The process of signal analysis and modeling described in the previous chapter results in a compact formulation of the information-bearing portions of the signal. This compact representation can be used to compress the signal to allow its transmission over limited-bandwidth channels or its storage within limited space.

This section discusses the real time implementation of signal compression, also known as coding. Various coding schemes are distinguished by several properties:

- **Compression ratio**: the amount of compression achieved, determined by the ratio of the size of the original signal to the size of the compressed signal;
- **Reconstruction quality**: some compression schemes are lossless, providing a reconstruction waveform that exactly matches, sample for sample, the original signal. Other methods achieve a higher compression ratio through lossy compression, which does not allow exact reconstruction of the waveform, but instead seeks to preserve its information-bearing portions;
- **Fixed versus variable transmission bit rate**: bit rate can vary in some schemes that are based on encoding the rate of change of the properties of the signal—a signal that is relatively stable, such as a sustained single-frequency sine wave, will require fewer bits per second than a speech signal;
9.1 Speech Coding

Speech coders fall into one of two classes. *Waveform coders* generate a reconstructed waveform (after coding, transmission, and decoding) that approximates the original waveform, thereby approximating the original speech sounds that carry the message. *Voice coders*, or vocoders, do not attempt to reproduce the waveform, but instead seek to approximate the speech-related parameters that characterize the individual segments of the waveform. Speech coding systems usually operate either within telephone bandwidth (200 Hz to 3.2 KHz) or wideband (up to 7 KHz, used in AM radio-commentary audio, multimedia, etc.).

9.1.1 Waveform Coders

The simplest waveform coder is pulse code modulation, or PCM. As shown in Fig. 9–1, the waveform is passed through a low pass filter to remove high-frequency components and then is passed through a sampler. The sampler per-

![Diagram](image-url)  
Figure 9–1 Pulse code modulation represents analog signal by low pass filtering, sample-and-hold, and analog-to-digital conversion.
forms a sample-and-hold operation by capturing the instantaneous value of the waveform at each sampling instant and holding it at that value, resulting in a stair step pattern. During the hold interval, an analog-to-digital converter computes and outputs a digital representation of the current analog value.

The sampling rate is at a frequency $f_s$, and each sample is represented by a $B$-bit word. The sampling frequency is set by the bandwidth of the low pass-filtered signal, $W$. This relationship is set by the Nyquist criterion, which requires a minimum of two samples to determine a frequency, so $f_s = 2W$.

Successive samples may be similar in a PCM system, especially when the bandwidth is well below $f_s/2$. This sample-to-sample correlation may be exploited to reduce bit rate by predicting each sample based on previous sample values, comparing the predicted sample value to the actual sample value, and encoding the difference between the two. This difference is usually smaller than either sample, so fewer bits are needed to encode it accurately. Extending the predictor to include the weighted sum of the previous $p$ samples can extend this method, known as differential PCM or DPCM, to improve the prediction of a sample:

$$\tilde{x}(n) = \sum_{i=1}^{p} a_i \tilde{x}(n - 1).$$

(9.1)

The difference $e(n)$ between the predicted sample $\tilde{x}(n)$ and the actual sample $x(n)$ is given by

$$e(n) = x(n) - \tilde{x}(n),$$

(9.2)

where $\tilde{x}(n - i)$ is the encoded and decoded $(n-i)$th sample. The quantizer produces $\hat{e}(n)$, which is the quantized version of $e(n)$. Fig. 9–2 shows the structure of a DPCM coder and the quantized prediction error, $\hat{e}(n)$, that is output.1

The quantizer and predictor may be adapted over time to follow the time-varying properties of the signal and adapt the use of bits to the signal. Adaptive DPCM (ADPCM) performs such adaptation. An adaptive predictor changes the number of bits, based on one or more of the following signal properties:

![Figure 9–2 DPCM coder.](image-url)
• probability density function (histogram of values) of the signal;
• mean value of input signals;
• dynamic range (variance from mean).

The ADPCM predictor is thus based on a shorter-term average of signal statistics than the long-term average. Adaptation may be performed in the forward direction, backward direction, or both directions. In forward adaptation, a set of \( N \) values is accumulated in a buffer and a set of \( p \) predictor coefficients is computed. This buffering induces a delay that corresponds to the acquisition time of \( M \) samples. For speech signals, this is typically 10 msec. The delay is compounded for telephone-routing paths that perform multiple stages of encode/decode. Backward adaptation uses both quantized and transmitted data to perform prediction, thereby reducing the delay.

An example of ADPCM implementation\(^2\) uses a backward adaptive algorithm to set the quantizer size. It uses a fixed first-order predictor and a robust adapting step size. It is implemented on a low-cost fixed point digital signal processor with a \( B \)-bit fixed uniform quantizer [shown at point (1) in Fig. 9–3]. A pair of tables (2) stores the step size and its inverse for adaptive scaling of the signal before and after quantization. A step-size adaptation loop (3) generates table addresses.

![Figure 9–3](image) ADPCM coder implemented on fixed-point digital signal processing chip\(^2\) consists of (1) fixed uniform quantizer, (2) tables to store step size and inverted step size, (3) step-size adaptation loop, and (4) fixed-predictor loop. © 1982 IEEE, adapted with permission.
The ADPCM fixed predictor loop (4) generates a predictor signal $\hat{x}(n)$ by multiplying the previously encoded-and-decoded sample $\hat{x}(n - 1)$ by the predictor coefficient $a$ and subtracting it from the current sample $x(n)$, forming the difference signal $e(n)$:

$$e(n) = x(n) - \hat{x}(n).$$  \hspace{1cm} (9.3)

The difference signal $e(n)$ is then adaptively quantized.

The step size used to quantize $e(n)$ is a function of the amplitude of $e(n)$. Using a prediction loop applied to $e(n)$ sets the step size. The step size and inverse step size are chosen using locked pointers, the placement of which is set by the weighted prediction error of $e(n)$.

To reconstitute the signal, the $b$-bit integer $I(n)$ resulting from the quantizer (1) is scaled by the step size and the predicted sample $\hat{x}(n)$ is added back.

