitch filter. Instead, the algorithm uses a 50th-order vocal tract filter. The order of the filter is large enough to model the pitch of most female speakers. Not being able to use pitch information for male speakers does not cause much degradation [235]. The vocal tract filter is updated every fourth frame, which is once every 20 samples or 2.5 milliseconds. The autocorrelation method is used to obtain the vocal tract parameters.

As the vocal tract filter is completely determined in a backward adaptive manner, we have all 10 bits available to encode the excitation sequence. Ten bits would be able to index 1024 excitation sequences. However, to examine 1024 excitation sequences every 0.625 milliseconds is a rather large computational load. In order to reduce this load, the G.728 algorithm uses a product codebook where each excitation sequence is represented by a normalized sequence and a gain term. The final excitation sequence is a product of the normalized excitation sequence and the gain. Of the 10 bits, 3 bits are used to encode the gain using a predictive encoding scheme, while the remaining 7 bits form the index to a codebook containing 127 sequences.

Block diagrams of the encoder and decoder for the CCITT G.728 coder are shown in Figure 17.8. The low-delay CCITT G.728 CELP coder operating at 16 kbps provides reconstructed speech quality superior to the 32 kbps CCITT G.726 ADPCM algorithm described in Chapter 10. Various efforts are under way to reduce the bit rate for this algorithm without compromising too much on quality and delay.

17.3.4 Sinusoidal Coders

A competing approach to CELP in the low-rate region is a relatively new form of coder called the sinusoidal coder [220]. Recall that the main problem with the LPC coder was the paucity of excitation signals. The CELP coder resolved this problem by using a codebook of excitation signals. The sinusoidal coders solve this problem by using an excitation signal

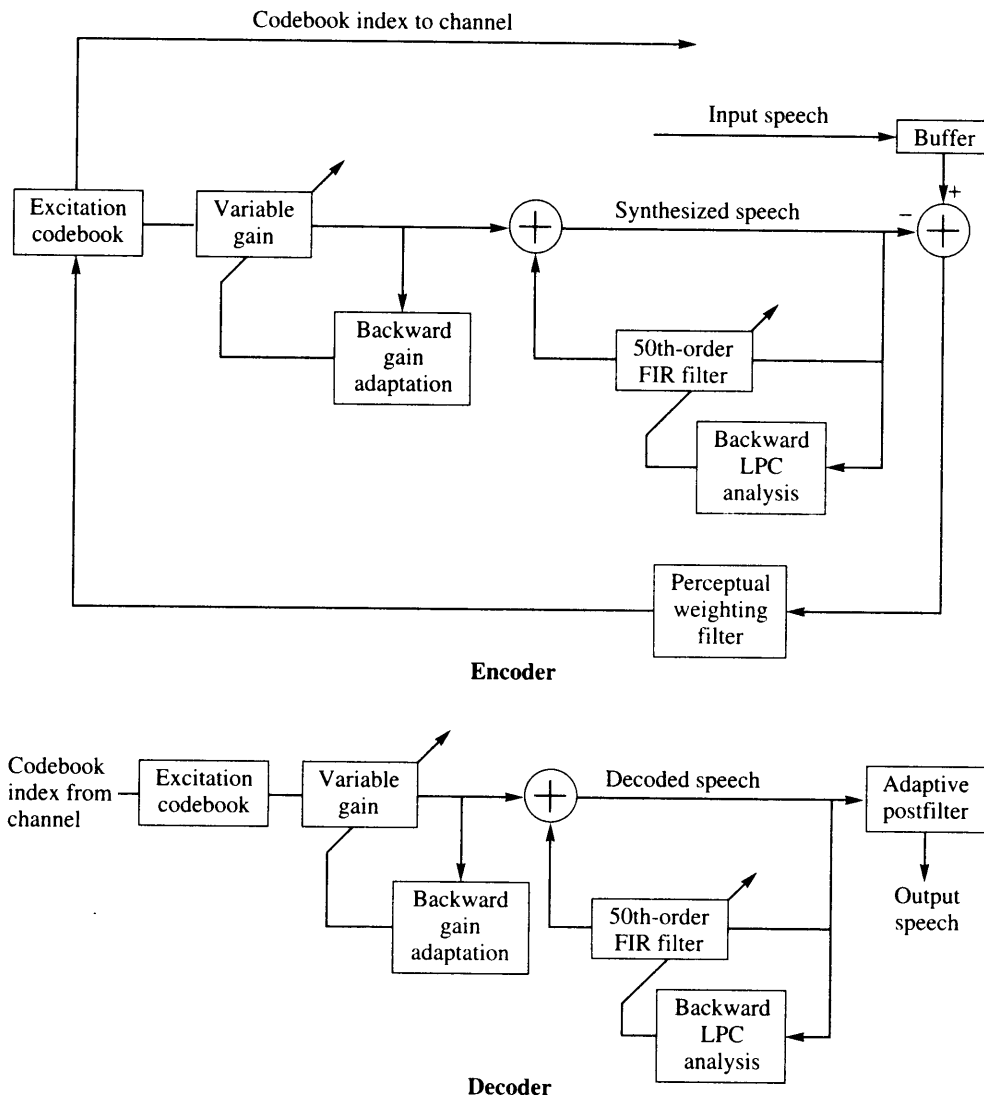


FIGURE 17.8 Encoder and decoder for the CCITT G.728 16 kbps speech coder.

that is the sum of sine waves of arbitrary amplitudes, frequencies, and phases. Thus, the excitation signal is of the form

$$e_n = \sum_{l=1}^L a_l \cos(n\omega_l + \psi_l) \quad (17.21)$$

where the number of sinusoids L required for each frame depends on the contents of the frame. If the input to a linear system is a sinusoid with frequency ω_l , the output will also be a sinusoid with frequency ω_l , albeit with different amplitude and phase. The vocal tract filter is a linear system. Therefore, if the excitation signal is of the form of (17.21), the synthesized speech $\{s_n\}$ will be of the form

$$s_n = \sum_{l=1}^L A_l \cos(n\omega_l + \phi_l). \quad (17.22)$$

Thus, each frame is characterized by a set of spectral amplitudes A_l , frequencies ω_l , and phase terms ϕ_l . The number of parameters required to represent the excitation sequence is the same as the number of parameters required to represent the synthesized speech. Therefore, rather than estimate and transmit the parameters of both the excitation signal and vocal tract filter and then synthesize the speech at the receiver by passing the excitation signal through the vocal tract filter, the sinusoidal coders directly estimate the parameters required to synthesize the speech at the receiver.

Just like the coders discussed previously, the sinusoidal coders divide the input speech into frames and obtain the parameters of the speech separately for each frame. If we synthesized the speech segment in each frame independent of the other frames, we would get synthetic speech that is discontinuous at the frame boundaries. These discontinuities severely degrade the quality of the synthetic speech. Therefore, the sinusoidal coders use different interpolation algorithms to smooth the transition from one frame to another.

Transmitting all the separate frequencies ω_l would require significant transmission resources, so the sinusoidal coders obtain a fundamental frequency ω_0 for which the approximation

$$\hat{y}_n = \sum_{k=1}^{K(\omega_0)} \hat{A}(k\omega_0) \cos(nk\omega_0 + \phi_k) \quad (17.23)$$

is close to the speech sequence y_n . Because this is a harmonic approximation, the approximate sequence $\{\hat{y}_n\}$ will be most different from the speech sequence $\{y_n\}$ when the segment of speech being encoded is unvoiced. Therefore, this difference can be used to decide whether the frame or some subset of it is unvoiced.

The two most popular sinusoidal coding techniques today are represented by the sinusoidal transform coder (STC) [236] and the multiband excitation coder (MBE) [237]. While the STC and MBE are similar in many respects, they differ in how they handle unvoiced speech. In the MBE coder, the frequency range is divided into bands, each consisting of several harmonics of the fundamental frequency ω_0 . Each band is checked to see if it is unvoiced or voiced. The voiced bands are synthesized using a sum of sinusoids, while the unvoiced bands are obtained using a random number generator. The voiced and unvoiced bands are synthesized separately and then added together.

In the STC, the proportion of the frame that contains a voiced signal is measured using a "voicing probability" P_v . The voicing probability is a function of how well the harmonic model matches the speech segment. Where the harmonic model is close to the speech signal,

the voicing probability is taken to be unity. The sine wave frequencies are then generated by

$$w_k = \begin{cases} kw_0 & \text{for } kw_0 \leq w_c P_v \\ k^*w_0 + (k - k^*)w_u & \text{for } kw_0 > w_c P_v \end{cases} \quad (17.24)$$

where w_c corresponds to the cutoff frequency (4 kHz), w_u is the unvoiced pitch corresponding to 100 Hz, and k^* is the largest value of k for which $k^*w_0 \leq w_c P_v$. The speech is then synthesized as

$$\hat{y}_n = \sum_{k=1}^K \hat{A}(w_k) \cos(nw_k + \phi_k). \quad (17.25)$$

Both the STC and the MBE coders have been shown to perform well at low rates. A version of the MBE coder known as the improved MBE (IMBE) coder has been approved by the Association of Police Communications Officers (APCO) as the standard for law enforcement.

17.3.5 Mixed Excitation Linear Prediction (MELP)

The mixed excitation linear prediction (MELP) coder was selected to be the new federal standard for speech coding at 2.4 kbps by the Defense Department Voice Processing Consortium (DDVPC). The MELP algorithm uses the same LPC filter to model the vocal tract. However, it uses a much more complex approach to the generation of the excitation signal.