Table 9–1 compares the features of waveform coding methods used for speech signals. In the table, “toll quality” refers to a quality level consistent with the best-quality 3.2-KHz bandwidth telephone speech, transmitted at 64 Kbit/sec and encoded with μ-law-companded PCM. Communication quality is less than toll quality, but preserves the characteristics that allow identifying the talker.

The above coders use a single $B$-bit quantizer, forming $2^B$ quantization levels, for digitizing all amplitude samples. Another approach decomposes the signal into a set of components, each of which is separately digitized with possibly different sampling rates. The signal is then reconstructed during decoding by summing these components. One method to separate a signal is by frequency. A multifrequency decomposition has the advantage of allowing control of the quantization error for each frequency band, based on the number of bits allocated to each band. As a result, the overall spectrum of the error signal can be shaped to allow placing most of the error spectrum within frequency areas that are less easily perceived. The error spectrum is thus shaped to complement the human perception of error.

An example of multifrequency decomposition is subband coding. Like the discrete wavelet transform discussed in the previous section, subband coding provides successive approximations to the waveform by recursively decomposing it into a low-frequency and high-frequency portion. Subband coding divides the input channel into multiple frequency bands and codes each band with ADPCM. The quadrature mirror filter described in connection with the wavelet transform

<table>
<thead>
<tr>
<th>Method</th>
<th>Bit Rate</th>
<th>Quality</th>
<th>Relative Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCM</td>
<td>64 kb/sec</td>
<td>toll</td>
<td>low</td>
</tr>
<tr>
<td>DPCM</td>
<td>24 kb/sec–32 kb/sec</td>
<td>communications</td>
<td>med</td>
</tr>
<tr>
<td>ADPCM</td>
<td>32 kb/sec</td>
<td>toll</td>
<td>high</td>
</tr>
</tbody>
</table>
was originally applied to subband coding, and it can be implemented on a pro-
gammable signal processor as was shown in Chapter 8. The ADPCM implementa-
tion just described may be used for the ADPCM portion of a subband coder.

9.1.2 Voice Coders

In contrast to the waveform coder, the voice coder seeks to preserve the informa-
tion and speech content without trying to match the waveform. A voice coder
uses a model of speech production (Fig. 9–4).

One model consists of a linear predictive coding (LPC) representation of the
vocal tract. The input speech signal is then filtered with the inverse of the vocal
tract filter. Because the filter is not exact, a residual signal is obtained at the out-
put of the inverse filter. This residual is regarded as the excitation signal for the
filter. A characterization of its periodicity is made, and if strongly periodic, the
section of speech is declared as voiced with the measured pitch period. If not,
the speech sound is declared unvoiced and the excitation is modeled with random
noise. In either case, the overall amplitude of the excitation is also measured to
preserve amplitude characteristics.

The LPC analysis and its mapping to real time processing were discussed in
Sections 6.4 and 8.4.2. To characterize the excitation, a pitch-period estimator,
based on the autocorrelation analysis of the signal, can be implemented. In this
pitch-period estimator, the speech signal is first low-pass filtered to remove en-
ergy above the highest likely pitch period ( > 800 Hz). Next, the unnormalized
autocorrelation is computed for lag values $m$, $m_{\text{min}} \leq m \leq m_{\text{max}}$. Here $m_{\text{min}}$ and $m_{\text{max}}$
are sample lags based on the minimum and maximum expected pitch period. The
computation is performed on a windowed version of the speech, where $w(n)$ is the
window:

$$r_n(m) = \sum_{l=-\infty}^{\infty} w(n - l)x(l - m); m_{\text{min}} \leq m \leq m_{\text{max}}.$$  \hspace{1cm} (9.4)

The pitch period is declared to be the lag $m_0$ over the range of allowed pitch
periods $m_{\text{min}} \leq m \leq m_{\text{max}}$ for which $r_n(m)$ is a maximum.

Figure 9–4  Voice coder model of speech production uses a vocal tract
representation that is excited by either a voiced or unvoiced
signal.
To place the pitch period estimation in a form suited for real-time implementation, the algorithm is cast into a stream-processing form. An exponential function is used for windowing:

$$w(n) = \begin{cases} \gamma^n; & n \geq 0 \\ 0; & n < 0 \end{cases} \quad (9.5)$$

The autocorrelations are computed on a sample-by-sample basis for each $m$:

$$r_n(m) = \gamma r_{n-1}(m) + x(n) \cdot x(n - m). \quad (9.6)$$

The computation load is reduced by updating the autocorrelation only every $j$th sample (e.g., $j = 4$):

$$r_n(m) = \gamma' r_{n-j}(m) + x(m) \cdot x(n - m). \quad (9.7)$$

A value of $\gamma' = 0.95$ is typical. The range of lags $m_{\text{min}} \leq m \leq m_{\text{max}}$ is distributed over $j$ samples, performing $1/j$th of the $m_{\text{max}} - m_{\text{min}} + 1$ values. The pitch is quantized to 6 bits, or 64 values, which are distributed over a range from 66.7 Hz to 320 Hz. Fig. 9–5 shows a structure for real time implementation of the pitch period estimation algorithm. A lowpass filter (LPF) is followed by a $j$-fold downsampling, and every $j$th sample is entered into a shift register of length $m_{\text{max}} - m_{\text{min}} + 1$ locations. The shift register is used in the autocorrelation update of $r_n(m)$, and the results are written in an autocorrelation buffer. The buffer is scanned for a peak value, and the pitch period associated with the lag of the peak is output.

A standardized version of an LPC vocoder has been used in the implementation of the Secure Telephone Unit Version 3, or STU-III. The STU-III is combined with an encryption system and error protection processing to implement a secure

![Figure 9–5](image-url)  Computational structure for computing pitch period estimate by autocorrelation analysis. © 1982 IEEE, adapted with permission.
voice communication link. In its original version, an enhanced 10th-order LPC analysis known as LPC-10e was used. In LPC-10e, the excitation signal is categorized as voiced or unvoiced, with pitch period transmitted for voiced and gain (energy) encoded for both voiced and unvoiced speech. The analysis properties of Table 9–2 are used for LPC-10e, and its coding and low bit rate render a buzzy quality to the speech that masks most characteristics of talker identification.

Computational blocks for LPC-10e include the LPC analysis, pitch and voiced/unvoiced decision, encoding, and communication processing. Pitch detection is performed by the Average Magnitude Difference Method (AMDF), which avoids the multiplications needed by the autocorrelation method. The AMDF subtracts a delayed version of the waveform from the incoming waveform at various lags and averages the differences across samples. The lag that produces the smallest difference is selected as the pitch period estimate. An 800-Hz low pass filter is applied, and 60 possible pitch values are accommodated. These values are not uniformly spaced, but follow a sequence of lags given by \{20, 21, \ldots, 39, 40, 42, \ldots, 78, 80, 84, \ldots, 156\}, corresponding to pitches at the 8-KHz sampling rate that range from 51.3 Hz to 400 Hz.

In addition to LPC analysis, and pitch-period estimation, the LPC-10e algorithm requires parameter encoding and “communication” processing. Parameter encoding, described by an example below (Table 9–3), assigns particular bit locations for the various LPC, pitch, and other parameters that are transmitted. In the transmit mode, communication processing includes parallel to serial conversion and forward error correction. In the receive mode, it includes initial acquisition of synchronization, frame-to-frame maintenance of synchronization, de-interleaving of frame information, serial-to-parallel conversion, error correction, and parameter decoding.

The LPC-10e algorithm was developed to run on a bit-sliced 16-bit computer with a dedicated multiplier, at a time when single-chip digital signal processors were a rarity. On a programmable signal processor, the block form of

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sampling rate</td>
<td>8 KHz</td>
</tr>
<tr>
<td>Frame period</td>
<td>22.5 msec</td>
</tr>
<tr>
<td>Speech samples/frame</td>
<td>180</td>
</tr>
<tr>
<td>Output bits/frame</td>
<td>54</td>
</tr>
<tr>
<td>Bit rate</td>
<td>2.4 Kb/sec</td>
</tr>
<tr>
<td>Bits per sample (average)</td>
<td>0.3</td>
</tr>
<tr>
<td>Compression factor (average)</td>
<td>30</td>
</tr>
<tr>
<td>LPC analysis method</td>
<td>Covariance</td>
</tr>
<tr>
<td>Transmission format</td>
<td>Serial</td>
</tr>
</tbody>
</table>
covariance-based LPC analysis requires a relatively large RAM. One implementation uses a standard microprocessor and three fixed-point digital signal processors to implement an LPC-10e encoder/decoder pair. The partitioning is shown in Fig. 9–6 and assigns non-repetitive operations such as voiced/unvoiced decision, pitch tracking, coefficient encoding, and synchronization to the microprocessor. The signal processors perform such repetitive tasks as LPC analysis, pitch period estimation, and LPC synthesis.

In an alternative implementation, custom-integrated circuits implement the LPC analysis, synthesis, and AMDF pitch analysis, and three microprocessors complete the pitch-period estimation, control the gain, perform error correcting, and format the coefficients (Fig. 9–7).

The efficient encoding of LPC and pitch parameters for transmission is exemplified by the method used in the speech synthesizer of the commercial learning aid known as “Speak and Spell,” developed by Texas Instruments. The encoded parameters consist of the frame energy, pitch (which is set to 0 to indicate unvoiced speech), and 10 LPC-derived reflection coefficients (Table 9–3). A frame period of 25 msec provides a rate of 40 frames/sec, and a bit rate of 1200 bits/sec is achieved. For the first voiced frames, 49 bits are used as shown in Table 9–3; a separate code for “repeat” transmits subsequent frames with identical LPC parameters (pitch and energy of repeated frames can vary) at 10 bits each. Unvoiced frames are transmitted at 28 bits each.

The quality of speech coding can be improved by allowing more flexibility in modeling the prediction residual than permitted by the binary choice of voiced/unvoiced. These two states can be blurred into a continuum for each analysis frame by exciting the computed LPC synthesis filter with a variety of candidate excitation functions, comparing the synthesized speech to the original

---

**Figure 9–6** Implementation of real-time LPC-10 encoder/decoder uses three first-generation digital signal processing chips (NEC μPD 7720) and a microprocessor CPU.
waveform within the encoder, and picking the excitation function that minimizes the difference between the re-synthesized and original speech, using a distance measure based on human perception. This selection of an excitation function replaces both the voiced/unvoiced decision and the pitch period excitation.

For example, code-excited linear prediction (CELP) uses vector quantization (VQ), by which a predetermined set of excitation signals is stored in a codebook. For each frame, the codebook is searched for the particular excitation sequence that, upon recreating a speech waveform through the synthesis filter, minimizes a perceptually weighted distance.

**Figure 9–7** Custom-integrated circuit implementation of LPC analysis and synthesis, augmented by three microprocessors, for real time implementation of LPC-10 speech encoder/decoder.

Table 9–3 Encoding method to achieve 1200 bit/sec average rate for LPC10 parameters $E$ (energy), $P$ (pitch), $K(n)$ ($n$th LPC-based reflection coefficient), $R$ (repeat flag). © 1982 IEEE, adapted with permission.

<table>
<thead>
<tr>
<th>Frame Type</th>
<th>How Determined</th>
<th># Bits/Frame</th>
<th>Parameters Sent (# bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voiced</td>
<td>$E \neq 15; P \neq 0; R = 0$</td>
<td>49</td>
<td>$E(4), P(5), R(1), K1(5), K2(5), K3(4), K4(4), K5(4), K6(4), K7(4), K8(3), K9(3), K10(3)$</td>
</tr>
<tr>
<td>Unvoiced</td>
<td>$E \neq 0$ or 15; $P = 0$; $R = 0$</td>
<td>28</td>
<td>$E(4), P(5), R(1), K1(5), K2(5)$</td>
</tr>
<tr>
<td>Repeated</td>
<td>$E \neq 0$ or 15; $R = 1$</td>
<td>10</td>
<td>$E(4), P(5), R(1)$</td>
</tr>
<tr>
<td>Zero Energy</td>
<td>$E = 0$</td>
<td>4</td>
<td>$E(4)$</td>
</tr>
<tr>
<td>End of Word</td>
<td>$E = 15$</td>
<td>4</td>
<td>$E(4)$</td>
</tr>
</tbody>
</table>
A CELP coder consists of an LPC filter and a VQ excitation section, which in turn includes a computation of the distance metric and a codebook search mechanism. Fig. 9–8 shows the reconstruction of the speech waveform, its comparison with the original, and the creation of a perceptually weighted filter controlled by both the LPC synthesis parameters $a_k$ and a frequency-weighted perceptual weight $\alpha$ that depends on the sampling frequency $f_s$.