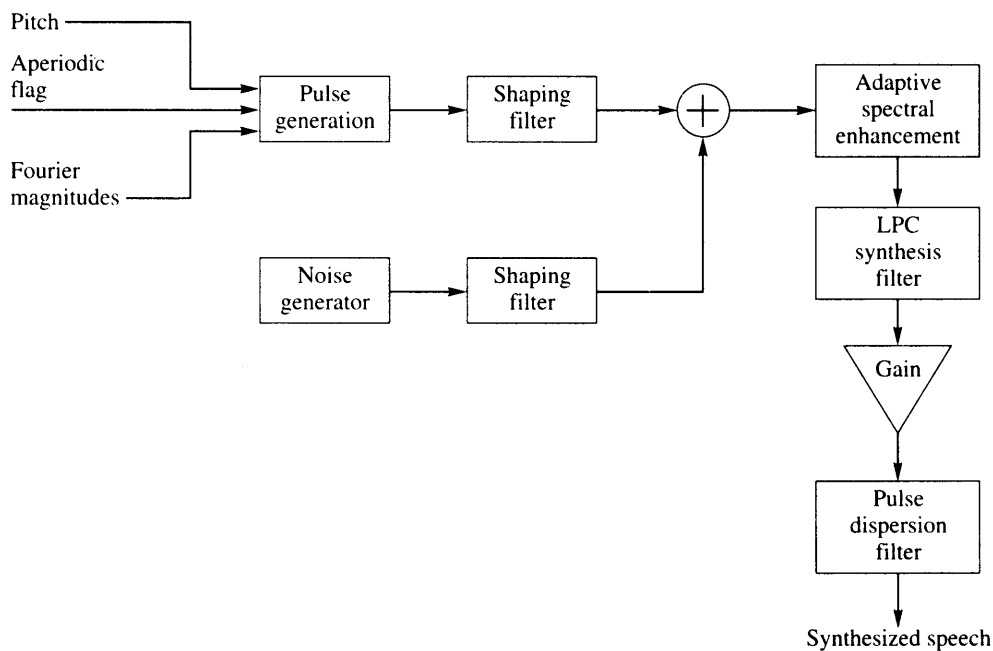


FIGURE 17. 9 Block diagram of MELP decoder.

A block diagram of the decoder for the MELP system is shown in Figure 17.9. As evident from the figure, the excitation signal for the synthesis filter is no longer simply noise or a periodic pulse but a multiband mixed excitation. The mixed excitation contains both a filtered signal from a noise generator as well as a contribution that depends directly on the input signal.

The first step in constructing the excitation signal is pitch extraction. The MELP algorithm obtains the pitch period using a multistep approach. In the first step an integer pitch value P_1 is obtained by

1. first filtering the input using a low-pass filter with a cutoff of 1 kHz
2. computing the normalized autocorrelation for lags between 40 and 160

The normalized autocorrelation $r(\tau)$ is defined as

$$r(\tau) = \frac{c_\tau(0, \tau)}{\sqrt{c_\tau(0, 0)c_\tau(\tau, \tau)}}$$

where

$$c_\tau(m, n) = \sum_{k=-\lfloor \tau/2 \rfloor - 80}^{-\lfloor \tau/2 \rfloor + 79} y_{k+m} y_{k+n}.$$

The first estimate of the pitch P_1 is obtained as the value of τ that maximizes the normalized autocorrelation function. This value is refined by looking at the signal filtered using a filter with passband in the 0–500 Hz range. This stage uses two values of P_1 , one from the current frame and one from the previous frame, as candidates. The normalized autocorrelation values are obtained for lags from five samples less to five samples more than the candidate P_1 values. The lags that provide the maximum normalized autocorrelation value for each candidate are used for *fractional pitch refinement*. The idea behind fractional pitch refinement is that if the maximum value of $r(\tau)$ is found for some $\tau = T$, then the maximum could be in the interval $(T - 1, T]$ or $[T, T + 1)$. The fractional offset is computed using

$$\Delta = \frac{c_T(0, T+1)c_T(T, T) - c_T(0, T)c_T(T, T+1)}{c_T(0, T+1)[c_T(T, T) - c_T(T, T+1)] + c_T(0, T)[c_T(T+1, T+1) - c_T(T, T+1)]}. \quad (17.26)$$

The normalized autocorrelation at the fractional pitch values are given by

$$r(T + \Delta) = \frac{(1 - \Delta)c_T(0, T) + \Delta c_T(0, T+1)}{\sqrt{c_T(0, 0)[(1 - \Delta)^2 c_T(T, T) + 2\Delta(1 - \Delta)c_T(T, T+1) + \Delta^2 c_T(T+1, T+1)]}}. \quad (17.27)$$

The fractional estimate that gives the higher autocorrelation is selected as the refined pitch value P_2 .

The final refinements of the pitch value are obtained using the linear prediction residuals. The residual sequence is generated by filtering the input speech signal with the filter obtained using the LPC analysis. For the purposes of pitch refinement the residual signal is filtered using a low-pass filter with a cutoff of 1 kHz. The normalized autocorrelation function is

computed for this filtered residual signal for lags from five samples less to five samples more than the candidate P_2 value, and a candidate value of P_3 is obtained. If $r(P_3) \geq 0.6$, we check to make sure that P_3 is not a multiple of the actual pitch. If $r(P_3) < 0.6$, we do another fractional pitch refinement around P_2 using the input speech signal. If in the end $r(P_3) < 0.55$, we replace P_3 with a long-term average value of the pitch. The final pitch value is quantized on a logarithmic scale using a 99-level uniform quantizer.

The input is also subjected to a multiband voicing analysis using five filters with passbands 0–500, 500–1000, 1000–2000, 2000–3000, and 3000–4000 Hz. The goal of the analysis is to obtain the voicing strengths Vbp_i for each band used in the shaping filters. Noting that P_2 was obtained using the output of the lowest band filter, $r(P_2)$ is assigned as the lowest band voicing strength Vbp_1 . For the other bands, Vbp_i is the larger of $r(P_2)$ for that band and the correlation of the envelope of the band-pass signal. If the value of Vbp_1 is small, this indicates a lack of low-frequency structure, which in turn indicates an unvoiced or transition input. Thus, if $Vbp_1 < 0.5$, the pulse component of the excitation signal is selected to be aperiodic, and this decision is communicated to the decoder by setting the aperiodic flag to 1. When $Vbp_1 > 0.6$, the values of the other voicing strengths are quantized to 1 if their value is greater than 0.6, and to 0 otherwise. In this way signal energy in the different bands is turned on or off depending on the voicing strength. There are several exceptions to this quantization rule. If Vbp_2 , Vbp_3 , and Vbp_4 all have magnitudes less than 0.6 and Vbp_5 has a value greater than 0.6, they are all (including Vbp_5) quantized to 0. Also, if the residual signal contains a few large values, indicating sudden transitions in the input signal, the voicing strengths are adjusted. In particular, the *peakiness* is defined as

$$\text{peakiness} = \frac{\sqrt{\frac{1}{160} \sum_{n=1}^{160} d_n^2}}{\frac{1}{160} \sum_{n=1}^{160} |d_n|}. \quad (17.28)$$

If this value exceeds 1.34, Vbp_1 is forced to 1. If the peakiness value exceeds 1.6, Vbp_1 , Vbp_2 , and Vbp_3 are all set to 1.

In order to generate the pulse input, the algorithm measures the magnitude of the discrete Fourier transform coefficients corresponding to the first 10 harmonics of the pitch. The prediction residual is generated using the quantized predictor coefficients. The algorithm searches in a window of width $\lfloor 512/\hat{P}_3 \rfloor$ samples around the initial estimates of the pitch harmonics for the actual harmonics where \hat{P}_3 is the quantized value of P_3 . The magnitudes of the harmonics are quantized using a vector quantizer with a codebook size of 256. The codebook is searched using a weighted Euclidean distance that emphasizes lower frequencies over higher frequencies.

At the decoder, using the magnitudes of the harmonics and information about the periodicity of the pulse train, the algorithm generates one excitation signal. Another signal is generated using a random number generator. Both are shaped by the multiband shaping filter before being combined. This mixture signal is then processed through an *adaptive spectral enhancement filter*, which is based on the LPC coefficients, to form the final excitation signal. Note that in order to preserve continuity from frame to frame, the parameters used for generating the excitation signal are adjusted based on their corresponding values in neighboring frames.

17.4 Wideband Speech Compression—ITU-T G.722.2

One of the earliest forms of (remote) speech communication was over the telephone. This experience set the expectations for quality rather low. When technology advanced, people still did not demand higher quality in their voice communications. However, the multimedia revolution is changing that. With ever-increasing quality in video and audio there is an increasing demand for higher quality in speech communication. Telephone-quality speech is limited to the band between 200 Hz and 3400 Hz. This range of frequency contains enough information to make speech intelligible and provide some degree of speaker identification. To improve the quality of speech, it is necessary to increase the bandwidth of speech. Wideband speech is bandlimited to 50–7000 Hz. The higher frequencies give more clarity to the voice signal while the lower frequencies contribute timbre and naturalness. The ITU-T G.722.2 standard, approved in January of 2002, provides a multirate coder for wideband speech coding.

Wideband speech is sampled at 16,000 samples per second. The signal is split into two bands, a lower band from 50–6400 Hz and a narrow upper band from 6400–7000 Hz. The coding resources are devoted to the lower band. The upper band is reconstructed at the receiver based on information from the lower band and using random excitation. The lower band is downsampled to 12.8 kHz.

The coding method is a code-excited linear prediction method that uses an algebraic codebook as the fixed codebook. The adaptive codebook contains low-pass interpolated past excitation vectors. The basic idea is the same as in CELP. A synthesis filter is derived from the input speech. An excitation vector consisting of a weighted sum of the fixed and adaptive codebooks is used to excite the synthesis filter. The perceptual closeness of the output of the filter to the input speech is used to select the combination of excitation vectors. The selection, along with the parameters of the synthesis filter, is communicated to the receiver, which then synthesizes the speech. A voice activity detector is used to reduce the rate during silence intervals. Let us examine the various components in slightly more detail.

The speech is processed in 20-ms frames. Each frame is composed of four 5-ms sub-frames. The LP analysis is conducted once per frame using an overlapping 30-ms window. Autocorrelation values are obtained for the windowed speech and the Levinson-Durbin algorithm is used to obtain the LP coefficients. These coefficients are transformed to Immitance Spectral Pairs (ISP), which are quantized using a vector quantizer. The reason behind the transformation is that we will need to quantize whatever representation we have of the synthesis filters. The elements of the ISP representation are uncorrelated if the underlying process is stationary, which means that error in one coefficient will not cause the entire spectrum to get distorted.