Two techniques are used for compiling the codebook of possible excitation waveforms for CELP. The first, stochastic excitation, assumes that the best excitation sequence cannot be predicted on the basis of such simplifications as pitch or voiced/unvoiced categories. Instead, each entry is a different sequence of random numbers. However, a stochastic codebook has no intrinsic order and is thus difficult to search. Several alternatives that ease the search include sparse-excited codebooks, which contain a large number of zeros; lattice-based codebooks, which have regularly spaced arrays of points; trained codebooks, which are built up from clustering a large number of previously-gathered excitation sequences (as will be described below); and multiple codebooks, which consist of both a stochastic and an adaptive codebook.

An adaptive codebook uses the set of excitation samples from the previous frame and performs a search for the optimal time lag at which to present them in the current frame. After this excitation, a stochastic codebook is then searched to minimize the perpetual difference between original and resynthesized waveforms, and this stochastic entry is added to the lag-adjusted excitation and the sum is used to represent frame excitation.

Code-excited linear predictive coding is used in the U.S. Standard 1016 for the newer-generation STU-III, which operates at 4,800 bits/sec. The new STU-III standard CELP uses 10th-order LPC, both stochastic and adaptive codebooks, pitch prediction, and post filtering to reduce speech buzziness.

**Figure 9–8** Computation of perceptual distance metric versus excitation source for code-excited linear prediction (CELP).
9.2 Vector Quantization

Vector quantization has been mentioned as a method used to generate and select possible excitation waveforms for code-excited linear prediction. It is more widely used, extending to both speech and image compression. To understand vector quantization in its more general application, one can envision a signal that generates samples of $B$ bits each at a rate of $R$ samples/sec. Because the number of possible values from $B$ bits is $2^B$, each sample may be regarded as a symbol taken from a dictionary (or “alphabet”) of $2^B$ elements, and for $R$ such samples/sec, a bit rate of $BR$ bits/sec results. Not all symbols occur with equal probability—for example, samples of maximum amplitude are usually less likely to occur than samples near zero magnitude. A method of lossless compression known as entropy coding assigns short indices to the highest-probability symbols and longer indices to lower-probability symbols. To further increase compression, a lossy method may be introduced that reduces the number of alphabet, or codebook, entries to a number less than $2^B$ by concentrating the smaller number of available symbols on values that are likely to occur. A large amount of collected data, used for training, is placed in the feature space, allowing a distance between two samples to be defined. The distance measure is used to define clusters among the data. As shown in Fig. 9–9, a codebook entry is placed at the centroid of each cluster. Each actual data value is replaced by its nearest codebook entry, introducing some distortion. The codebook-generation algorithm, described below, minimizes the total amount of distortion.

A codebook of $J$ codewords requires $\log_2 J$ bits to transmit each codebook index. If the number of entries in the code book is less than the number of values available with $B$ bits ($J < 2^B$), then a compression factor of $B/\log_2 J$ results. Sending the index of the nearest of $J$ codebook entries instead of the exact value reduces the data rate, but at the expense of increased distortion.

9.2.1 Vector Encoding/Decoding

Vector quantization can be applied to time-domain or image signals directly, but more recently and effectively, it has been applied to the residual after the signal passes through a matched inverse filter. As with CELP, the signal is encoded by sending the filter parameters and codebook index of the model.

More precisely, for a feature vector $v$, a codebook consisting of $J$ codewords, $\{w(i), 1 \leq i \leq J\}$ and a distance function $d[v, w(i)]$ defined between feature vector $v$ and codeword $w(i)$, vector quantization finds the particular codeword index $i^*$ that is associated with the codeword $w(i)$ that is the minimum distance from $v$:

$$
\hat{i} = \arg \min_{1 \leq i \leq J} d[v, w(i)].
$$

Then instead of transmitting feature vector $v$, the index $i^*$ of the best-matching codeword is sent. Fig. 9–10 shows a flowchart of the vector-quantization coding operations for each input vector $v$. 
To meet the requirements of real time encoding, the feature vectors must be encoded as fast as they arrive. For a full-search encoding algorithm, \( J \) comparisons, one against each codebook entry, are required every feature vector period. Each comparison must access and examine each of the values in each feature vector. The computation is regular, but it requires high throughput.

Parameters that define a particular instance of vector quantization and impact real time requirements are:

\[
\begin{align*}
J &= \text{number of codebook vectors;} \\
\&= \text{number of codebook vectors;} \\
\&= \text{number of codebook vectors;} \\
\&= \text{number of codebook vectors;} \\
\&= \text{number of codebook vectors;} \\
\&= \text{number of codebook vectors;}
\end{align*}
\]

\[
\begin{align*}
d &= \text{type of local distance selected (e.g., sum of products, sum of difference, ratio,...);}
\end{align*}
\]
9.2 Vector Quantization

\[ P = \text{number of features per distance comparison}; \]
\[ \tau = \text{frame period}; \frac{1}{\tau} = \text{number of frames/sec}; \]
\[ l = \text{number of codebook indices submitted per frame (for images, image is divided into } l \text{ multiple subblocks with one index submitted per subblock; for speech, } l = 1); \]
\[ \text{Method of codebook search (full, tree, trellis, \ldots)}. \]

For an on-line, dynamically adapted codebook as described below, additional parameters influence the throughput:

\[ k = \text{size of training set}; \]
\[ F = \text{frequency of adaptation}. \]

Table 9–4 provides an estimate of codebook throughput for a codebook size \( J \) of 1024 = \( 2^{10} \).