Given a set of sixteen LP coefficients $\{a_i\}$, define two polynomials

$$f_1'(z) = A(z) + z^{-16}A(z^{-1}) \quad (17.29)$$

$$f_2'(z) = A(z) - z^{-16}A(z^{-1}) \quad (17.30)$$

Clearly, if we know the polynomials their sum will give us $A(z)$. Instead of sending the polynomials, we can send the roots of these polynomials. These roots are known to all lie

on the unit circle, and the roots of the two polynomials alternate. The polynomial $f_2'(z)$ has two roots at $z = 1$ and $z = -1$. These are removed and we get the two polynomials

$$f_1(z) = f_1'(z) \quad (17.31)$$

$$f_2(z) = \frac{f_2'(z)}{1 - z^{-2}} \quad (17.32)$$

These polynomials can now be factored as follows

$$f_1(z) = (1 + a_{16}) \prod_{i=0,2,\dots,14} (1 - 2q_i z^{-i} + z^{-2}) \quad (17.33)$$

$$f_2(z) = (1 + a_{16}) \prod_{i=1,3,\dots,13} (1 - 2q_i z^{-i} + z^{-2}) \quad (17.34)$$

where $q_i = \cos(\omega_i)$ and ω_i are the immittance spectral frequencies. The ISP coefficients are quantized using a combination of differential encoding and vector quantization. The vector of sixteen frequencies is split into subvectors and these vectors are quantized in two stages. The quantized ISPs are transformed to LP coefficients, which are then used in the fourth subframe for synthesis. The ISP coefficients used in the other three subframes are obtained by interpolating the coefficients in the neighboring subframes.

For each 5-ms subframe we need to generate an excitation vector. As in CELP, the excitation is a sum of vectors from two codebooks, a fixed codebook and an adaptive codebook. One of the problems with vector codebooks has always been the storage requirements. The codebook should be large enough to provide for a rich set of excitations. However, with a dimension of 64 samples (for 5 ms), the number of possible combinations can get enormous. The G.722.2 algorithm solves this problem by imposing an algebraic structure on the fixed codebook. The 64 positions are divided into four tracks. The first track consists of positions 0, 4, 8, . . . , 60. The second track consists of the positions 1, 5, 9, . . . , 61. The third track consists of positions 2, 6, 10, . . . , 62 and the final track consists of the remaining positions. We can place a single signed pulse in each track by using 4 bits to denote the position and a fifth bit for the sign. This effectively gives us a 20-bit fixed codebook. This corresponds to a codebook size of 2^{20} . However, we do not need to store the codebook. By assigning more or fewer pulses per track we can dramatically change the “size” of the codebook and get different coding rates. The standard details a rapid search procedure to obtain the excitation vectors.

The voice activity detector allows the encoder to significantly reduce the rate during periods of speech pauses. During these periods the background noise is coded at a low rate by transmitting parameters describing the noise. This *comfort noise* is synthesized at the decoder.

17.5 Image Compression

Although there have been a number of attempts to mimic the linear predictive coding approach for image compression, they have not been overly successful. A major reason for this is that while speech can be modeled as the output of a linear filter, most images cannot.

However, a totally different analysis/synthesis approach, conceived in the mid-1980s, has found some degree of success—fractal compression.

17.5.1 Fractal Compression

There are several different ways to approach the topic of fractal compression. Our approach is to use the idea of fixed-point transformation. A function $f(\cdot)$ is said to have a fixed point x_0 if $f(x_0) = x_0$. Suppose we restrict the function $f(\cdot)$ to be of the form $ax + b$. Then, except for when $a = 1$, this equation always has a fixed point:

$$\begin{aligned} ax_0 + b &= x_0 \\ \Rightarrow x_0 &= \frac{b}{1-a}. \end{aligned} \quad (17.35)$$

This means that if we wanted to transmit the value of x_0 , we could instead transmit the values of a and b and obtain x_0 at the receiver using (17.35). We do not have to solve this equation to obtain x_0 . Instead, we could take a guess at what x_0 should be and then refine the guess using the recursion

$$x_0^{(n+1)} = ax_0^{(n)} + b. \quad (17.36)$$

Example 17.5.1:

Suppose that instead of sending the value $x_0 = 2$, we sent the values of a and b as 0.5 and 1.0. The receiver starts out with a guess for x_0 as $x_0^{(0)} = 1$. Then

$$\begin{aligned} x_0^{(1)} &= ax_0^{(0)} + b = 1.5 \\ x_0^{(2)} &= ax_0^{(1)} + b = 1.75 \\ x_0^{(3)} &= ax_0^{(2)} + b = 1.875 \\ x_0^{(4)} &= ax_0^{(3)} + b = 1.9375 \\ x_0^{(5)} &= ax_0^{(4)} + b = 1.96875 \\ x_0^{(6)} &= ax_0^{(5)} + b = 1.984375 \end{aligned} \quad (17.37)$$

and so on. As we can see, with each iteration we come closer and closer to the actual x_0 value of 2. This would be true no matter what our initial guess was. \blacklozenge

Thus, the value of x_0 is accurately specified by specifying the fixed-point equation. The receiver can retrieve the value either by the solution of (17.35) or via the recursion (17.36).

Let us generalize this idea. Suppose that for a given image \mathcal{J} (treated as an array of integers), there exists a function $f(\cdot)$ such that $f(\mathcal{J}) = \mathcal{J}$. If it was cheaper in terms of bits to represent $f(\cdot)$ than it was to represent \mathcal{J} , we could treat $f(\cdot)$ as the compressed representation of \mathcal{J} .

This idea was first proposed by Michael Barnsley and Alan Sloan [238] based on the idea of self-similarity. Barnsley and Sloan noted that certain natural-looking objects can be obtained as the fixed point of a certain type of function. If an image can be obtained as a fixed point of some function, can we then solve the *inverse* problem? That is, given an image, can we find the function for which the image is the fixed point? The first practical public answer to this came from Arnaud Jacquin in his Ph.D. dissertation [239] in 1989. The technique we describe in this section is from Jacquin's 1992 paper [240].

Instead of generating a single function directly for which the given image is a fixed point, we partition the image into blocks R_k , called *range* blocks, and obtain a transformation f_k for each block. The transformations f_k are not fixed-point transformations since they do not satisfy the equation

$$f_k(R_k) = R_k. \quad (17.38)$$

Instead, they are a mapping from a block of pixels D_k from some other part of the image. While each individual mapping f_k is not a fixed-point mapping, we will see later that we can combine all these mappings to generate a fixed-point mapping. The image blocks D_k are called *domain* blocks, and they are chosen to be larger than the range blocks. In [240], the domain blocks are obtained by sliding a $K \times K$ window over the image in steps of $K/2$ or $K/4$ pixels. As long as the window remains within the boundaries of the image, each $K \times K$ block thus encountered is entered into the domain pool. The set of all domain blocks does not have to partition the image. In Figure 17.10 we show the range blocks and two possible domain blocks.

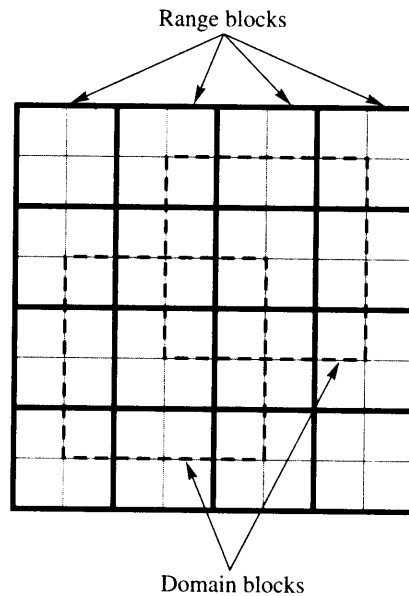


FIGURE 17. 10 Range blocks and examples of domain blocks.

The transformations f_k are composed of a *geometric* transformation g_k and a *massic* transformation m_k . The geometric transformation consists of moving the domain block to the location of the range block and adjusting the size of the domain block to match the size of the range block. The massic transformation adjusts the intensity and orientation of the pixels in the domain block after it has been operated on by the geometric transform. Thus,

$$\hat{R}_k = f_k(D_k) = m_k(g_k(D_k)). \quad (17.39)$$

We have used \hat{R}_k instead of R_k on the left-hand side of (17.39) because it is generally not possible to find an exact functional relationship between domain and range blocks. Therefore, we have to settle for some degree of loss of information. Generally, this loss is measured in terms of mean squared error.

The effect of all these functions together can be represented as the transformation $f(\cdot)$. Mathematically, this transformation can be viewed as a union of the transformations f_k :

$$f = \bigcup_k f_k. \quad (17.40)$$

Notice that while each transformation f_k maps a block of different size and location to the location of R_k , looking at it from the point of view of the entire image, it is a mapping from the image to the image. As the union of R_k is the image itself, we could represent all the transformations as

$$\hat{I} = f(\hat{I}) \quad (17.41)$$

where we have used \hat{I} instead of I to account for the fact that the reconstructed image is an approximation to the original.

We can now pose the encoding problem as that of obtaining D_k , g_k , and m_k such that the difference $d(R_k, \hat{R}_k)$ is minimized, where $d(R_k, \hat{R}_k)$ can be the mean squared error between the blocks R_k and \hat{R}_k .

Let us first look at how we would obtain g_k and m_k assuming that we already know which domain block D_k we are going to use. We will then return to the question of selecting D_k .

Knowing which domain block we are using for a given range block automatically specifies the amount of displacement required. If the range blocks R_k are of size $M \times M$, then the domain blocks are usually taken to be of size $2M \times 2M$. In order to adjust the size of D_k to be the same as that of R_k , we generally replace each 2×2 block of pixels with their average value. Once the range block has been selected, the geometric transformation is easily obtained.

Let's define $T_k = g_k(D_k)$, and t_{ij} as the ij th pixel in T_k , $i, j = 0, 1, \dots, M-1$. The massic transformation m_k is then given by

$$m_k(t_{ij}) = i(\alpha_k t_{ij} + \Delta_k) \quad (17.42)$$

where $i(\cdot)$ denotes a shuffling or rearrangement of the pixels with the block. Possible rearrangements (or *isometries*) include the following:

1. Rotation by 90 degrees, $i(t_{ij}) = t_{j(M-1-i)}$
2. Rotation by 180 degrees, $i(t_{ij}) = t_{(M-1-i)(M-1-j)}$

3. Rotation by -90 degrees, $i(t_{ij}) = t_{(M-1-i)j}$
4. Reflection about midvertical axis, $i(t_{ij}) = t_{i(M-1-j)}$
5. Reflection about midhorizontal axis, $i(t_{ij}) = t_{(M-1-i)j}$
6. Reflection about diagonal, $i(t_{ij}) = t_{ji}$
7. Reflection about cross diagonal, $i(t_{ij}) = t_{(M-1-j)(M-1-i)}$
8. Identity mapping, $i(t_{ij}) = t_{ij}$

Therefore, for each massic transformation m_k , we need to find values of α_k , Δ_k , and an isometry. For a given range block R_k , in order to find the mapping that gives us the closest approximation \hat{R}_k , we can try all possible combinations of transformations and domain blocks—a massive computation. In order to reduce the computations, we can restrict the number of domain blocks to search. However, in order to get the best possible approximation, we would like the pool of domain blocks to be as large as possible. Jacquin [240] resolves this situation in the following manner. First, he generates a relatively large pool of domain blocks by the method described earlier. The elements of the domain pool are then divided into *shade blocks*, *edge blocks*, and *midrange blocks*. The shade blocks are those in which the variance of pixel values within the block is small. The edge block, as the name implies, contains those blocks that have a sharp change of intensity values. The midrange blocks are those that fit into neither category—not too smooth but with no well-defined edges. The shade blocks are then removed from the domain pool. The reason is that, given the transformations we have described, a shade domain block can only generate a shade range block. If the range block is a shade block, it is much more cost effective simply to send the average value of the block rather than attempt any more complicated transformations.

The encoding procedure proceeds as follows. A range block is first classified into one of the three categories described above. If it is a shade block, we simply send the average value of the block. If it is a midrange block, the massic transformation is of the form $\alpha_k t_{ij} + \Delta_k$. The isometry is assumed to be the identity isometry. First α_k is selected from a small set of values—Jacquin [240] uses the values (0.7, 0.8, 0.9, 1.0)—such that $d(R_k, \alpha_k T_k)$ is minimized. Thus, we have to search over the possible values of α and the midrange domain blocks in the domain pool in order to find the (α_k, D_k) pair that will minimize $d(R_k, \alpha_k T_k)$. The value of Δ_k is then selected as the difference of the average values of R_k and $\alpha_k T_k$.

If the range block R_k is classified as an edge block, selection of the massic transformation is a somewhat more complicated process. The block is first divided into a bright and a dark region. The dynamic range of the block $r_d(R_k)$ is then computed as the difference of the average values of the light and dark regions. For a given domain block, this is then used to compute the value of α_k by

$$\alpha_k = \min \left\{ \frac{r_d(R_k)}{r_d(T_j)}, \alpha_{\max} \right\} \quad (17.43)$$

where α_{\max} is an upper bound on the scaling factor. The value of α_k obtained in this manner is then quantized to one of a small set of values. Once the value of α_k has been obtained, Δ_k is obtained as the difference of either the average values of the bright regions or the average values of the dark regions, depending on whether we have more pixels in the dark regions

or the light regions. Finally, each of the isometries is tried out to find the one that gives the closest match between the transformed domain block and the range block.

Once the transformations have been obtained, they are communicated to the receiver in terms of the following parameters: the location of the selected domain block and a single bit denoting whether the block is a shade block or not. If it is a shade block, the average intensity value is transmitted; if it is not, the quantized scale factor and offset are transmitted along with the label of the isometry used.

The receiver starts out with some arbitrary initial image I_0 . The transformations are then applied for each of the range blocks to obtain the first approximation. Then the transformations are applied to the first approximation to get the second approximation, and so on. Let us see an example of the decoding process.

Example 17.5.2:

The image Elif, shown in Figure 17.11, was encoded using the fractal approach. The original image was of size 256×256 , and each pixel was coded using 8 bits. Therefore, the storage space required was 65,536 bytes. The compressed image consisted of the transformations described above. The transformations required a total of 4580 bytes, which translates to an average rate of 0.56 bits per pixel. The decoding process started with the transformations being applied to an all-zero image. The first six iterations of the decoding process are shown in Figure 17.12. The process converged in nine iterations. The final image is shown in Figure 17.13. Notice the difference in this reconstructed image and the low-rate reconstructed image obtained using the DCT. The blocking artifacts are for the most part gone. However,



FIGURE 17.11 Original Elif image.



FIGURE 17. 12 The first six iterations of the fractal decoding process.



FIGURE 17.13 Final reconstructed Elif image.

this does not mean that the reconstruction is free of distortions and artifacts. They are especially visible in the chin and neck region. ♦

In our discussion (and illustration) we have assumed that the size of the range blocks is constant. If so, how large should we pick the range block? If we pick the size of the range block to be large, we will have to send fewer transformations, thus improving the compression. However, if the size of the range block is large, it becomes more difficult to find a domain block that, after appropriate transformation, will be close to the range block, which in turn will increase the distortion in the reconstructed image. One compromise between picking a large or small value for the size of the range block is to start out with a large size and, if a good enough match is not found, to progressively reduce the size of the range block until we have either found a good match or reached a minimum size. We could also compute a weighted sum of the rate and distortion

$$J = D + \beta R$$

where D is a measure of the distortion, and R represents the number of bits required to represent the block. We could then either subdivide or not depending on the value of J .

We can also start out with range blocks that have the minimum size (also called the *atomic blocks*) and obtain larger blocks via merging smaller blocks.

There are a number of ways in which we can perform the subdivision. The most commonly known approach is *quadtree partitioning*, initially introduced by Samet [241]. In quadtree partitioning we start by dividing up the image into the maximum-size range

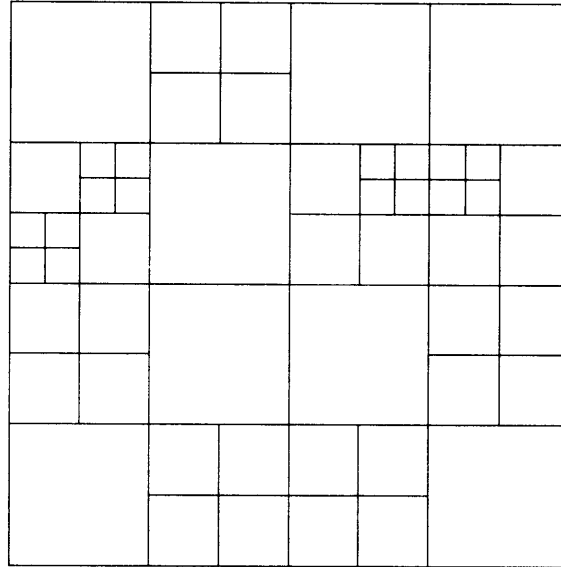


FIGURE 17. 14 An example of quadtree partitioning.

blocks. If a particular block does not have a satisfactory reconstruction, we can divide it up into four blocks. These blocks in turn can also, if needed, be divided into four blocks. An example of quadtree partitioning can be seen in Figure 17.14. In this particular case there are three possible sizes for the range blocks. Generally, we would like to keep the minimum size of the range block small if fine detail in the image is of greater importance [242]. Since we have multiple sizes for the range blocks, we also need multiple sizes for the domain blocks.

Quadtree partitioning is not the only method of partitioning available. Another popular method of partitioning is the HV method. In this method we allow the use of rectangular regions. Instead of dividing a square region into four more square regions, rectangular regions are divided either vertically or horizontally in order to generate more homogeneous regions. In particular, if there are vertical or horizontal edges in a block, it is partitioned along these edges. One way to obtain the locations of partitions for a given $M \times N$ range block is to calculate the biased vertical and horizontal differences:

$$v_i = \frac{\min(i, N-i-1)}{N-1} \left(\sum_j \mathcal{J}_{i,j} - \sum_j \mathcal{J}_{i+1,j} \right)$$

$$h_j = \frac{\min(j, M-j-1)}{M-1} \left(\sum_j \mathcal{J}_{i,j} - \sum_j \mathcal{J}_{i,j+1} \right)$$

The values of i and j for which $|v_i|$ and $|h_j|$ are the largest indicate the row and column for which there is maximum difference between two halves of the block. Depending on whether $|v_i|$ or $|h_j|$ is larger, we can divide the rectangle either vertically or horizontally.

Finally, partitioning does not have to be rectangular, or even regular. People have experimented with triangle partitions as well as irregular-shaped partitions [243].

The fractal approach is a novel way of looking at image compression. At present the quality of the reconstructions using the fractal approach is about the same as the quality of the reconstruction using the DCT approach employed in JPEG. However, the fractal technique is relatively new, and further research may bring significant improvements. The fractal approach has one significant advantage: decoding is simple and fast. This makes it especially useful in applications where compression is performed once and decompression is performed many times.

17.6 Summary

We have looked at two very different ways of using the analysis/synthesis approach. In speech coding the approach works because of the availability of a mathematical model for the speech generation process. We have seen how this model can be used in a number of different ways, depending on the constraints of the problem. Where the primary objective is to achieve intelligible communication at the lowest rate possible, the LPC algorithm provides a very nice solution. If we also want the quality of the speech to be high, CELP and the different sinusoidal techniques provide higher quality at the cost of more complexity and processing delay. If delay also needs to be kept below a threshold, one particular solution is the low-delay CELP algorithm in the G.728 recommendation. For images, fractal coding provides a very different way to look at the problem. Instead of using the physical structure of the system to generate the source output, it uses a more abstract view to obtain an analysis/synthesis technique.

Further Reading

1. For information about various aspects of speech processing, *Voice and Speech Processing*, by T. Parsons [105], is a very readable source.
2. The classic tutorial on linear prediction is "Linear Prediction," by J. Makhoul [244], which appeared in the April 1975 issue of the *Proceedings of the IEEE*.
3. For a thorough review of recent activity in speech compression, see "Advances in Speech and Audio Compression," by A. Gersho [220], which appeared in the June 1994 issue of the *Proceedings of the IEEE*.
4. An excellent source for information about speech coders is *Digital Speech: Coding for Low Bit Rate Communication Systems*, by A. Kondo [127].
5. An excellent description of the G.728 algorithm can be found in "A Low Delay CELP Coder for the CCITT 16 kb/s Speech Coding Standard," by J.-H. Chen, R.V. Cox,

Y.-C. Lin, N. Jayant, and M.J. Melchner [235], in the June 1992 issue of the *IEEE Journal on Selected Areas in Communications*.

6. A good introduction to fractal image compression is *Fractal Image Compression: Theory and Application*, Y. Fisher (ed.) [242], New York: Springer-Verlag, 1995.
7. The October 1993 issue of the *Proceedings of the IEEE* contains a special section on fractals with a tutorial on fractal image compression by A. Jacquin.

17.7 Projects and Problems

1. Write a program for the detection of voiced and unvoiced segments using the AMDF function. Test your algorithm on the `test.snd` sound file.
2. The `testf.raw` file is a female voice saying the word *test*. Isolate 10 voiced and unvoiced segments from the `testm.raw` file and the `testf.snd` file. (Try to pick the same segments in the two files.) Compute the number of zero crossings in each segment and compare your results for the two files.
3. (a) Select a voiced segment from the `testf.raw` file. Find the fourth-, sixth-, and tenth-order LPC filters for this segment using the Levinson-Durbin algorithm.
(b) Pick the corresponding segment from the `testf.snd` file. Find the fourth-, sixth-, and tenth-order LPC filters for this segment using the Levinson-Durbin algorithm.
(c) Compare the results of (a) and (b).
4. Select a voiced segment from the `test.raw` file. Find the fourth-, sixth-, and tenth-order LPC filters for this segment using the Levinson-Durbin algorithm. For each of the filters, find the multipulse sequence that results in the closest approximation to the voiced signal.