The computation of vector quantization encoding can be partitioned to a linear array. Each processor is assigned one codebook entry; \( J \) processors cover the entire codebook. The vector to be encoded, \( \mathbf{v}(n) \), enters each codebook processor as shown in Fig. 9–11. An initial value of \( d_{\text{min}} = \infty \) is inserted at the left-hand entry point into the array. Each processor computes \( d[\mathbf{v}(n), \mathbf{w}(i)] \) and outputs

\[
\begin{align*}
    &d_{\text{min}} = \infty \\
&i^* = 0 \\
&i = 0 \\
&\quad \text{evaluate } d[\mathbf{v}(n), \mathbf{w}(i)] \\
&i = i + 1 \\
&\quad \text{if } i > J \text{ then stop} \\
&\quad \text{else } \quad \text{if } d_{\text{min}} > d[\mathbf{v}(n), \mathbf{w}(i)] \text{ then } \\
&\quad \quad \quad d_{\text{min}} = d[\mathbf{v}(n), \mathbf{w}(i)] \\
\end{align*}
\]

Figure 9–10 Flowchart for vector quantization encoding of feature vector \( \mathbf{v}(n) \) by \( J \)-element codebook \( \mathbf{w}(i), 0 \leq i \leq J \). © IEEE 1995, adapted with permission.
min($d_{\text{min}}$, $d$) to its right-hand neighboring processors as well as its corresponding value of $i$. From the right side of the array emerges $v(n)$, $d_{\text{min}}$ and $i^*$, which is the index of the best-matching code word. Multiple processors allow pipeline computation—while $v(n)$ is being compared to $w(n)$ in processor $i$, $v(n + 1)$ is being compared to $w(i - 1)$ in processor $i - 1$. This pipeline provides a $J$-fold speedup.

### 9.2.2 Codebook Generation

The codebook itself may either be produced offline or may be adapted or regenerated in real time. If the codebook is adapted in real time, its changes must be communicated to the receiving end (in addition to the message encoded with the current codebook), thereby introducing a tradeoff of total bit rate and adaptation rate. A commonly-used codebook generation algorithm is the one proposed by Y. Linde, A. Buzo, and R. M. Gray known as the LBG algorithm. The algorithm begins with an initial codebook of $J$ vectors, which may be generated in several ways:

- random set of $J$ training vectors;
- $J$ vectors that uniformly sample the feature space;
- previous codebook (especially for an adaptive system).

![Figure 9–11](image_url) Array of $J$ processors, each assigned to one of $J$ codebook entries, for real-time vector quantization. © 1995 IEEE.
Codebook generation proceeds according to the following steps:

1. Initialize.
2. For each of the $K$ training vectors, find the closest codebook vector (this requires computing the distance of the training vector from each codebook vector and selecting the minimum).
3. Add the distance between the training vector and its closest codebook neighbor to an accumulating sum of overall distance.
4. After assigning each training vector to a codebook value, replace that codebook value with a vector computed as the centroid of the set of training vectors that are closest to it.
5. Compare the total distance (across all pairs of training vectors and the nearest codebook entry of each) with the total distance from the previous iteration—if the change is less than a present convergence criterion, stop; else go to 2.

Fig. 9–12 represents the process graphically by showing a set of training values as $\bullet$, a set of codebook vectors (o), and the association of a set of training vectors.

Figure 9–12  Vector quantization codebook generation via the LBG algorithm: Each training vector ($\bullet$) is associated with its nearest codebook vector (o), where the large dashed circle shows association; the next training iteration is begun by moving each codebook vector to the centroid of the training vectors that were mapped to it, shown by the dotted small circle. This movement may change the association of a borderline training vector to another codebook vector.
vectors with a codebook vector by a dashed circle around the training vectors associated with a codebook vector. The arrow shows the movement of the codebook vector upon the next iteration to the centroid of its training set neighbors that were mapped to it. This movement may bring new training vectors into the neighborhood of the newly placed codebook entry, indicated by the dotted hollow circle.

The LBG training algorithm may be described in pseudocode, tailored for high-speed implementation on parallel processors (Fig. 9–13). Specifically, a stream processing adaptation of the updated centroid location is implemented in which each relevant training vector is added into \( \mathbf{w}_{\text{new}}(i) \), and after all training vectors have been assigned, a division by the number of training vectors assigned to the codebook vector is performed.

![Figure 9–13](image)

**Figure 9–13** Pseudocode listing of LBG algorithm for realtime implementation. © 1995 IEEE, adapted with permission.
For a high-speed implementation of training, the parallel VQ encoding array of Fig. 9–11 can be augmented to allow execution of the LBG algorithm by adding a second processor array that receives the training vectors closest to each codebook entry and computes a new codebook vector. In Fig. 9–14, a dotted arrow between \( w(i) \) and \( w_{\text{new}}(i) \) indicates a transfer for each training iteration of \( K \) samples.

### 9.3 IMAGE COMPRESSION

Two types of image compression, or coding, are discussed here. Single-image frame coding compresses a still picture, while video coding is built up from single-frame coding by adding interframe coding techniques to compress the image sequence that makes up a video stream. Methods for single-frame coding include the discrete cosine transform (DCT), described below, and subband (or wavelet) encoding described in Section 8.3. Interframe coding supplements single-frame coding techniques with motion estimation, using search methods from frame to frame to predict and encode object motion.

#### 9.3.1 Single-Frame Coding Methods

The discrete cosine transform (DCT) is an important element in image coding. It is performed on a two-dimensional image by acting upon subblocks of adjacent pixels within the image. For example, a 512 \( \times \) 512 pixel image may be broken up into an array of 8 \( \times \) 8 pixel blocks and a DCT may be performed on each block. The DCT packs most of the energy of image data into a few coefficients. It is approximately equal to a 2\( N \)-point FFT of a reflected version of the signal sequence.
concatenated with the N-point sequence itself, and it exploits even symmetry and the restriction of image data values to the real domain.

The DCT may be compared to the FFT on a one-dimensional sequence (Fig. 9–15).10 The FFT operates on a waveform segment formed by a finite-duration analysis window, and the waveform behaves as if it were periodically extended beyond the analysis frame. At the point of extension, the signal experiences a discontinuity ("glitch") as the low-amplitude windowed tail is abutted to the full-amplitude center of the next analysis window. The glitch at this joint introduces high-frequency components into the Fourier spectrum. Alternatively, the DCT causes the waveform to behave as if it were first reflected and then periodically extended, such that the low-amplitude tail of one analysis frame is abutted to the low-amplitude head of the next in a smoother transition. The DCT spectrum does not contain the high-frequency components introduced by the periodic extension of the FFT.