Video Compression

18.1 Overview

Video compression can be viewed as image compression with a temporal component since video consists of a time sequence of images. From this point of view, the only “new” technique introduced in this chapter is a strategy to take advantage of this temporal correlation. However, there are different situations in which video compression becomes necessary, each requiring a solution specific to its peculiar conditions. In this chapter we briefly look at video compression algorithms and standards developed for different video communications applications.

18.2 Introduction

Of all the different sources of data, perhaps the one that produces the largest amount of data is video. Consider a video sequence generated using the CCIR 601 format (Section 18.4). Each image frame is made up of more than a quarter million pixels. At the rate of 30 frames per second and 16 bits per pixel, this corresponds to a data rate of about 21 Mbytes or 168 Mbits per second. This is certainly a change from the data rates of 2.4, 4.8, and 16 kbits per second that are the targets for speech coding systems discussed in Chapter 17.

Video compression can be viewed as the compression of a sequence of images; in other words, image compression with a temporal component. This is essentially the approach we will take in this chapter. However, there are limitations to this approach. We do not perceive motion video in the same manner as we perceive still images. Motion video may mask coding artifacts that would be visible in still images. On the other hand, artifacts that may not be visible in reconstructed still images can be very annoying in reconstructed motion video sequences. For example, consider a compression scheme that introduces a modest random amount of change in the average intensity of the pixels in the image. Unless a reconstructed

still image was being compared side by side with the original image, this artifact may go totally unnoticed. However, in a motion video sequence, especially one with low activity, random intensity changes can be quite annoying. As another example, poor reproduction of edges can be a serious problem in the compression of still images. However, if there is some temporal activity in the video sequence, errors in the reconstruction of edges may go unnoticed.

Although a more holistic approach might lead to better compression schemes, it is more convenient to view video as a sequence of correlated images. Most of the video compression algorithms make use of the temporal correlation to remove redundancy. The previous reconstructed frame is used to generate a prediction for the current frame. The difference between the prediction and the current frame, the prediction error or residual, is encoded and transmitted to the receiver. The previous reconstructed frame is also available at the receiver. Therefore, if the receiver knows the manner in which the prediction was performed, it can use this information to generate the prediction values and add them to the prediction error to generate the reconstruction. The prediction operation in video coding has to take into account motion of the objects in the frame, which is known as motion compensation (described in the next section).

We will also describe a number of different video compression algorithms. For the most part, we restrict ourselves to discussions of techniques that have found their way into international standards. Because there are a significant number of products that use proprietary video compression algorithms, it is difficult to find or include descriptions of them.

We can classify the algorithms based on the application area. While attempts have been made to develop standards that are “generic,” the application requirements can play a large part in determining the features to be used and the values of parameters. When the compression algorithm is being designed for two-way communication, it is necessary for the coding delay to be minimal. Furthermore, compression and decompression should have about the same level of complexity. The complexity can be unbalanced in a broadcast application, where there is one transmitter and many receivers, and the communication is essentially one-way. In this case, the encoder can be much more complex than the receiver. There is also more tolerance for encoding delays. In applications where the video is to be decoded on workstations and personal computers, the decoding complexity has to be extremely low in order for the decoder to decode a sufficient number of images to give the illusion of motion. However, as the encoding is generally not done in real time, the encoder can be quite complex. When the video is to be transmitted over packet networks, the effects of packet loss have to be taken into account when designing the compression algorithm. Thus, each application will present its own unique requirements and demand a solution that fits those requirements.

We will assume that you are familiar with the particular image compression technique being used. For example, when discussing transform-based video compression techniques, we assume that you have reviewed Chapter 13 and are familiar with the descriptions of transforms and the JPEG algorithm contained in that chapter.

18.3 Motion Compensation

In most video sequences there is little change in the contents of the image from one frame to the next. Even in sequences that depict a great deal of activity, there are significant portions of the image that do not change from one frame to the next. Most video compression schemes take advantage of this redundancy by using the previous frame to generate a prediction for the current frame. We have used prediction previously when we studied differential encoding schemes. If we try to apply those techniques blindly to video compression by predicting the value of each pixel by the value of the pixel at the same location in the previous frame, we will run into trouble because we would not be taking into account the fact that objects tend to move between frames. Thus, the object in one frame that was providing the pixel at a certain location (i_0, j_0) with its intensity value might be providing the same intensity value in the next frame to a pixel at location (i_1, j_1) . If we don't take this into account, we can actually increase the amount of information that needs to be transmitted.

Example 18.3.1:

Consider the two frames of a motion video sequence shown in Figure 18.1. The only differences between the two frames are that the devious looking individual has moved slightly downward and to the right of the frame, while the triangular object has moved to the left. The differences between the two frames are so slight, you would think that if the first frame was available to both the transmitter and receiver, not much information would need to be transmitted to the receiver in order to reconstruct the second frame. However, if we simply

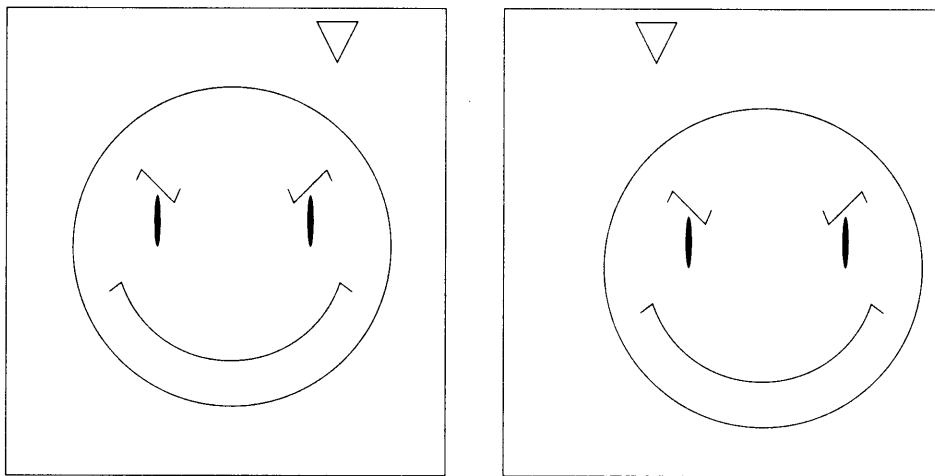


FIGURE 18.1 Two frames of a video sequence.

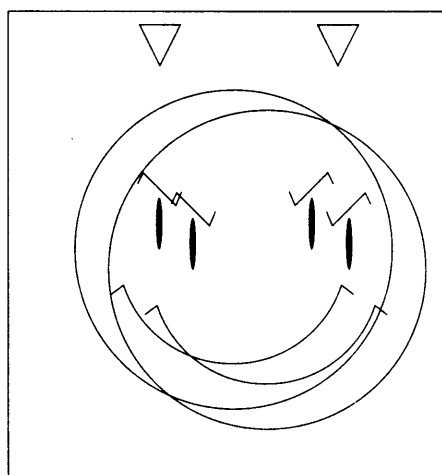


FIGURE 18.2 Difference between the two frames.

take the difference between the two frames, as shown in Figure 18.2, the displacement of the objects in the frame results in an image that contains more detail than the original image. In other words, instead of the differencing operation reducing the information, there is actually more information that needs to be transmitted. ♦

In order to use a previous frame to predict the pixel values in the frame being encoded, we have to take the motion of objects in the image into account. Although a number of approaches have been investigated, the method that has worked best in practice is a simple approach called *block-based motion compensation*. In this approach, the frame being encoded is divided into blocks of size $M \times M$. For each block, we search the previous reconstructed frame for the block of size $M \times M$ that most closely matches the block being encoded. We can measure the closeness of a match, or distance, between two blocks by the sum of absolute differences between corresponding pixels in the two blocks. We would obtain the same results if we used the sum of squared differences between the corresponding pixels as a measure of distance. Generally, if the distance from the block being encoded to the closest block in the previous reconstructed frame is greater than some prespecified threshold, the block is declared uncompensable and is encoded without the benefit of prediction. This decision is also transmitted to the receiver. If the distance is below the threshold, then a *motion vector* is transmitted to the receiver. The motion vector is the relative location of the block to be used for prediction obtained by subtracting the coordinates of the upper-left corner pixel of the block being encoded from the coordinates of the upper-left corner pixel of the block being used for prediction.

Suppose the block being encoded is an 8×8 block between pixel locations (24, 40) and (31, 47); that is, the upper-left corner pixel of the 8×8 block is at location (24, 40). If the block that best matches it in the previous frame is located between pixels at location (21, 43) and (28, 50), then the motion vector would be $(-3, 3)$. The motion vector was

obtained by subtracting the location of the upper-left corner of the block being encoded from the location of the upper-left corner of the best matching block. Note that the blocks are numbered starting from the top-left corner. Therefore, a positive x component means that the best matching block in the previous frame is to the right of the location of the block being encoded. Similarly, a positive y component means that the best matching block is at a location below that of the location of the block being encoded.

Example 18.3.2:

Let us again try to predict the second frame of Example 18.3.1 using motion compensation. We divide the image into blocks and then predict the second frame from the first in the manner described above. Figure 18.3 shows the blocks in the previous frame that were used to predict some of the blocks in the current frame.

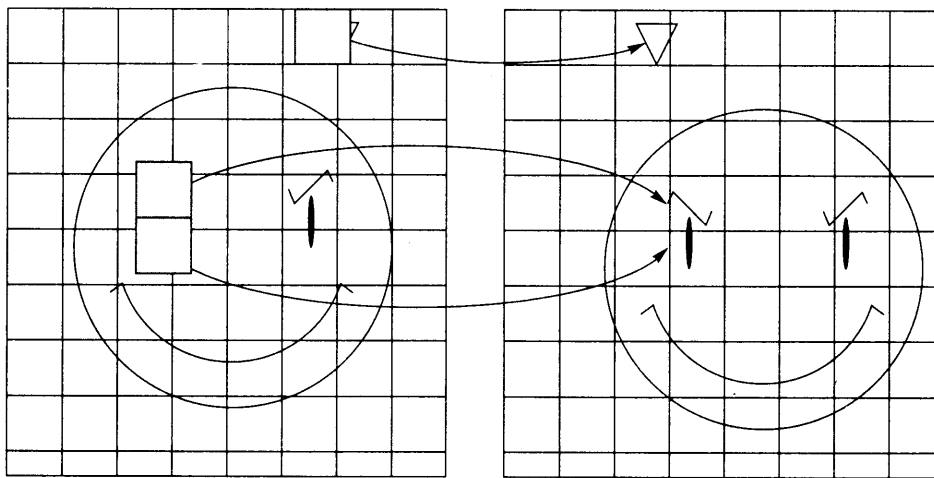


FIGURE 18.3 Motion-compensated prediction.