The one-dimensional DCT of a function \( x(n) \) is given by:

\[
\text{DCT: } Y(k) = \sum_{n=0}^{N-1} x(n) \cos\left(\frac{\pi}{N} k \left( n + \frac{1}{2} \right) \right) \tag{9.9}
\]

\[
\text{Inverse DCT: } x(n) = \frac{2}{N} \sum_{k=0}^{N-1} Y(k) \cos\left(\frac{\pi}{N} k \left( n + \frac{1}{2} \right) \right). \tag{9.10}
\]

Similarly, the two-dimensional DCT is given by:

\[
\text{DCT: } Y(k,l) = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x(n,m) \cos\left(\frac{\pi}{N} k \left( n + \frac{1}{2} \right) \right) \cos\left(\frac{\pi}{M} l \left( m + \frac{1}{2} \right) \right) \tag{9.11}
\]

Figure 9–15 For an original waveform spectrum (a), as contrasted with the FFT (b), the DCT periodically extends a reflected version of the signal (c), reducing the high-frequency component resulting from the glitch when the extended signal meets the original in the FFT.10
The two-dimensional DCT can be computed by row/column decomposition into one-dimensional DCTs:

\[ Y(k,l) = \sum_{m=0}^{M-1} \left\{ \sum_{n=0}^{N-1} x(n,m) \cos \left( \frac{\pi}{N} k \left( n + \frac{1}{2} \right) \right) \cos \left( \frac{\pi}{M} l \left( m + \frac{1}{2} \right) \right) \right\} \]

(9.13)

N-point DCT of columns

M-point DCT of rows

Inverse DCT: \( x(n,m) = \frac{4}{NM} \sum_{k=0}^{N-1} \sum_{l=0}^{M-1} Y(k,l) \cos \left( \frac{\pi}{N} k \left( n + \frac{1}{2} \right) \right) \cos \left( \frac{\pi}{M} l \left( m + \frac{1}{2} \right) \right) \).

(9.12)

A thorough review and comparison of real-time implementations of the DCT has identified four types of architectural approaches\(^1\)11: Direct, separate rows and columns, fast transform, and distributed arithmetic. Each will now be discussed in turn.

The direct method of DCT implementation uses the formula of (9.11) directly. For an image block size of \( L \times L \) pixels, it requires \( L^4 \) multiplications and \( L^4 \) adds (alternatively, \( L^2 \) multiplies and \( L^2 \) adds per pixel). The direct implementation of processing DCT blocks can be mapped to an array of processing elements (PEs). An image of \( N \times M \) pixels is divided into \( L \times L \) blocks. There are no data dependencies across blocks, so a separate DCT processor may be devoted to each block.

A separable implementation of the DCT performs \( L \) one-dimensional DCTs on the rows and \( L \) one-dimensional DCTs on the resulting columns. This requires \( 2L^3 \) multiplications and \( 2L^3 \) additions (2\( L \) of each per pixel). To avoid the need to rearrange the coefficient memory and processors between the row and column

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure.png}
\caption{Two arrays of processing elements (PE), for rows and columns, interspersed with a corner turning memory for real-time DCT.\(^\text{11} \) © 1995 IEEE.}
\end{figure}
computations, a corner-turn memory is interposed between the row and column processors that switches rows with columns. The corner turn memory was described in Section 6.6.4 in the example of synthetic aperture radar image formation. Fig. 9–16 shows two linear arrays of processor elements with the corner turn memory in between. It pipelines successive frames into rows and columns.

The DCT, like the Fourier transform, can be cast into a fast form by decomposing it into smaller DCTs. Such transformation reduces the computation on an \( L \times L \) block from \( 2L^4 \) operations to \( 2L^2 \log_2 L \). This proceeds as follows. The one-dimensional DCT can be written in matrix form:

\[
[Y] = [C][X]
\]

(9.14)

where \([C]\) is an \( L \times L \) matrix of coefficients based on the cosine function and \([X]\) and \([Y]\) are \( L \)-point input or output vectors.

For a specific instance, the matrices and vectors are written out for \( L = 8 \):

\[
\begin{bmatrix}
Y_0 \\
Y_1 \\
Y_2 \\
Y_3 \\
Y_4 \\
Y_5 \\
Y_6 \\
Y_7 \\
\end{bmatrix} =
\begin{bmatrix}
c_4 & c_4 & c_4 & c_4 & c_4 & c_4 & c_4 & c_4 \\
c_1 & c_3 & c_5 & c_7 & -c_7 & -c_5 & -c_3 & -c_1 \\
c_2 & -c_6 & -c_2 & -c_6 & -c_2 & -c_6 & -c_2 & -c_6 \\
c_3 & -c_7 & -c_1 & -c_5 & c_5 & c_1 & c_7 & -c_3 \\
c_4 & -c_4 & -c_4 & c_4 & -c_4 & -c_4 & c_4 & -c_4 \\
c_5 & -c_1 & c_7 & c_3 & -c_3 & -c_7 & c_1 & -c_5 \\
c_6 & -c_2 & c_2 & -c_6 & -c_2 & -c_6 & c_2 & -c_2 \\
c_7 & -c_5 & c_3 & -c_1 & c_1 & -c_3 & c_5 & -c_7 \\
\end{bmatrix}
\begin{bmatrix}
x_0 \\
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5 \\
x_6 \\
x_7 \\
\end{bmatrix}
\]

(9.15)

where \( c_i = \cos(i\theta); i = L/16 \).

This requires \( L^2 = 64 \) multiplications and 64 additions. The matrices and vectors can be decomposed into two \( L/2 \times L/2 \) matrix and vector products to save computation:

\[
\begin{bmatrix}
Y_0 \\
Y_1 \\
Y_2 \\
Y_3 \\
Y_4 \\
Y_5 \\
Y_6 \\
Y_7 \\
\end{bmatrix} =
\begin{bmatrix}
c_4 & c_4 & c_4 & c_4 \\
c_2 & c_6 & -c_6 & -c_2 \\
c_4 & -c_4 & -c_4 & c_4 \\
c_6 & -c_2 & c_2 & -c_6 \\
c_1 & c_3 & c_5 & c_7 \\
c_3 & c_7 & -c_1 & -c_3 \\
c_5 & -c_1 & -c_7 & c_3 \\
c_7 & -c_5 & c_3 & -c_1 \\
\end{bmatrix}
\begin{bmatrix}
x_0 + x_7 \\
x_0 - x_7 \\
x_0 + x_7 \\
x_0 - x_7 \\
x_0 + x_7 \\
x_0 - x_7 \\
x_0 + x_7 \\
x_0 - x_7 \\
\end{bmatrix}
\]

(9.16)

This requires \( 2(L/2)^2 = 32 \) multiplications and \( 2 \times 2(L/2)^2 = 64 \) additions for computation. A flowgraph that results from successive decompositions provides a fast version of this algorithm (Fig. 9–17), as proposed by B. G. Lee.\(^{12}\)

The final of the four DCT implementation approaches uses distributed arithmetic to avoid multiplications. Given that the DCT can be performed as a set of scalar products as shown above, all multiplications can be replaced with addi-
tions. Distributed arithmetic converts the scalar product of the matrix multiply into an efficient form by explicitly treating both the data value \(x\) and coefficient \(c\) as bit-weighted powers of two:

\[
x_i = -x_{i,0} + \sum_{b=0}^{B-1} x_{i,b} \cdot 2^{-b} \tag{9.17}
\]

\[
C_i = -c_{i,0} + \sum_{b=0}^{B_c-1} c_{i,b} \cdot 2^{-b}. \tag{9.18}
\]

The product \(Y = [C] \cdot [X]\) becomes:

\[
Y = \sum_{i=0}^{B_i-1} c_i x_i
\]

\[
= \sum_{i=0}^{B_i-1} \left[ -x_{i,0} + \sum_{b=0}^{B-1} x_{i,b} \cdot 2^{-b} \right] c_i
\]

where \(B\) is the number of bits in the binary representation of \(x\) and \(B_i\) is the number of bits in the binary representation of \(C_i\).

Next, each sum that multiplies \(c_i\) is expanded and grouped as shown [example for term (0)]:

\[
c_0 [x_{0,0} \cdot 2^{-0} + x_{0,1} \cdot 2^{-1} + \ldots + x_{0,B-1} \cdot 2^{-(B-1)}]
\]

(9.20)

to obtain:

\[Y = \sum_{j=0}^{b-1} C_j \cdot 2^{-j}, \]

where

\[C_b = \sum_{i=0}^{B_c-1} c_i \cdot x_{i,b} \tag{9.21}\]
Since $C_j$ has only $2^8$ possible values, which are determined by $x_{ib/r}$, a read-only memory (ROM) is used to store its value, and $B_c \{x_{0,b/r}x_{1,b/r} \ldots x_{b,c}\}$ bits are used to access and retrieve $C_i$ values. Fig. 9–18 shows a circuit to implement distributed arithmetic, in which a ROM stores the appropriate values.

These four structures for real-time implementation of the DCT are compared in Table 9–5.

Having examined various structures for efficient implementation of the DCT, image compression algorithms that use the DCT are now discussed. The DCT is an important component of both the still-frame JPEG (Joint Photographic Experts Group) and MPEG (Motion Pictures Experts Group) standards for image coding.

In the JPEG standard for still pictures, the image is broken into $8 \times 8$ pixel blocks, and the DCT is applied to each block (Fig. 9–19). The DCT results are quantized by spatial-frequency-dependent quantization, which divides each of the 64 ($8 \times 8$) DCT coefficients by a corresponding value in the quantization table. The array of 64 coefficients are then stored in order of increasing spatial frequency, using a zig-zag search that starts with DC (0 frequency), then moves to next-lowest horizontal spatial frequency, then the next lowest vertical, etc. As a result, the coefficients with the highest spatial frequencies are positioned sequentially toward the end of the 64 coefficients. If these values are zero, as they often are as a result of the low-frequency energy packing property of the DCT, they are easily encoded as a string of consecutive zeros using run-length encoding.

For the coding of coefficients across neighboring blocks, the DC coefficient does not vary much, so differential coding is used to transmit the differences between the DC values of successive blocks. The higher-frequency coefficients are encoded using run-length encoding as mentioned above. Huffman coding at the output uses predefined codes based on the statistics of the image, assigning shorter codes to frequently occurring values and longer codes for less frequently occurring values.

![Figure 9–18](image-url) Distributed arithmetic implementation of eight-point DCT replaces multiplication with ROM lookups.© 1995 IEEE.
9.3 Image Compression

9.3.2 Moving Picture Encoding

The coding of image sequences for moving picture transmission includes both JPEG-based still-frame encoding and frame-to-frame coding of object movement. Motion estimation for interframe coding is accomplished by block matching. A frame is divided into blocks. Based on the frame rate, a postulate is made of the maximum number of pixels that an object can move between frames, which is labeled as $w$ pixels in each direction. Block matching is based on a search of $-w$ blocks in all directions on one frame to find the best matching region with the previous frame. A mean absolute distance is used:

$$D(m,n) = \sum_{k=0}^{N-1} \sum_{l=0}^{M-1} |x_{i+1}(k,l) - x_i(k, m, l + n)|$$

$$v = \arg \min_{m,n} [D(m,n)]$$

The search proceeds as shown in Fig. 9–20.