Notice that in this case all that needs to be transmitted to the receiver are the motion vectors. The current frame is completely predicted by the previous frame. ♦

We have been describing motion compensation where the displacement between the block being encoded and the best matching block is an integer number of pixels in the horizontal and vertical directions. There are algorithms in which the displacement is measured in half pixels. In order to do this, pixels of the coded frame being searched are interpolated to obtain twice as many pixels as in the original frame. This “doubled” image is then searched for the best matching block.

TABLE 18.1 "Doubled" image.

| | | |
|-------|-------|-------|
| A | h_1 | B |
| v_1 | c | v_2 |
| C | h_2 | D |

The doubled image is obtained as follows: Consider Table 18.1. In this image A , B , C , and D are the pixels of the original frame. The pixels h_1 , h_2 , v_1 , and v_2 are obtained by interpolating between the two neighboring pixels:

$$\begin{aligned}
 h_1 &= \left\lfloor \frac{A+B}{2} + 0.5 \right\rfloor \\
 h_2 &= \left\lfloor \frac{C+D}{2} + 0.5 \right\rfloor \\
 v_1 &= \left\lfloor \frac{A+C}{2} + 0.5 \right\rfloor \\
 v_2 &= \left\lfloor \frac{B+D}{2} + 0.5 \right\rfloor
 \end{aligned} \tag{18.1}$$

while the pixel c is obtained as the average of the four neighboring pixels from the coded original:

$$c = \left\lfloor \frac{A+B+C+D}{4} + 0.5 \right\rfloor.$$

We have described motion compensation in very general terms in this section. The various schemes in this chapter use specific motion compensation schemes that differ from each other. The differences generally involve the region of search for the matching block and the search procedure. We will look at the details with the study of the compression schemes. But before we begin our study of compression schemes, we briefly discuss how video signals are represented in the next section.

18.4 Video Signal Representation

The development of different representations of video signals has depended a great deal on past history. We will also take a historical view, starting with black-and-white television proceeding to digital video formats. The history of the development of analog video signal formats for the United States has been different than for Europe. Although we will show the development using the formats used in the United States, the basic ideas are the same for all formats.

A black-and-white television picture is generated by exciting the phosphor on the television screen using an electron beam whose intensity is modulated to generate the image we see. The path that the modulated electron beam traces is shown in Figure 18.4. The line created by the horizontal traversal of the electron beam is called a line of the image. In order

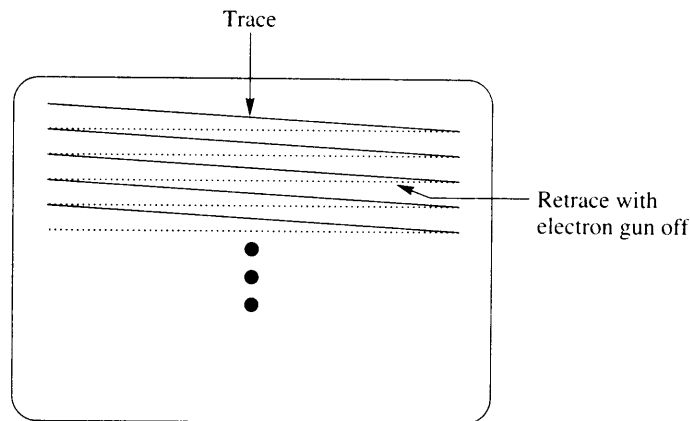


FIGURE 18.4 The path traversed by the electron beam in a television.

to trace a second line, the electron beam has to be deflected back to the left of the screen. During this period, the gun is turned off in order to prevent the retrace from becoming visible. The image generated by the traversal of the electron gun has to be updated rapidly enough for persistence of vision to make the image appear stable. However, higher rates of information transfer require higher bandwidths, which translate to higher costs.

In order to keep the cost of bandwidth low it was decided to send 525 lines 30 times a second. These 525 lines are said to constitute a *frame*. However, a thirtieth of a second between frames is long enough for the image to appear to flicker. To avoid the flicker, it was decided to divide the image into two interlaced fields. A field is sent once every sixtieth of a second. First, one field consisting of 262.5 lines is traced by the electron beam. Then, the second field consisting of the remaining 262.5 lines is traced *between* the lines of the first field. The situation is shown schematically in Figure 18.5. The first field is shown with

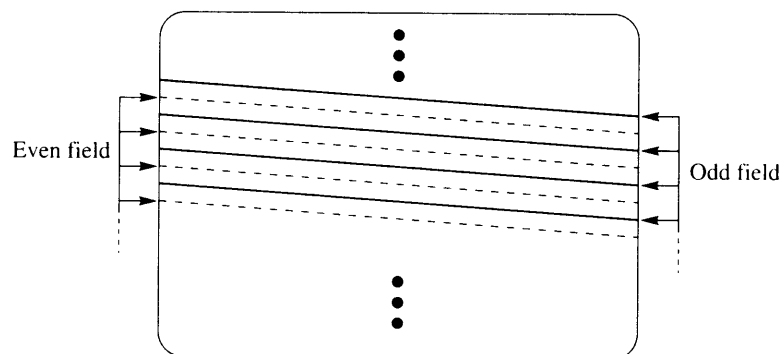


FIGURE 18.5 A frame and its constituent fields.

solid lines while the second field is shown with dashed lines. The first field begins on a full line and ends on a half line while the second field begins on a half line and ends on a full line. Not all 525 lines are displayed on the screen. Some are lost due to the time required for the electron gun to position the beam from the bottom to the top of the screen. We actually see about 486 lines per frame.

In a color television, instead of a single electron gun, we have three electron guns that act in unison. These guns excite red, green, and blue phosphor dots embedded in the screen. The beam from each gun strikes only one kind of phosphor, and the gun is named according to the color of the phosphor it excites. Thus, the red gun strikes only the red phosphor, the blue gun strikes only the blue phosphor, and the green gun strikes only the green phosphor. (Each gun is prevented from hitting a different type of phosphor by an aperture mask.)

In order to control the three guns we need three signals: a red signal, a blue signal, and a green signal. If we transmitted each of these separately, we would need three times the bandwidth. With the advent of color television, there was also the problem of backward compatibility. Most people had black-and-white television sets, and television stations did not want to broadcast using a format that most of the viewing audience could not see on their existing sets. Both issues were resolved with the creation of a composite color signal. In the United States, the specifications for the composite signal were created by the National Television Systems Committee, and the composite signal is often called an NTSC signal. The corresponding signals in Europe are PAL (Phase Alternating Lines), developed in Germany, and SECAM (Séquential Couleur avec Mémoire), developed in France. There is some (hopefully) good-natured rivalry between proponents of the different systems. Some problems with color reproduction in the NTSC signal have led to the name *Never Twice the Same Color*, while the idiosyncracies of the SECAM system have led to the name *Système Essentiellement Contre les Américains* (system essentially against the Americans).

The composite color signal consists of a *luminance* component, corresponding to the black-and-white television signal, and two *chrominance* components. The luminance component is denoted by Y :

$$Y = 0.299R + 0.587G + 0.114B \quad (18.2)$$

where R is the red component, G is the green component, and B is the blue component. The weighting of the three components was obtained through extensive testing with human observers. The two chrominance signals are obtained as

$$C_b = B - Y \quad (18.3)$$

$$C_r = R - Y \quad (18.4)$$

These three signals can be used by the color television set to generate the red, blue, and green signals needed to control the electron guns. The luminance signal can be used directly by the black-and-white televisions.

Because the eye is much less sensitive to changes of the chrominance in an image, the chrominance signal does not need to have higher frequency components. Thus, lower bandwidth of the chrominance signals along with a clever use of modulation techniques

permits all three signals to be encoded without need of any bandwidth expansion. (A simple and readable explanation of television systems can be found in [245].)

The early efforts toward digitization of the video signal were devoted to sampling the composite signal, and in the United States the Society of Motion Picture and Television Engineers developed a standard that required sampling the NTSC signal at a little more than 14 million times a second. In Europe, the efforts at standardization of video were centered around the characteristics of the PAL signal. Because of the differences between NTSC and PAL, this would have resulted in different "standards." In the late 1970s, this approach was dropped in favor of sampling the components and the development of a worldwide standard. This standard was developed under the auspices of the International Consultative Committee on Radio (CCIR) and was called CCIR recommendation 601-2. CCIR is now known as ITU-R, and the recommendation is officially known as ITU-R recommendation BT.601-2. However, the standard is generally referred to as recommendation 601 or CCIR 601.

The standard proposes a family of sampling rates based on the sampling frequency of 3.725 MHz (3.725 million samples per second). Multiples of this sampling frequency permit samples on each line to line up vertically, thus generating the rectangular array of pixels necessary for digital processing. Each component can be sampled at an integer multiple of 3.725 MHz, up to a maximum of four times this frequency. The sampling rate is represented as a triple of integers, with the first integer corresponding to the sampling of the luminance component and the remaining two corresponding to the chrominance components. Thus, 4:4:4 sampling means that all components were sampled at 13.5 MHz. The most popular sampling format is the 4:2:2 format, in which the luminance signal is sampled at 13.5 MHz, while the lower-bandwidth chrominance signals are sampled at 6.75 MHz. If we ignore the samples of the portion of the signal that do not correspond to active video, the sampling rate translates to 720 samples per line for the luminance signal and 360 samples per line for the chrominance signal. The sampling format is shown in Figure 18.6. The luminance component of the digital video signal is also denoted by Y , while the chrominance components are denoted by U and V . The sampled analog values are converted to digital values as follows. The sampled values of YC_bC_r are normalized so that the sampled Y values, Y_s , take on values between 0 and 1, and the sampled chrominance values, C_{rs} and C_{bs} , take on values

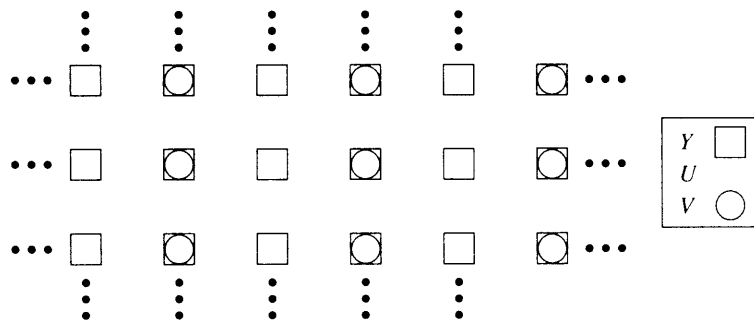


FIGURE 18.6 Recommendation 601 4:2:2 sampling format.

between $-\frac{1}{2}$ and $\frac{1}{2}$. These normalized values are converted to 8-bit numbers according to the transformations

$$Y = 219Y_s + 16 \quad (18.5)$$

$$U = 224C_{bs} + 128 \quad (18.6)$$

$$V = 224C_{rs} + 128. \quad (18.7)$$

Thus, the Y component takes on values between 16 and 235, and the U and V components take on values between 16 and 240.

An example of the Y component of a CCIR 601 frame is shown in Figure 18.7. In the top image we show the fields separately, while in the bottom image the fields have been interlaced. Notice that in the interlaced image the smaller figure looks blurred. This is because the individual moved in the sixtieth of a second between the two fields. (This is also proof—if any was needed—that a three-year-old cannot remain still, even for a sixtieth of a second!)

The YUV data can also be arranged in other formats. In the Common Interchange Format (CIF), which is used for videoconferencing, the luminance of the image is represented by an array of 288×352 pixels, and the two chrominance signals are represented by two arrays consisting of 144×176 pixels. In the QCIF (Quarter CIF) format, we have half the number of pixels in both the rows and columns.

The MPEG-1 algorithm, which was developed for encoding video at rates up to 1.5 Mbits per second, uses a different subsampling of the CCIR 601 format to obtain the MPEG-SIF format. Starting from a 4:2:2, 480-line CCIR 601 format, the vertical resolution is first reduced by taking only the odd field for both the luminance and the chrominance components. The horizontal resolution is then reduced by filtering (to prevent aliasing) and then subsampling by a factor of two in the horizontal direction. This results in 360×240 samples of Y and 180×240 samples each of U and V . The vertical resolution of the chrominance samples is further reduced by filtering and subsampling in the vertical direction by a factor of two to obtain 180×120 samples for each of the chrominance signals. The process is shown in Figure 18.8, and the resulting format is shown in Figure 18.9.

In the following we describe several of the video coding standards in existence today. Our order of description follows the historical development of the standards. As each standard has built upon features of previous standards this seems like a logical plan of attack. As in the case of image compression, most of the standards for video compression are based on the discrete cosine transform (DCT). The standard for teleconferencing applications, ITU-T recommendation H.261, is no exception. Most systems currently in use for videoconferencing use proprietary compression algorithms. However, in order for the equipment from different manufacturers to communicate with each other, these systems also offer the option of using H.261. We will describe the compression algorithm used in the H.261 standard in the next section. We will follow that with a description of the MPEG algorithms used in Video CDs, DVDs and HDTV, and a discussion of the latest joint offering from ITU and MPEG.

We will also describe a new approach towards compression of video for videophone applications called three-dimensional model-based coding. This approach is far from maturity, and our description will be rather cursory. The reason for including it here is the great promise it holds for the future.

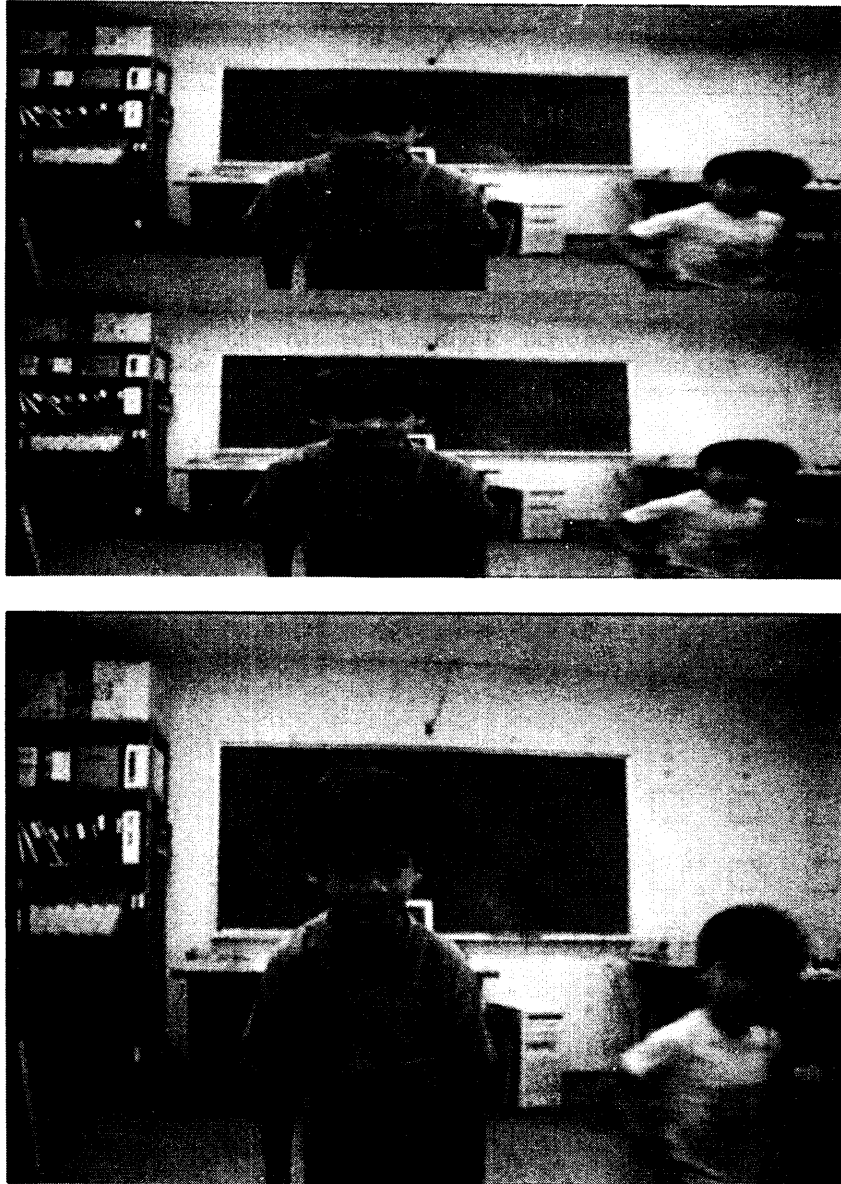


FIGURE 18.7 Top: Fields of a CCIR 601 frame. Bottom: An interlaced CCIR 601 frame.

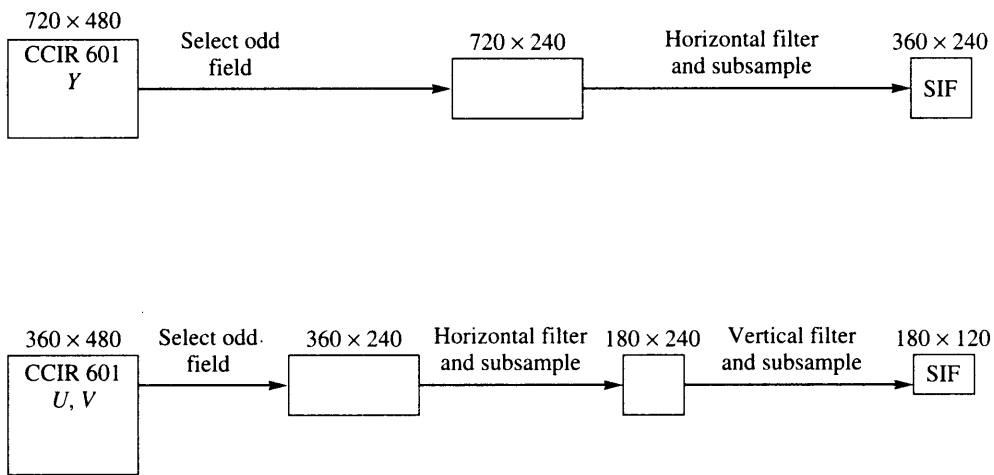


FIGURE 18.8 Generation of an SIF frame.

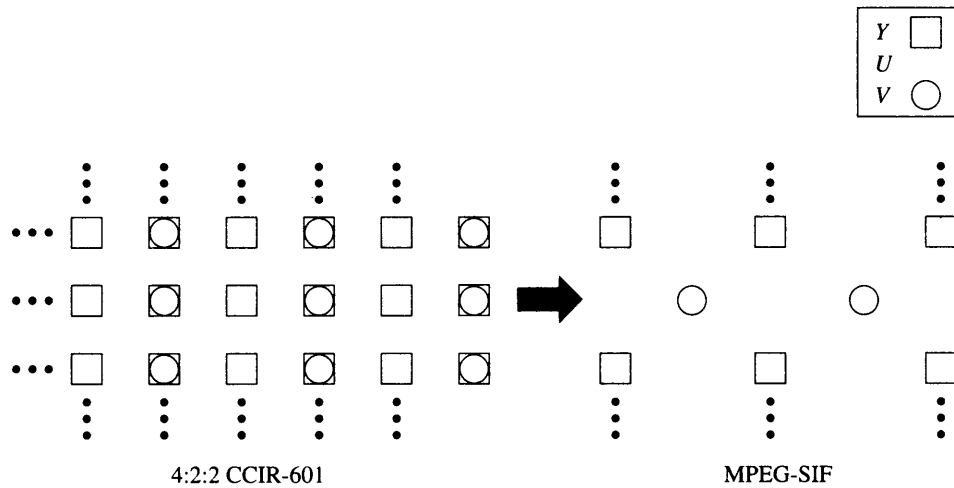


FIGURE 18.9 CCIR 601 to MPEG-SIF.