<table>
<thead>
<tr>
<th>Method</th>
<th># Multiplies/Pixel</th>
<th># Adds/Pixel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct</td>
<td>$L^2$</td>
<td>$L^2$</td>
</tr>
<tr>
<td>Separable</td>
<td>$2L$</td>
<td>$2L$</td>
</tr>
<tr>
<td>Fast algorithm</td>
<td>$\log_2L$</td>
<td>$2\log_2L$</td>
</tr>
<tr>
<td>Distributed arithmetic</td>
<td>0</td>
<td>$32/L$</td>
</tr>
</tbody>
</table>

Table 9–5 Comparison of alternative structures for real time implementation of discrete cosine transform for block size of $L \times L$ pixels.\textsuperscript{11}

Figure 9–19 JPEG encoding algorithm performs DCT on $8 \times 8$ subblocks of image, then quantizes and orders the DCT coefficients in order of increasing frequency for final Huffman coding.
To develop an implementation of block matching, a dependency graph of the search operation is generated (Fig. 9–21). In this example, the block size \( L \) is 4 pixels, and the window excursion is \( \pm 1 \) pixel in each direction. Each minimum selection processor \( P_M \) computes at one search position of the window, and each absolute difference processor \( P_{AD} \) computes the absolute difference for a pair of pixels chosen from frame \( i \) and frame \( i + 1 \). The sum over the \( 4 \times 4 \) block is generated by accumulating the individual values across all processors \( P_{AD} \).

This processor arrangement can be made more efficient by exploiting the locality of reference, that is, by taking advantage of the fact that data for adjacent pixel positions has already been used and is in the processor. To do this, a two-dimensional shift register is used, which stores the search window of size \( L(2w + L) \) and can shift the coefficients up/down and right to execute the search. Each processor \( P_M \) checks whether the current distortion \( D(m, n) \) is smaller than the previous distortion value and if so, it updates the \( D_{\text{min}} \) register (Fig. 9–22, right processor).

Processors of type \( P_{AD} \) store \( x_i(k, l) \) and receive the value of \( x_{i+1}(m + k, n + l) \) that corresponds to the current position of the reference block within the search window. \( P_{AD} \) then performs 1) subtraction, 2) computation of absolute value, and 3) addition to partial result coming from the upper processing element. Each \( P_{AD} \) has one register that, when combined with other processors in an array, provides

**Figure 9–20** For block matching motion estimation, a block of size \( L \times L \) in frame \( i \) is translated by an amount of \( w \) pixels in all directions in frame \( i + 1 \), and the position of best match is found and indicated by the motion vector, \( v \).© 1995 IEEE.
Figure 9–21  Dependence graph of block matching algorithm includes an absolute difference processor \( P_{AD} \) for each pixel in the block (shown as circle labeled \( AD \)), and a minimum select processor \( P_{M} \) for each search position of the block (circle \( M \)).11 © 1995 IEEE.

Figure 9–22  Absolute difference processor \( P_{AD} \) includes one element of shift register for storing neighborhood values, double buffer \( x_i \) and \( x'_i \), and absolute value and adder circuits; minimum processor \( P_{M} \) selects and identifies minimum value.11 © 1995 IEEE.
a shift register for elements of the pixel neighborhood, and each processor obtains its particular needed value of $x_{i+1}(m + k, n + l)$. The processors $P_{AD}$ and shift registers $R$ are arranged as shown in Fig. 9–23. This array consists of $L(2w + L)$ processing and storage elements. Data enters serially as a new column of $2w + L$ pixels of the search area, which is stored in the shift registers $R$. The minimum processor $P_M$ at the lower left selects the minimum value of $D$ across the search area.

An MPEG coder (Fig. 9–24a) includes both the still image coder, a decoder in a feedback loop, and the motion estimator described above. In the still image coder, a variable length coder (VLC) follows the DCT. The decoder consists of an inverse quantizer and DCT ($Q^{-1}$ and DCT$^{-1}$). The motion estimation section provides motion compensated prediction and provides the prediction errors to the

Figure 9–23  Two-dimensional processing architecture for block matching includes minimum processor $P_M$, absolute difference processors $P_{AD}$, and shift register $R$.© 1995 IEEE.
DCT coder. The image sequence decoder (Fig. 9–24b) includes the decoding path and motion reconstruction components and includes an inverse variable length coder (VLC⁻¹).

Both still-image and moving-image coders are subject to standards such as JPEG and MPEG. Table 9–6 shows the common still- and moving-image coding standards in use.

In addition to encoding for transmission purposes, video sequence coding for storage purposes introduces the additional requirements of random access, high-speed search, and reverse playback. These functions may be accomplished using a special predictive coding technique (Fig. 9–25). This technique, known as temporal

Figure 9–24 Image sequence encoder (a) and decoder (b) as used in MPEG standard.© 1995 IEEE.
predictive coding, encodes one frame of every $J$ frames and interpolates for the interspersed frames. Frame are encoded according to type $I$ (coded, not predicted), $P$ (predicted frame, every $J$th frame), and $B$ (bidirectionally interpolated frames), which are used to reconstruct the skipped frames between every $J$th frame.

It has been shown here that the structures for signal compression arise from processing to preserve the information content of the signal, which is extracted using the signal analysis computing structures discussed in the previous chapter. Once information is extracted, it may be compressed and then compared to compressed information obtained from other signals. Comparing signals based on their information content forms the subject of the next chapter.

Table 9–6 Standards for still image and video coding.\(^{11}\)

<table>
<thead>
<tr>
<th>Name</th>
<th>Application</th>
<th>Typical Image Format</th>
<th>Coded Bit Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPEG</td>
<td>Still, photo-CD, photo videotext</td>
<td>Any size, 8 bits/pixel</td>
<td>0.25–2.25 bits/pixel</td>
</tr>
<tr>
<td>H.261</td>
<td>Video, videophone, video conference</td>
<td>QCIF, CIF; 10-Hz–30-Hz frame rate</td>
<td>$p \times 64$ Kbit/sec; $1 \leq p \leq 30$</td>
</tr>
<tr>
<td>MPEG1</td>
<td>Video CD-ROM, CD-1, computer</td>
<td>SIF, 25-Hz–30-Hz, CCIR</td>
<td>1.2 Mbit/sec</td>
</tr>
<tr>
<td>MPEG2</td>
<td>Video distribution &amp; contribution</td>
<td>CCIR</td>
<td>4–9 Mbit/sec</td>
</tr>
<tr>
<td>MPEG4</td>
<td>Multimedia video coding—video &amp; audio, object-based coding</td>
<td>CCIR CIF, TV</td>
<td>up to 4 Mbit/sec</td>
</tr>
</tbody>
</table>

Figure 9–25 Sequential frame encoding for video storage to support rapid search and playback.\(^{11}\) © 1995 IEEE.
REFERENCES