18.5 ITU-T Recommendation H.261

The earliest DCT-based video coding standard is the ITU-T H.261 standard. This algorithm assumes one of two formats, CIF and QCIF. A block diagram of the H.261 video coder is shown in Figure 18.10. The basic idea is simple. An input image is divided into blocks of 8×8 pixels. For a given 8×8 block, we subtract the prediction generated using the previous frame. (If there is no previous frame or the previous frame is very different from the current frame, the prediction might be zero.) The difference between the block being encoded and

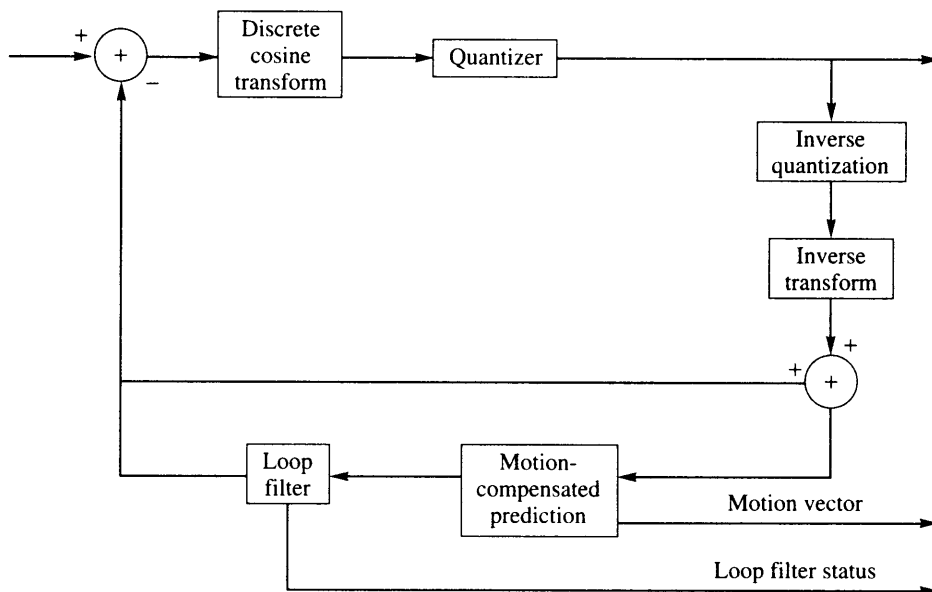


FIGURE 18.10 Block diagram of the ITU-T H.261 encoder.

the prediction is transformed using a DCT. The transform coefficients are quantized and the quantization label encoded using a variable-length code. In the following discussion, we will take a more detailed look at the various components of the compression algorithm.

18.5.1 Motion Compensation

Motion compensation requires a large amount of computation. Consider finding a matching block for an 8×8 block. Each comparison requires taking 64 differences and then computing the sum of the absolute value of the differences. If we assume that the closest block in the previous frame is located within 20 pixels in either the horizontal or vertical direction of the block to be encoded, we need to perform 1681 comparisons. There are several ways we can reduce the total number of computations.

One way is to increase the size of the block. Increasing the size of the block means more computations per comparison. However, it also means that we will have fewer blocks per frame, so the number of times we have to perform the motion compensation will decrease. However, different objects in a frame may be moving in different directions. The drawback to increasing the size of the block is that the probability that a block will contain objects moving in different directions increases with size. Consider the two images in Figure 18.11. If we use blocks that are made up of 2×2 squares, we can find a block that exactly matches the 2×2 block that contains the circle. However, if we increase the size of the block to 4×4 squares, the block that contains the circle also contains the upper part of the octagon. We cannot find a similar 4×4 block in the previous frame. Thus, there is a trade-off involved.

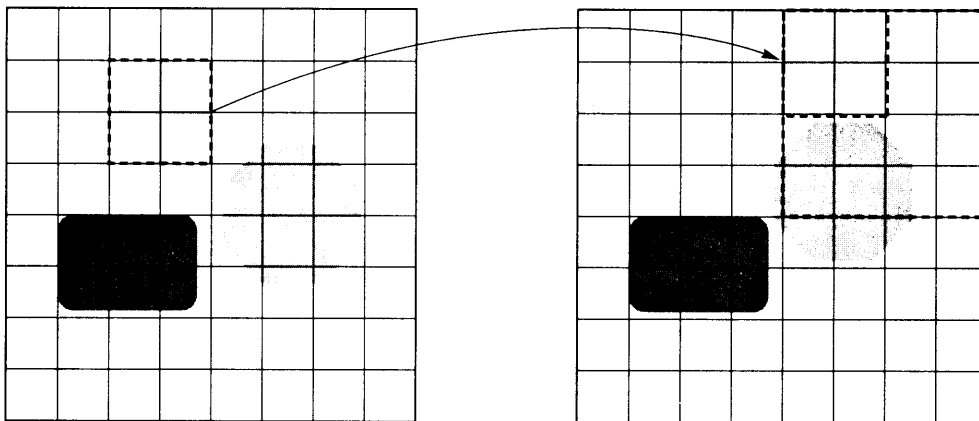


FIGURE 18.11 Effect of block size on motion compensation.

Larger blocks reduce the amount of computation; however, they can also result in poor prediction, which in turn can lead to poor compression performance.

Another way we can reduce the number of computations is by reducing the search space. If we reduce the size of the region in which we search for a match, the number of computations will be reduced. However, reducing the search region also increases the probability of missing a match. Again, we have a trade-off between computation and the amount of compression.

The H.261 standard has balanced the trade-offs in the following manner. The 8×8 blocks of luminance and chrominance pixels are organized into *macroblocks*, which consist of four luminance blocks, and one each of the two types of chrominance blocks. The motion-compensated prediction (or motion compensation) operation is performed on the macroblock level. For each macroblock, we search the previous reconstructed frame for the macroblock that most closely matches the macroblock being encoded. In order to further reduce the amount of computations, only the luminance blocks are considered in this matching operation. The motion vector for the prediction of the chrominance blocks is obtained by halving the component values of the motion vector for the luminance macroblock. Therefore, if the motion vector for the luminance blocks was $(-3, 10)$, then the motion vector for the chrominance blocks would be $(-1, 5)$.

The search area is restricted to ± 15 pixels of the macroblock being encoded in the horizontal and vertical directions. That is, if the upper-left corner pixel of the block being encoded is (x_c, y_c) , and the upper-left corner of the best matching macroblock is (x_p, y_p) , then (x_c, y_c) and (x_p, y_p) have to satisfy the constraints $|x_c - x_p| < 15$ and $|y_c - y_p| < 15$.

18.5.2 The Loop Filter

Sometimes sharp edges in the block used for prediction can result in the generation of sharp changes in the prediction error. This in turn can cause high values for the high-frequency coefficients in the transforms, which can increase the transmission rate. To avoid this, prior

to taking the difference, the prediction block can be smoothed by using a two-dimensional spatial filter. The filter is separable; it can be implemented as a one-dimensional filter that first operates on the rows, then on the columns. The filter coefficients are $\frac{1}{4}, \frac{1}{2}, \frac{1}{4}$, except at block boundaries where one of the filter taps would fall outside the block. To prevent this from happening, the block boundaries remain unchanged by the filtering operation.

Example 18.5.1:

Let's filter the 4×4 block of pixel values shown in Table 18.2 using the filter specified for the H.261 algorithm. From the pixel values we can see that this is a gray square with a white *L* in it. (Recall that small pixel values correspond to darker pixels and large pixel values correspond to lighter pixels, with 0 corresponding to black and 255 corresponding to white.)

TABLE 18.2 Original block of pixels.

| | | | |
|-----|-----|-----|-----|
| 110 | 218 | 116 | 112 |
| 108 | 210 | 110 | 114 |
| 110 | 218 | 210 | 112 |
| 112 | 108 | 110 | 116 |

Let's filter the first row. We leave the first pixel value the same. The second value becomes

$$\frac{1}{4} \times 110 + \frac{1}{2} \times 218 + \frac{1}{4} \times 116 = 165$$

where we have assumed integer division. The third filtered value becomes

$$\frac{1}{4} \times 218 + \frac{1}{2} \times 116 + \frac{1}{4} \times 112 = 140.$$

The final element in the first row of the filtered block remains unchanged. Continuing in this fashion with all four rows, we get the 4×4 block shown in Table 18.3.

TABLE 18.3 After filtering the rows.

| | | | |
|-----|-----|-----|-----|
| 110 | 165 | 140 | 112 |
| 108 | 159 | 135 | 114 |
| 110 | 188 | 187 | 112 |
| 112 | 109 | 111 | 116 |

Now repeat the filtering operation along the columns. The final 4×4 block is shown in Table 18.4. Notice how much more homogeneous this last block is compared to the original

block. This means that it will most likely not introduce any sharp variations in the difference block, and the high-frequency coefficients in the transform will be closer to zero, leading to compression.

TABLE 18.4 Final block.

| | | | |
|-----|-----|-----|-----|
| 110 | 165 | 140 | 112 |
| 108 | 167 | 148 | 113 |
| 110 | 161 | 154 | 113 |
| 112 | 109 | 111 | 116 |

This filter is either switched on or off for each macroblock. The conditions for turning the filter on or off are not specified by the recommendations.

18.5.3 The Transform

The transform operation is performed with a DCT on an 8×8 block of pixels or pixel differences. If the motion compensation operation does not provide a close match, then the transform operation is performed on an 8×8 block of pixels. If the transform operation is performed on a block level, either a block or the difference between the block and its predicted value is quantized and transmitted to the receiver. The receiver performs the inverse operations to reconstruct the image. The receiver operation is also simulated at the transmitter, where the reconstructed images are obtained and stored in a frame store. The encoder is said to be in *intra* mode if it operates directly on the input image without the use of motion compensation. Otherwise, it is said to be in *inter* mode.

18.5.4 Quantization and Coding

Depending on how good or poor the prediction is, we can get a wide variation in the characteristics of the coefficients that are to be quantized. In the case of an intra block, the DC coefficients will take on much larger values than the other coefficients. Where there is little motion from frame to frame, the difference between the block being encoded and the prediction will be small, leading to small values for the coefficients.

In order to deal with this wide variation, we need a quantization strategy that can be rapidly adapted to the current situation. The H.261 algorithm does this by switching between 32 different quantizers, possibly from one macroblock to the next. One quantizer is reserved for the intra DC coefficient, while the remaining 31 quantizers are used for the other coefficients. The intra DC quantizer is a uniform midrise quantizer with a step size of 8. The other quantizers are midtread quantizers with a step size of an even value between 2 and 62. Given a particular block of coefficients, if we use a quantizer with smaller step size, we are likely to get a larger number of nonzero coefficients. Because of the manner in which the labels are encoded, the number of bits that will need to be transmitted will increase. Therefore, the availability of transmission resources will have a major impact on the quantizer selection. We will discuss this aspect further when we talk about the transmission